CSCI 132 Basic Data Structures

James Goudy

Nov 03, 2022

CONTENTS

This class introduces the JAVA language and basic data structures. Some of the basic data structures include arrays, linked lists, and some sorting.

> "Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live" - John Woods

Part I

Java Language

CHAPTER

ONE

JAVA LANGUAGE

An introduction to the Java Language

Java is C++ without the guns, clubs and knives. *James Gosling*

C makes it easy to shoot yourself in the foot. C++ makes it harder. But when you do, it blows your whole leg off. *Bjarne Stroustrup*

CHAPTER

TWO

NETBEANS INSTALLATION

2.1 Three Main Steps

1. Install Open JDK
2. Set the Environmental Variables in Windows
3. Install Apache NetBeans.

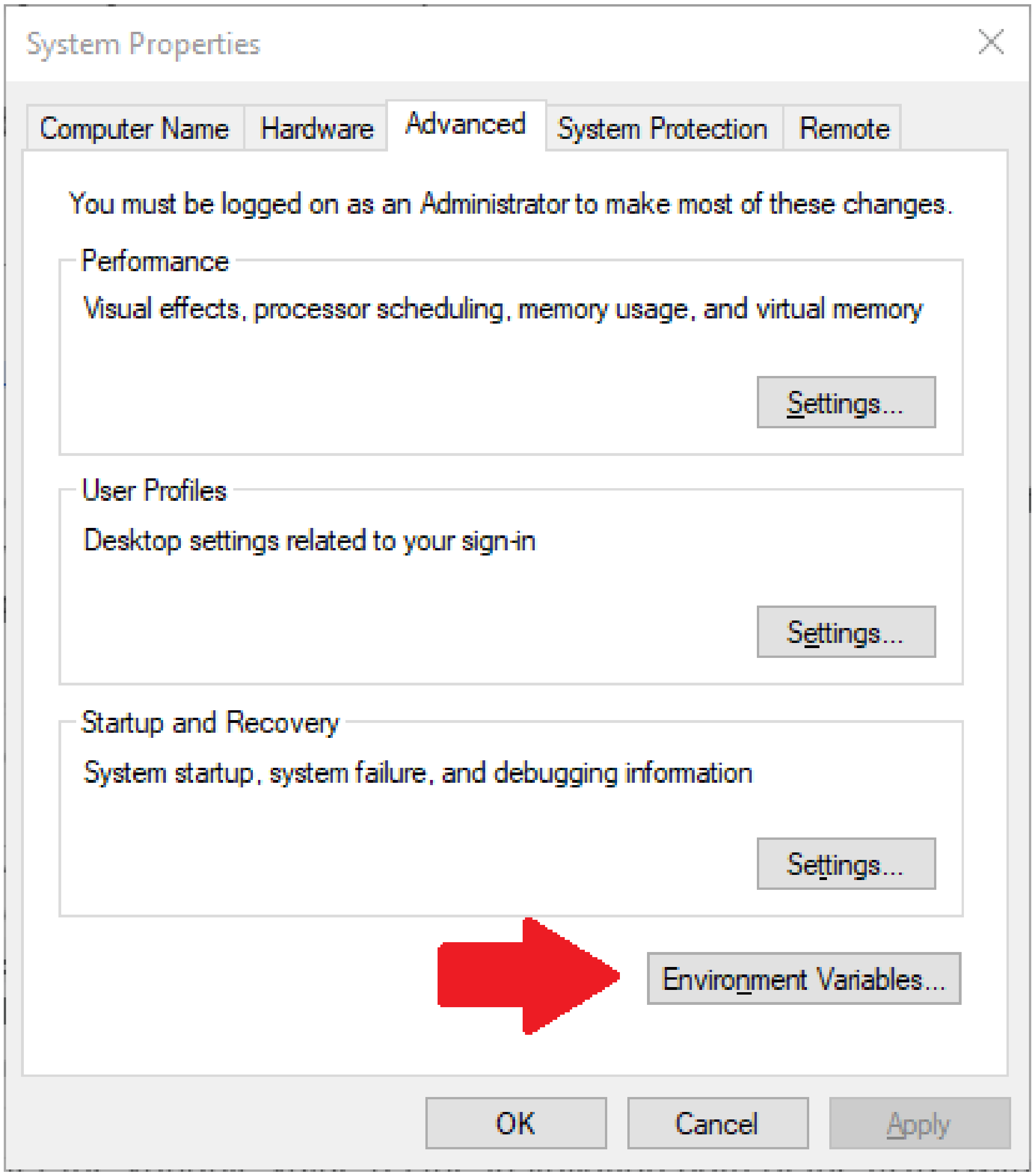
   For fresh installs, it has to be done in this order.

2.2 Install Open JDK

1. Go to https://openjdk.java.net/Click on the version you wish to install.
   1. *Uncompress the zip or gz file*. For Windows, copy the java-xx file to c:\Program Files in a folder called java. **C:\Program Files\Java\java-xx**.

2.3 Set the Environmental Variables in Windows

1. Depress the Windows Key and release then type ENV. This will open the window to change the environmental variables.



You must be logged on as an Administrator to make most of these changes.

Performance

Visual effects, processor scheduling, memory usage, and virtual memory

User Profiles

Desktop settings related to your sign-in

Startup and Recovery

System startup, system failure, and debugging information

1. Set a PATH: Select Control Panel and then System.
   1. Click Advanced and then Environment Variables.
   2. Add the location of the **bin folder** of the JDK installation to the PATH variable in System Variables.
   3. Click **Path** in the System Variables (bottom half), click **edit**, and **browse**. Browse to the **bin** folder and click **ok**."C:\Program Files\Java\jdk-xx\bin"
2. Set JAVA_HOME:

1. Under System Variables, click New.
2. Enter the variable name as **JAVA_HOME**.

Warning: JAVA_HOME has to be in all caps.

```
:::

  3. Enter the variable value as the installation path of the JDK (without the bin␣
↪sub-folder)
  4. Click OK.
  5. Click OK.
```

2.4 Install Apache NetBeans

1. Go to https://netbeans.apache.org/
2. Click the download button.
3. Install

Updated May 2022

CHAPTER

THREE

JAVA INTRODUCTION

3.1 Key Ideas

- History
- Program Features
- Integrated Development Environments

3.2 Readings

https://books.trinket.io/thinkjava2/chapter1.html

History of Java

Language Ranking

3.3 History

- Created by James Gosling and a team of developers.

- Released in the fall of 1995Created applets – run on web pages.

- Was controlled by Sun Microsystems until 2010 then bought out by Oracle.

3.4 Program Features

- Object-oriented programming Platform Neutral
- Java programs are transformed into a format called **bytecode** which can be run by any computer or device running a java virtual machine
- Java automatically takes care of memory allocations and deallocations. (automatic garbage collection)
- Java does not include pointers.
- Java includes only single inheritance.

3.5 IDE – Integrated Development Environment

3.5.1 NetBeans

- First supported by Sun then by Oracle
- Now part of the Apache Software Foundation (2016)
- Used by lots of corporations
- https://netbeans.apache.org/

3.5.2 Eclipse

- open source
- uses lots of plugins
- first to the game hence a little larger market share
- https://www.eclipse.org/ide/

3.5.3 Others

- JDeveloper
- IntelliJ Idea

End Of Topic

CHAPTER

FOUR

JAVA BASICS

4.1 Key Topics

- Comments
- Variables
- Data types
- Operators
- Order of Operations
- Incrementors and Decrementors
- Escape Characters
- Comparative Operators
- Concatenation

CHAPTER FIVE

READINGS

https://books.trinket.io/thinkjava/chapter1.html

5.1 Comments

Definition

Comment allows the programmer to document their code. It is not compiled.

5.1.1 Examples

```
// This is a single line comment

/*
This is
a multiline
comment
*/
```

5.2 Variables

Definition

Variable stores one item of information.

5.2.1 Naming Variables

- Start with a letter, underscore or a dollar sign
- Can not start with an number
- **NO SPACES**
- CASE SENSITIVE
  - All different are different variable names: fname, Fname, fName, FNAME, fnamE
- Java as a whole is case sensitive

Declaring A Variable

datatype variableName; *datatype* variableName = value;

5.2.2 Examples

```java
// declaring variables
int Age;
String First_Name;
boolean Breathing;

//declaring variables and intializing them with a value
int numberOfStates = 50;
double salary = 50000;

// note that strings get double quotes
// chars get single quotes
String dogName = "Spot";
char middleInital = 'A';

boolean isBreathing = true;
```

Warning: A variable is usually declared once within a set of braces{}.

5.2.3 Data Types

Data Type	Memory Space	Max Values
Integer Types - whole numbers		
byte8	bits - 1 byte	-128 to 128
short	16 bits – 2 bytes	-32,768 to 32,767
int	32 bits – 4 bytes	-2,147,483,648 to 2,147,483,647
long	64 bits – 8 bytes	(+-)9,223,372,036,854,775,808)
Real / Floating Point		
float	32 bits - 4 bytes	1.4 E-45 to 3.4 E+38
double	64 bits - 8 bytes	4.9 E -324 to 1.7 E 308
Other		
boolean		true / false - Use as flags
char		Individual letters or characters
Acts Like A Variable		
String		~ 2 billion characters
Object		Store binaries - “anything”
var		limited use within functions when datatype is not known

Note: for precise decimals i.e. currency – use java.math.BigDecimal – reason – rounding accuracy

5.3 Operators

Sign	Operations
+	Addition
-	Subtraction
*	Multiplication
/	Divide
%	Modulus (return the remainder of an integer)

5.4 Order Of Operations

- Increment and decrement
- Parentheses ()
- Exponents
- * /
- + -

- Comparisons > < >= <= == !=
- Logical Operations && ||
- Assignment Expressions

5.5 Incrementors and Decrementors

- use ++ to add the number 1 to a variable.
- use – to subtract the number one from a variable.

Examples

```
x++;
age++;

y--;
salary--;
```

5.6 Escape Characters

Definition

Escape Character is a backslash \ followed by the character you want to insert into your for printing or inserting into a string.

Escape Character	Definition
\n	new line
\t	tab
\b	backspace
\r	carriage return
\f	form feed
\	backslash
‘	single quotation mark
\”	double quotation mark
\d	octal
\xd	hexadecimal
\ud	Unicode character

5.7 Comparative Operators

Definition

Comparative Operators evaluate is something is true or false.

Operator	Definition
==	Equal to x==y; (NOT x=y)
!=	Not Equal to x != y;
<	Less Than x < y;
>	Greater Than x>y.
<=	Less Than or Equal to x <= y ; (no space between the signs)
>=	Greater Than or Equal to x >=y;

5.8 Logical Operators

Definition

Logica Operators connect two or more expression together and evalute to true or false.

Operator	Definition
&&	AND (both sides must be true to evaluate to true)
||	OR (only one side must be true to evaluate to true)

5.8.1 Truth Table

(x < y && z > 10)

expression 1x<y	operator	expression 2z >10	outcome
AND			
true	&&	true	true
true	&&	false	false
false	&&	true	false
false	&&	false	false
OR			
true	||	true	true
true	||	false	true
false	||	true	true
false	||	false	false

5.9 Concatenation

Definition

Concatenation is the action of putting strings togeher to form a new string.

5.9.1 Example

```
String fName = "Bubba";
String lName =  "Smith";
String fullName = fName  + " " + lname;
// Note we concantenate a space " " between the first and last name
```

End Of Topic

CHAPTER

SIX

VARIABLE LECTURE CODE

6.1 Key Ideas

- Printing
- Getting Information From The Console / Keyboard

6.2 Readings

https://books.trinket.io/thinkjava2/chapter2.html

https://books.trinket.io/thinkjava2/chapter3.html

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/variables.html

https://jenkov.com/tutorials/java/variables.html

6.3 Printing

To print to the console, the following command is used:

System.out.println("your text here or variable here");

System.out.**println**() will drop the cursor to the next line.

System.out.**print()** will leave the cursor on the same line at the end of the text.

6.4 Getting Input From The Console

- The library / class *java.util.Scanner* needs to be imported.
- A scanner is created
- Input from the scanner is stored in a variable
  - By default, all data from a scanner is a String datatype
  - To convert it into an Integer or Double use the parse command;

Example

```java
import java.util.Scanner;

public class MyProgram
{
    public static void main()
    {
        double salary = 0.0;
        int age = 0;
        String name = "";

        //create the scanner
        Scanner myScan = new Scanner(System.in);

        System.out.print("Enter your name: ");
        salary = myScan.nextLine();

        System.out.print("Enter your salary: ");
        salary = Double.parseDouble(myScan.nextLine());

        System.out.print("Enter your age: ");
        age = Integer.parseDouble(myScan.nextLine());

    }


}
```

Example 2

```java
/*
 Lecture Code Class
 Topic: Variables
 */
package j1_20150902;

import java.util.Scanner;

public class J1_20150902 {

    public static void main(String[] args) {

        //put variables here
        //Integer datatypes
        int age = 20;
        int NumberOfDogs;
        short dogage = 7;
        long geologicalyears = 200000000;

        //Real Numbers
        float Salary = 50000;
        double SpaceMiles = 0;
        float dogYears;
        float humanYears;

        //Byte
        byte Codingblock = 64;

```

(continues on next page)

(continued from previous page)

```
        //Boolean
        boolean IsItRaining = true;
        boolean AmIHungry = false;

        //Char
        char MiddleInitial = 'C';
        char Gender = 'M';

        //Strings
        String FirstName;
        String LastName = "Smith";
        String Fruit;

        //Constants - values that will not change
        //The practice is to put constants in
        //ALL CAPS.
        final double PI = 3.14159265359;
        final String MYTITLE = "Code Guru";

        //----------------Scanner-------------------------
        /*
         Scanners are used to read information from the keyboard.
         Scanner is the key word.  "scan" is a name designated by the programmer.
         "scan" could be called anything the programmer wishes to name it.
         "new Scanner(System.in)" is the command to create an object that will
         read keystrokes from the keyboard.
         */
        Scanner scan = new Scanner(System.in);


        //Ask a question or present information to user
        //"System.out.println("your text")" writes a message to the screen
        System.out.println("Please enter your first name");

        //"scan nextline will read kestrokes until enter key is depressed.
        //All keystrokes that are read from the keyboard using "nextline"
        //reads that data/keystrokes as the String datatype
        FirstName = scan.nextLine();
        System.out.println("Hello " + FirstName);

        //Get an age
        System.out.println("Please enter your age");
        age = Integer.parseInt(scan.nextLine());
        System.out.println("Your age is " + age);

        /*
         scan.nextDouble()
         scan.nextInteger()
         scan.nextFloat()
         scan.nextByte()
         Using these, the scanner will read in only the numbers but leave the
         "enter" keystroke in the buffer. If the next scanner command is
         "nextLine()", it will immediately see the "enter" keystroke that was
         in the buffer and not allow the user to enter any data. This is
         demonstrated by the following code.
         */
```

(continues on next page)

(continued from previous page)

```
        //--------  using nextInteger example ---------------------
        System.out.println("\n\n***** nextInteger Example ****");
        System.out.println("Please enter your age");
        age = scan.nextInt();
        System.out.println("Your age is " + age);

        System.out.println("Please enter your favorite fruit");
        Fruit = scan.nextLine();  //nextline sees the enter from nextInt and
                                  //executes the line immediately - not
                                  //allowing the user to enter a fruit.
        System.out.println("Your favorite fruit is" + Fruit);


        //--------  Fixing the  nextInteger previous example -------------------
        System.out.println("\n\n***** nextInteger Example ****");
        System.out.println("***** The Fix ****");
        System.out.println("Please enter your age");
        age = scan.nextInt();
        scan.nextLine(); //This will clear the "enter" keystroke from the buffer
        System.out.println("Your age is " + age);

        System.out.println("Please enter your favorite fruit");
        Fruit = scan.nextLine();  //The user can now enter a fruit.
        System.out.println("Your favorite fruit is " + Fruit);

        /*
        It is better to use the following:
        yourVariableName = Double.parseDouble(scannerName.nextLine())
        yourVariableName = Integer.parseInt(scannerName.nextLine())
        yourVariableName = Float.parseFloat(scannerName.nextLine())
        */

        //Calculating dog years
        System.out.println("\n\nWe are going to calculate dog years");
        System.out.println("Enter your human age");
        humanYears = float.parseFloat(scan.nextLine());
        dogYears = humanYears / 7.0f;
        System.out.println("Your age in dogyears is " + dogYears);

        //Exit program
        System.out.println("\n\nBye Bye");

    }

}
```

End Of Topic

CHAPTER

SEVEN

FIRST PROGRAM

7.1 Key Ideas

- Declaring variables
- Creating a scanner
- Printing to the console
- Storing information from the keyboard to a variable
- Casting data from a String to a number

7.2 Readings

https://books.trinket.io/thinkjava2/chapter3.html

Programs are written functions/methods. In the example below there is only one function called *main()*.

Definition

main() is the function where the program starts and finishes.

Print to the console, the following command is used:

-
  ```
  System.out.println("your text here or variable here");
  // Variable names do not get quotes around them.
  ```
- System.out.**println()** will move the cursor to the next line.
- System.out.**print()** will leave the cursor on the same line at the end of the text.

Get Input From The Console

- The library / class *java.util.Scanner* needs to be imported.

  ```
  import java.util.Scanner;
  ```

- Create scanner

```
Scanner myScan = new Scanner(System.in);
// NOTE: myScan is your name for the scanner
// You can name it anything you want as long as it
// follows the rules for variable names.
```

- Input from the scanner is stored in a variable
  - By default, all data from a scanner is a String datatype
  - To convert it into an Integer or Double use the parse command;

```
System.out.print("Please enter your first name: ");
firstname = myScan.nextLine();

System.out.print("Please enter your salary: ");
firstname = Double.parseDouble(myScan.nextLine());
```

Warning: Do not use .nextDouble(), nextInt() without using a .nextLine() to ensure the "return" is removed from the keyboard buffer.

Programming Pattern

Notice the programing pattern:

1. Asks for information from the user with a System.out.print().
2. Collect the information from the user using a scanner and store it in a variable.
3. Calculate the data
4. Return an answer using the System.out.print().

Note: The filename of the project is always the name of the public class. In the example below, the project name is J1_FirstProgrA

```
/*
Names:  Jim Goudy
Program: First_Program

Note: When you are naming projects - do not use spaces.
 */

import java.util.Scanner;

public class J1_FirstProgrA {

    public static void main(String[] args) {

        //variables
        String xFirstName, xLastName;
        String xCity;
        String xNickName = "Ramone";

```

(continues on next page)

(continued from previous page)

```
        double xInches = 0;
        double xAnswer = 0;
        double xCentimeterss = 0;
        double xNum1, xNum2, xNum3;

        //Create a scanner
        Scanner myScan = new Scanner(System.in);

        //Create a greeting
        System.out.print("Please enter"
                + " your name: ");
        xFirstName = myScan.nextLine();

        System.out.println("Enter your city");
        xCity = myScan.nextLine();

        System.out.println("\n\nGreetings " + xFirstName + " from " + xCity);

        //convert Inches To Centimeters
        System.out.print("\nPlease \nenter a \nmeasurment in inches : ");
        xInches = Double.parseDouble(myScan.nextLine());
        xAnswer = xInches * 2.54;
        System.out.println("The is answer is" + xAnswer);

        //Centers To Inches  - divide 2.54
        System.out.print("\n\nPleaseenter a measurment in centimeters : ");
        xCentimeterss = Double.parseDouble(myScan.nextLine());
        xAnswer = xCentimeterss / 2.54;
        System.out.println("The is answer is" + xAnswer);

        //Add three numbers
        System.out.println("Add three numbers\nEner the first number");
        xNum1 = Double.parseDouble(myScan.nextLine());

        System.out.println("Ener the second number");
        xNum2 = Double.parseDouble(myScan.nextLine());

        System.out.println("Ener the third number");
        xNum3 = Double.parseDouble(myScan.nextLine());

        xAnswer = xNum1 + xNum2 + xNum3;

        System.out.println("\nThe sum of three numbers is " + xAnswer);

        /*
        //DO NOT USE THIS IN JIM'S CLASS!!!!!!!!!!!!!!!!!!!!
        System.out.println("Enter a number");
        xInches = myScan.nextDouble();
        System.out.println("Enter your FirstName xxxxx- ");
        xFirstName = myScan.nextLine();

        NOTE: Suppose the user enters the number 42.  They literally press the
        "4" key, the "2" key and the "enter" key. Because the
        "myScan.nextDouble()" only takes the numbers, it does not
        take the enter key - which is left in the keyboard buffer.
        nextLine() excutes when it sees an enter key in the buffer.  Since the
```

(continues on next page)

(continued from previous page)

```
        nextDouble() does not clear the enter key, it is still left over for
        the nextLine(), which will excecute immediately and not allow the user
        to enter their name.
        */

        //Message to let the user know the program is finished.
        System.out.println("Thank you for using my software");

    }
}
```

End Of Topic

CHAPTER

EIGHT

SECOND PROGRAM

8.1 Observations

Note the following:

- The programmer name and project title are in the comments.
- We can put the project requirements in the comments as well. In bigger projects this would be a separate document.
- Variables are declared at the top of the function - **main()** . When we have more functions the variables will always go at the top, with the exception of **var**.
- Note the programming pattern
  - Ask a question

    ```
    System.out.print("Question?");
    ```

  - Store the response in a variable using a scanner
  - Make a calculation
  - Show the answer using another

    ```
    System.out.print("Answer is " + x);
    ```

    * Note the commenting.

8.2 Lecture Code

```
/*
Name: James Goudy
Project: Example 2

Program requirements
Greeting: Name and major

Problem 1 : Calculate the circumference of a circle
Formula:    C = πRD
            C - circumference
            π - pi 3.14
            R - Radius
```

(continues on next page)

(continued from previous page)

```
            D - Diameter

Problem 2 : Calculate the volume of a prism
Formula:    V = lwd
            V - Volume
            l - length
            w - width
            d - depth

Exit Message

 */


import java.util.Scanner;

public class JavaIntro_ExampleProgram2 {

    public static void main(String[] args) {

        //Greeting Variables
        String xName;       //variable to hold name
        String xMajor;      //variable to hold major

        //Problem 1 varialbes
        double xCircumference = 0;
        final double xPI = 3.14;
        double xRadius = 0;
        double xDiameter = 0;

        //Problem 2 variables
        double xVolume = 0;
        double xLength = 0;
        double xWidth = 0;
        double xDepth = 0;

        //Create a scanner to collect keyboard information
        Scanner myScanner = new Scanner(System.in);

        // -------------------------------------------------------------
        //Greeting
        System.out.println("Please Enter your name: ");     //ask the question
        xName = myScanner.nextLine();                       //collect the answer

        System.out.println("Please enter your major: ");    //ask the question
        xMajor = myScanner.nextLine();                      //collect the answer

        //Display the greeting
        System.out.println("Hello " + xName + " from " + xMajor);

        // ------------------------- Problem 1----------------------------
        System.out.println("\nProblem 1 - Circumference");

        System.out.print("\nPlease enter the radius: ");      //ask the question
        xRadius = Double.parseDouble(myScanner.nextLine()); //collect the answer
```

(continues on next page)

(continued from previous page)

```
        System.out.print("\nPlease enter the diameter: ");      //ask the question
        xDiameter = Double.parseDouble(myScanner.nextLine()); //collect the answer

        //calculate the answer
        xCircumference = xPI * xRadius * xDiameter;

        //Display the answer
        System.out.println("\nThe circumerence is " + xCircumference);

        // ------------------------- Problem 2----------------------------
        System.out.println("\nProblem 2 - Volume of a prism");

        System.out.print("\nPlease enter the length: ");      //ask the question
        xLength = Double.parseDouble(myScanner.nextLine()); //collect the answer

        System.out.print("\nPlease enter the width: ");      //ask the question
        xWidth = Double.parseDouble(myScanner.nextLine()); //collect the answer

        System.out.print("\nPlease enter the depth: ");      //ask the question
        xDepth = Double.parseDouble(myScanner.nextLine()); //collect the answer


        //calculate the answer
        xVolume = xLength * xWidth * xDepth;

        //Display the answer
        System.out.println("\nThe volume is " + xVolume);

        //------------------------- Exit Messasge
        System.out.println("\nThank you for using Goudy Software");

    }

}
```

End Of Topic

CHAPTER NINE

FUNCTIONS

9.1 Key Ideas

- Definition
- Readings
- Examples

9.2 Readings

https://books.trinket.io/thinkjava2/chapter4.html

https://www.geeksforgeeks.org/methods-in-java/?ref=gcse

9.3 Definition

It is important to segment the code, break it into chunks. This is done by creating **functions**. Using functions have the following advantages:

- **Reusability** - a function will allow code to be reused when needed.

Helpful Tip

If you have coded the same thing more than once, that code should be turned into a function.

- **Organization** - It allows large complex programs to be broken down into smaller steps.
- **Testing** - Functions can be easily tested for errors.

Tip: The term functions and methods are usually interchangeable. The do the same thing. However, the term method is usually applied to a function in a class that has a public interface - meaning it can be called by the created object of that class.

9.4 Concepts

- Function form that takes parameters and returns a variable / data
- Function form that takes parameters and returns nothing (displays answer in function)
- Function form that takes no parameters and returns a variable / data
- Function form that takes no parameters and returns nothing (does everything within the function)

Structure Of A Function

Function

```
static datatype functionName(arguments)
{
        // if the function does a calculation
        // the datatype will always match the
        // answer of that calculation

        // datatypes are int, long, float, double
        // boolean,chr, String, Object
        // void is used when the function does
        // return anything to whatever other function called it

        // arguments are the values sent
        // to the function - they are optional

        //code goes here


        // if there is a datatype define then
        // then there will always be a return
        // with a variable of the same datatype
        return aVariable;
}

static void main()
{
        //call the function
        functionName(arguments)
}
```

Functions are always called by other functions, including main().

Suppose a programmer wants to write a function to convert inches to feet. There are four ways to write the function pending if the function is supplied arguments or not and whether the function or the main program displays the answer.

Options	With Arguments(arguments)	Without Arguments () - empty
Return answer	static double inchesFeet(double inches){ double ans = 0; ans = inches / 12.0; return ans;}	static double inchesFeet(){ double ans = 0; inches = 0; System.out.print("Enter inches: "); inches = Double.parseDouble(myScanner.nextLine()); ans = inches / 12.0; return ans;}
Display answer	static void inchesFeet(double inches){ double ans = 0; ans = inches / 12; System.out.print("Answer is "+ ans);}	static double inchesFeet(){ double ans = 0; inches = 0; System.out.print("Enter inches: "); inches = Double.parseDouble(myScanner.nextLine()); ans = inches / 12.0; System.out.print("Answer is "+ ans);}

Example

```
static double inchesFeet()
{
    double ans = 0;
    inches = 0;

    System.out.print("Enter inches: ");
    inches = Double.parseDouble(myScanner.nextLine());

         ans = inches / 12.0;
    System.out.print("Answer is "+ ans);
}
```

Example

```
 /*
    Programmer: James Goudy
    Project: Function Lecture Code
 */


import java.util.Scanner;

public class Function_Demo_Rev1
{

    //functions can here
    //Add two numbers - Functions requires 2 numbers
    //and returns the total
    static double add2nums_a(double num1, double num2)
    {
        //Note - double num1 and double num2 act as if
        //they are within the braces

        //declare variables
        double ans;

        ans = num1 + num2;

        return ans;
    }

    //add two numbers B
```

(continues on next page)

(continued from previous page)

```
    //Add two numbers - Functions requires 2 numbers
    //returns nothing, and displays the answer
    static void add2nums_b(double num1, double num2)
    {
        double ans;

        ans = num1 + num2;

        System.out.println("\n\nB The answer for add2nums_b is " + ans);
    }

    //add two numbers C
    //Add two numbers - the asks for the numbers
    //and returns the answer
    static double add2nums_c()
    {
        //variabls
        double ans;
        double num1 = 0;
        double num2 = 0;

        //create a scanner
        Scanner myScan = new Scanner(System.in);

        System.out.println(" C Please enter first number");
        num1 = Double.parseDouble(myScan.nextLine());

        System.out.println("C Please enter second number");
        num2 = Double.parseDouble(myScan.nextLine());

        ans = num1 + num2;

        return ans;
    }

    //add two numbers d
    //Add two numbers - the function does everything
    //Function asks for the numbers and displays the answer
    static void add2nums_d()
    {
        double ans;
        double num1 = 0;
        double num2 = 0;

        //create a scanner
        Scanner myScan = new Scanner(System.in);

        System.out.println(" D Please enter first number");
        num1 = Double.parseDouble(myScan.nextLine());

        System.out.println("D Please enter second number");
        num2 = Double.parseDouble(myScan.nextLine());

        ans = num1 + num2;

        System.out.println("\n\nB The answer for add2nums_d is " + ans);
```

(continues on next page)

(continued from previous page)

```
    }

    // -------------- InClass Dogyears ----------------
    static double dogYears_a(double humanYears)
    {
        double ans = 0;

        ans = humanYears * 7;

        return ans;
    }

    static void dogYears_d()
    {
        double ans = 0;
        double humanYears = 0;

        //create a scanner
        Scanner myScan = new Scanner(System.in);

        System.out.println(" Please enter human years");
        humanYears = Double.parseDouble(myScan.nextLine());

        ans = humanYears * 7;

        System.out.println("DogYears D - Dogs years is equal to " + ans);
    }


    // ------------- Greetings A --------------------------
    static String Greetings_A(String firstName, String lastName, int PlayerNum)
    {
        String ans = "";

        ans = "Welcome " + firstName + " " + lastName + " player " + PlayerNum;

        return ans;
    }



    public static void main(String[] args)
    {
         //variables
        double xNum1 = 0;
        double xNum2 = 0;
        double xAns = 0;

        double xhumanYears = 0;

        String xFirstName, xLastName, xAnsString;
        int xPlayerNumber = 0;



        //create a scanner
```

(continues on next page)

(continued from previous page)

```
        Scanner myScan = new Scanner(System.in);

        //Base Code
        System.out.println("Add two numbers");
        System.out.println("Please enter first number");
        xNum1 = Double.parseDouble(myScan.nextLine());

        System.out.println("Please enter second number");
        xNum2 = Double.parseDouble(myScan.nextLine());

        xAns = xNum1 + xNum2;

        System.out.println("The answer is " + xAns);

        //Example A
        System.out.println("\n\n--------- Example A ---------\n\n");

        System.out.println(" A Please enter first number");
        xNum1 = Double.parseDouble(myScan.nextLine());

        System.out.println("A Please enter second number");
        xNum2 = Double.parseDouble(myScan.nextLine());

        xAns = add2nums_a(xNum1, xNum2);

        System.out.println("A The answer for add2nums_A is " + xAns);

        //Example B
        System.out.println("\n\n--------- Example B ---------\n\n");

        System.out.println(" B Please enter first number");
        xNum1 = Double.parseDouble(myScan.nextLine());

        System.out.println("B Please enter second number");
        xNum2 = Double.parseDouble(myScan.nextLine());

        add2nums_b(xNum1, xNum2);

        //EXample C
        System.out.println("\n\n--------- Example c ---------\n\n");

        xAns = add2nums_c();

        System.out.println("C The answer for add2nums_c is " + xAns);

        //Example D
        System.out.println("\n\n--------- Example d ---------\n\n");

        add2nums_d();

        // ************** Inclass *******************************
        // Inclass - Dog Years
                  // in the style of A - parenthesis are have arguments return answer
        // in the style of D - parenthesis are empty - function does everything

        // ----------------- Greeting_A ---------------------
```

(continues on next page)

(continued from previous page)

```
        System.out.println("\n\nGreeting Please enter your first name");
        xFirstName= myScan.nextLine();

        System.out.println("Please enter your last name");
        xLastName = myScan.nextLine();

        System.out.println("Please enter your player number");
        xPlayerNumber = Integer.parseInt(myScan.nextLine());

        xAnsString = Greetings_A(xFirstName, xLastName, xPlayerNumber);

        System.out.println(xAnsString);

        // -------------- End Of Program ----------------------------

        System.out.println("Thank you for using Goudy Software - good bye");
    }

    //functions can also go here
}
```

End Of Topic

CHAPTER

TEN

DECISION TREES

10.1 Key Ideas

If Statements Switch Statements String Functions

10.2 Readings

https://books.trinket.io/thinkjava2/chapter5.html

https://www.geeksforgeeks.org/decision-making-javaif-else-switch-break-continue-jump/?ref=lbp

10.3 Observations

Definition

The is statement has the following form:

if(condition){

}

```
        if(condition)
        {
                    // runs if true
        }

        if(condition)
        {
                    // code here runs if true
        }
        else
        {
                    // code here runs if false
        }

        if(condition)
        {
                    // code here runs if true
```

(continues on next page)

(continued from previous page)

```
        }
        else if(condition)
        {
                    // code here runs if true
        }
        else if(condition)
        {
                    // code here runs if true
        }
        else
        {
                    // code here runs if false
                    // the else is optional
        }
```

Warning: In order to compare Strings, you must use **.equal(argument)**

```
String string1 = "aaa";
String string2 = "bbb";

if (string1.equals(string2))
{
    // Do something
}
```

Tip: Note in the code below how a string is converted to an integer - *Integer.parseInt()* and to a double *Double.parseDouble()*

10.4 Lecture Code

```
//James Goudy
//If demo


import java.util.Scanner;

public class If_Demo {

    static void if_one(double num) {

        //Example is checking for true only
        System.out.println("Example 1  \n");

        if (num < 100) {
            System.out.println(num + " is less than 100 \n");
        }
```

(continues on next page)

(continued from previous page)

```java
    }

    static void if_two(double num) {

        //This example is checking for both true and false
        System.out.println("Example 2  \n");

        if (num < 100) {
            System.out.println(num + " is less than 100 \n");
        } else {
            System.out.println(num + " is greater than 100 \n");
        }

    }

    static void if_three(double num) {

        //This example is an  example of an else if statement
        System.out.println("Example 3  \n");

        if (num < 10) {
            System.out.println(num + " is less than 10 \n");
        } else if (num < 20) {
            System.out.println(num + " is less than 20 \n");
        } else if (num <= 30) {
            System.out.println(num + " is less than 30 \n");

        } else {
            System.out.println(num + " is greater than 30 \n");
        }

    }

    static void if_range(double salary) {

        //check to see if a number is within a range of numbers
        System.out.println("Example Range of Numbers  \n");

        if (salary > 0 && salary <= 50000) {
            System.out.println(" You are poor \n");
        } else if (salary > 50000 && salary <= 200000) {
            System.out.println(" You are middle class \n");
        } else {
            System.out.println(" You are rich \n");
        }

    }

    static void if_string(String vehicle) {
        //you should always convert your comparasion to
        //a consistent state.  Comparisons for strings are
        //case sensitive.

        vehicle = vehicle.toLowerCase();

        //String is an object so Java requres
```

(continues on next page)

(continued from previous page)

```
        //the  .equals to compare the equality of
        //strings.  In other language you may
        // be able to use the  ==

        if (vehicle.equals("boat")) {
            System.out.println("You are a boater \n");

        } else if (vehicle.equals("car")) {
            System.out.println("You are a driver \n");
        } else if (vehicle.equals("plane")) {
            System.out.println("You are a pilot \n");
        } else {
            System.out.println("That wasn't a choice \n");
        }

    }

    public static void main(String[] args) {

        //variables
        double num1 = 0;
        double mySalary = 0;
        String myVehicle;

        //create scanner
        Scanner sc1 = new Scanner(System.in);

        //input number
        System.out.println("Enter a number");
        num1 = Integer.ParseInt(sc1.nextLine());

        //if example one
        // if the number is less than 100 the message will appear
        System.out.println("If Example - checking for true only\n");
        if_one(num1);

        //if example two
        //this example for both a true and false condition
        System.out.println("If Example Two - "
                            + "checking for a both true and false\n");
        if_two(num1);

        //if example three
        //this example an else if example
        System.out.println("Else if example - "
                            + "checking for multiple conditions \n");
        if_three(num1);

        //if example - check for a number within a range
        System.out.println("Checkin to see if a number is within a range\n");
        System.out.println("Enter your salary\n");
        mySalary = Double.ParseDouble(sc1.nextLine());
        if_range(mySalary);

        //An example using a string
        System.out.println("This is an example of comparing strings \n");
```

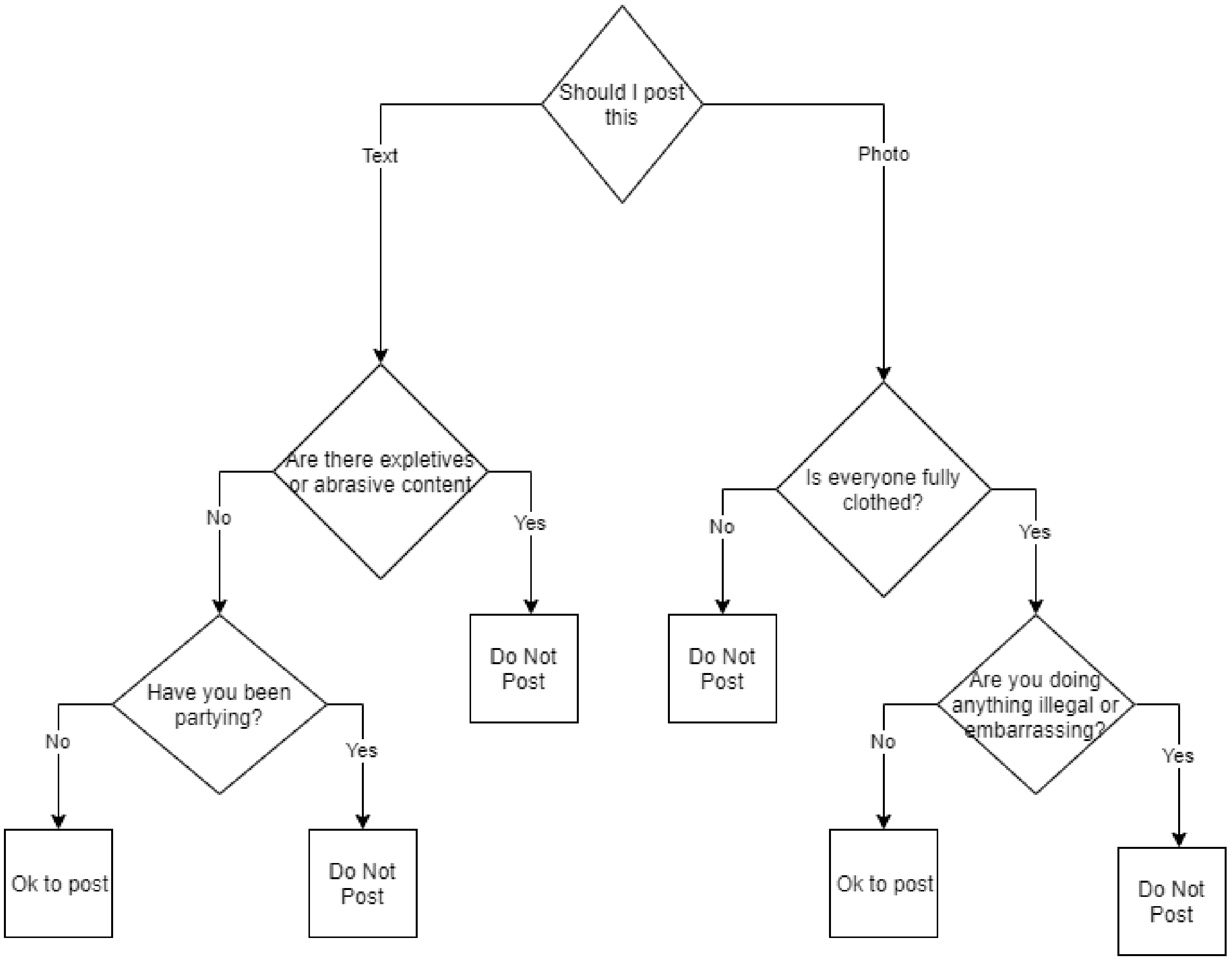
(continues on next page)

(continued from previous page)

```
        System.out.println("Please enter your favorite "
                            + "vehicle - boat, car, plane \n");
        sc1.nextLine();

        myVehicle = sc1.nextLine();
        if_string(myVehicle);

        //exit program
        System.out.println("Press enter to exit \n");
        sc1.nextLine();

    }

}
```

10.5 In Class

Post This Exercise



End Of Topic

CHAPTER

ELEVEN

SWITCH STATEMENT

11.1 Key Ideas

- Switch Statement

11.2 Readings

If / Switch

Swtich Statement

11.3 Lecture Code

Definition

A switch statement acts like an if statement, but there are definite choices. The switch statement will allow specific code to run pending the choice. Swtich statements are usually used in building menus, but can be used anywhere where there is a definite choice.

11.3.1 Example

```java
/* Switch / Menu Demo
This is a demo program to demonstrate a design pattern for creating
a menu system in the console environment.
:)---- Bob Ross Coding ---- :)
Jim Goudy
*/


import java.util.Scanner;
public class Menu_Demo {

    // Happy Function 1
    static void HappyFunction1() {
            System.out.println("\n\nThis is happy function 1");
```

(continues on next page)

(continued from previous page)

```
    }

    // Happy Function 2
    static void HappyFunction2() {
            System.out.println("\n\nThis is happy function 2");
    }

    // Happy Function 3
    static void HappyFunction3() {
        System.out.println("\n\nThis is happy function 3");
    }

    // This function creates a menu system.
    static void menu() {

        // variables
        int choice = 0; // hold user choice

        // Scanner for user input
        Scanner myScan = new Scanner(System.in);

        // display the user choices
        System.out.println("\nMENU\n"
        + "\n1. Happy Function 1"
        + "\n2. Happy Function 2"
        + "\n3. Happy Function 3"
        + "\nPlease choose a function 1,2 or 3");
        choice = Integer.parseInt(myScan.nextLine());

        // take the choice stored in choice and use it in the switch
        // case values can be an int, char or enum
        switch (choice) {
                case 1:
        HappyFunction1();
                break;
        case 2:
        HappyFunction2();
                break;
        case 3:
        HappyFunction3();
                break;
        default:
            // This is optional
            // default is used if the user entered any
            // number that was note 1,2 or 3
            System.out.println("\nThat wasn't a choice");
        }
    }

    public static void main(String[] args) {

        // variables
        String quit = "n";

        // Scanner for user input
        Scanner myScan = new Scanner(System.in);
```

(continues on next page)

(continued from previous page)

```
        // quit has to be initialized with the value of n
        // so the loop will run at least once. Putting the menu
        // call in a while statement will allow the user to run
        // the program as many times as they wish.

        while (quit.equals("n")) {
            // call the menu
            menu();

            // prompt the user if they want to quit
            System.out.println("\nWould you like to quit - y or n");
            quit = myScan.nextLine().toLowerCase();
        }

        // Inform the user the program is done executing
        System.out.println("Good Bye");
    }
}
```

11.3.2 More Menu Examples

```
package switch_menuexamples;

import java.util.Scanner;


public class Switch_MenuExamples {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {

            //create variables
            int UserChoice1 = 0;
            char UserChoice2 = 'A';
            String UserChoice3 = "";

            //create a scanner for input
            Scanner myScanner = new Scanner(System.in);

            //Example One
            //Menu using integer choices
            System.out.println("Example 1 using integer choices");
            System.out.println("Choose one of the following:" +
                              "\n1. Move the character to the left" +
                               "\n2. Move the character to the right" +
                               "\n3. Move the character straight ahead" +
                               "\n4. Quit" +
                               "\nPlease choose 1, 2, 3 or 4:"
                                 );
            UserChoice1 = Integer.parseInt(myScanner.nextLine());
```

(continues on next page)

(continued from previous page)

```
        switch (UserChoice1)
        {
            case 1:
                System.out.println("Character moves to the left");
                break;
            case 2:
                System.out.println("Character moves to the right");
                break;
            case 3:
                System.out.println("Character moves straight");
                break;
            case 4:
                System.out.println("Quit Program");
                break;
            default:
                //if user chooses a value that is not 1,2,3 or 4
                System.out.println("That was not a choice");
                break;
        }

                //Example two
        //Menu using char choices
        System.out.println("\n\nExample 2 using char choices");
        System.out.println("Choose one of the following:" +
               "\nL  Move the character to the left" +
                "\nR  Move the character to the right" +
                "\nS  Move the character straight ahead" +
                "\nQ  Quit" +
                "\nPlease choose L, R, S or Q:"
                  );
        //NOTE the "toUpperCase()" This will convert anything typed
        //from the keyboard into uppercase.
        UserChoice2 = myScanner.nextLine().toUpperCase().charAt(0);

        switch (UserChoice2)
        {
            case 'L':
                System.out.println("Character moves to the left");
                break;
            case 'R':
                System.out.println("Character moves to the right");
                break;
            case 'S':
                System.out.println("Character moves straight");
                break;
            case 'Q':
                System.out.println("Quit Program");
                break;
            default:
                //if user chooses a value that is not L, R, S or Q
                System.out.println("That was not a choice");
                break;
        }

        //Example three
        //Menu using string choices
```

(continues on next page)

(continued from previous page)

```
            System.out.println("\n\nExample 3 using string choices");
            System.out.println("Choose one of the following:" +
                    "\nLeft      Move the character to the left" +
                    "\nRight     Move the character to the right" +
                    "\nStraight  Move the character straight ahead" +
                    "\nQuit      Quit" +
                    "\nPlease choose Left, Right, Straight or Quit:"
                      );
            //NOTE the "toUpperCase()" This will convert anything typed
            //from the keyboard into uppercase. Converting all the strings
            //to uppercase ensures we are comparing uppercase to uppercase
            UserChoice3 = myScanner.nextLine().toUpperCase();

            switch (UserChoice3)
            {
                case "LEFT":
                    System.out.println("Character moves to the left");
                    break;
                case "RIGHT":
                    System.out.println("Character moves to the right");
                    break;
                case "STRAIGHT":
                    System.out.println("Character moves straight");
                    break;
                case "QUIT":
                    System.out.println("Quit Program");
                    break;
                default:
                    //if user chooses a value that
                    //is not Left, Right, Straight or Quit
                    System.out.println("That was not a choice");
                    break;
            }


            //Inform the user the program is done executing
            System.out.println("\n\nGood Bye - Press enter to exit");
            myScanner.nextLine();
    }

}
```

11.3.3 New format of switch statement after version 14+

```
/*
 * Switch Statement
 * As of Java 14 went to the "->"
 */
package com.mycompany.switch_statement_revised;


import java.util.Scanner;

/**
```

(continues on next page)

(continued from previous page)

```
 *
 * @author jgoudy
 */
public class Switch_statement_revised {

    static int menu() {
        int choice = 0;

        try {
            Scanner myScan = new Scanner(System.in);

            System.out.print("Menu\n"
                    + "1. Pick Larry\n"
                    + "2. Pick Curly\n"
                    + "3. Pick Moe\n"
                    + "Please pick 1,2 or 3:  ");
            choice = Integer.parseInt(myScan.nextLine());

            return choice;
        } catch (Exception e) {
            return -1;
        }

    }

    static void Larry() {
        System.out.println("You chose Larry");
    }

    static void Curly() {
        System.out.println("You chose Curly");
    }

    static void Moe() {
        System.out.println("You chose Moe");
    }

    static void switch_example1() {
        // this format is the same in most languages
        // and in all version of java

        int choice = -1;

        System.out.println("Example 1");
        choice = menu();

        switch (choice) {
            case 1:
                Larry();
                break;
            case 2:
                Curly();
                break;
            case 3:
                Moe();
                break;
```

(continues on next page)

(continued from previous page)

```
            default:
                System.out.println("That wasn't a choice");

        }

        System.out.println("\n--------------------------\n");
    }

    static void switch_example2() {
        // This format of a switch effective version 14+

        int choice = -1;

        System.out.println("Example 1");
        choice = menu();

        switch (choice) {
            case 1 -> Larry();
            case 2 -> Curly();
            case 3 -> Moe();
            default ->
                System.out.println("That wasn't a choice");

        }

        System.out.println("\n--------------------------\n");
    }

    public static void main(String[] args) {

        switch_example1();

        switch_example2();

    }
}
```

End Of Topic

CHAPTER

TWELVE

STRING FUNCTIONS

12.1 Key Ideas

- Convert To Uppercase
- Covert To Lowercase
- Length of a string
- Substring - get parts of a string
- CharAt - retreive a character out of a string
- Replace a character

```
/*
String Functions
This program demonstrates some of the common
string functions

Jim Goudy
 */


public class String_Functions {

    public static void main(String[] args) {

        // Variables
        String xInputString;
        String xInputString2;
        String xTemp;
        int xLength = 0;
        char xChar;

        // Data for input strings
        xInputString = "Codingjava";
        xInputString2 = "Feed";

        // Original Word
        System.out.println("\nConvert To Uppercase");
        System.out.println(xInputString);


        // Get the length of a string
```

(continues on next page)

(continued from previous page)

```
        System.out.println("\nGet lenght of a string");
        xLength = xInputString.length();
        System.out.println(xInputString + " is " + xLength + " characters long.");

        // Convert To Uppercase
        System.out.println("\nConvert To Uppercase");
        xInputString = xInputString.toUpperCase();
        System.out.println(xInputString);

        // Convert To Lowercase
        System.out.println("\nConvert To Lowercase");
        xInputString = xInputString.toLowerCase();
        System.out.println(xInputString);

        // Replace a character
        System.out.println("\nReplace a character");
        xInputString = xInputString.replace('g', 'b');
        System.out.println(xInputString);

        // Replace a charater - notices how it
        // turns Feed into Food
        System.out.println("\nReplace a character");
        xInputString2 = xInputString2.replace('e', 'o');
        System.out.println(xInputString2);



        // Returns a string starting at the position
        // NOTE: positions for this command start at 0
        // Bumpoint --> this command will return point
        System.out.println("\nRetreive part of a string");
        xTemp = xInputString.substring(3);
        System.out.println(xTemp);

        // Returns a string starting at the position
        // NOTE: positions for this command start at 0
        // Orignial word is bumpoint
        System.out.println("\nRetrieve Partical Strings");
        xTemp = xInputString.substring(0, 3);   // pulls out bum
        System.out.println(xTemp);
        xTemp = xInputString.substring(3, 8); // points
        System.out.println(xTemp);
        // note how you can use length in the following expression
        xTemp = xInputString.substring(3, xInputString.length());
        System.out.println(xTemp);

        // Retrieve a char from a string
        // NOTE  scanners only retrive strings. There are no methods
        // for chars
        System.out.println("\nRetrieve a char");
        xChar = xInputString.charAt(0);   // retreives the B as a char
        System.out.println(xChar);
        xChar = xInputString.charAt(3); // retreive the P as a char
        System.out.println(xChar);

        // check the equalality of a string
```

(continues on next page)

(continued from previous page)

```
        // Note that String is an object and not a native datatype
        // Hence - we cannot use == for String
        // we must use the .equal()
        System.out.println("\nCheck For Equality");
        if (xInputString.equals(xInputString2)) {
            System.out.println("It matches");
        } else {
            System.out.println("NO match");
        }

        // Another example
        if (xInputString.equals("Montana")) {
            System.out.println("It matches");
        } else {
            System.out.println("NO match");
        }

        // Another example
        xTemp = "Montana";
        if (xTemp.equals("Montana")) {
            System.out.println("It matches");
        } else {
            System.out.println("NO match");
        }

    }


}
```

End Of Topic

CHAPTER

THIRTEEN

NUMBER FORMATS

13.1 Key Ideas

- Limiting decimal places
- Leading zeros
- Left padding
- Formatting numbers to have comas
- Formatting phone numbers
- Formatting dates and time

Example Code

```
/*
        Number Formats
        Programmer: James Goudy
*/


import java.math.BigInteger;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.util.Scanner;

public class NumberDateFormats
{

    //A function for left padding
    public static String LeftPadding(String input,
            int totalWidth, char paddingCharacter)
    {
            String output = null;       //Outpout String
            int PaddingLength = 0;

            //Set oupt to input
            output = input;

            //calculate the amount of padding
            PaddingLength = totalWidth - input.length();

            //Add padding in front of input
```

(continues on next page)

(continued from previous page)

```
        for(int cntr = 0; cntr <PaddingLength; cntr++)
        {
            output = paddingCharacter + output;
        }

        return output;
}


/**
 * @param args the command line arguments
 */
public static void main(String[] args)
{
    double answer = 0;
    double num1 = 1.0;
    double num2 = 3.0;
    String phoneNumber = "5554065555";
    double bigNumber = 123456789.1277;

    //DateTime(int year, int month, int day,
    //          int hours, int minutes, int seconds)
    LocalDateTime myDate = LocalDateTime.of(2030, 8, 5, 20, 7, 9);

    answer = num1 / num2;

    //
    System.out.println("Answer is " + answer);

    //output using place holders
    System.out.println("\nOutput using placeholders");
    System.out.println(num1 + " \\ " + num2 + " = " + answer);

    //The first line is using the "ToString"
    //The second line is using String.format()
    // Limit answer to two decimal places
    // # means if there is a number display it
    System.out.println("\nLimit Answer To Two Decimal Places");
    System.out.println("Answer is " + String.format("%.2f", answer));

    // Leading Zero - use 0 or 0's
    System.out.println("\nLimit answer to 3 decimal places"
            + "\nand include 2 leading zeros");
    System.out.println("Answer is " + String.format("%06.3f", answer));

    // Format a telephone number
    System.out.println("\nFormat a telephone number");
    System.out.println(String.format("(%s) %s-%s",
            phoneNumber.substring(0, 3),
            phoneNumber.substring(3, 6),
            phoneNumber.substring(6, 10)));

    // format number to include commas
    // 0's used after a decimal point
    // means always display that those decimal places
    System.out.println("\nFormat number to include commas");
```

(continues on next page)

(continued from previous page)

```
        System.out.println(String.format("%,f", bigNumber));
        //System.out.println(String.format("{0:0,0.00}", bigNumber));

        //Date Examples
        System.out.println("\n\nDate Examples");
        System.out.println(String.format("%1$tm/%1$td/%1$ty", myDate));
        System.out.println(String.format("%1$tM/%1$td/%1$tY", myDate));
        System.out.println(String.format("%1$tA %1$tB %1$td, %1$tY", myDate));
        System.out.println(String.format("%1$ta %1$tb %1$td, %1$tY", myDate));
        System.out.println(String.format("%1$tB %1$td, %1$tY "
                + "%1$tI:%1$tM:%1$tS %1$tp", myDate));
        System.out.println(String.format("%1$tB %1$td, %1$tY "
                + "%1$tH:%1$tM:%1$tS", myDate));


        //Left Padding Example
        System.out.println(LeftPadding(String.valueOf(122.33), 10, '-'));
        System.out.println(LeftPadding(String.valueOf( 2.33), 10, '-'));
        System.out.println(LeftPadding(String.valueOf( 88882.56), 10, '-'));
        System.out.println(LeftPadding(String.valueOf(
                String.format("%06.2f", answer)), 10, '-'));
        System.out.println(LeftPadding(String.valueOf(
                String.format("%03.2f", answer)), 10, '-'));
        System.out.println(LeftPadding(String.valueOf(
                String.format("%1.4f", answer)), 10, '-'));

        // Exit Program
        System.out.println("\n\nPress Enter To Quit");

    }

}

/*
Answer is 0.3333333333333333

Output using placeholders
1.0 \ 3.0 = 0.3333333333333333

Limit Answer To Two Decimal Places
Answer is 0.33

Limit answer to 3 decimal places
and include 2 leading zeros
Answer is 00.333

Format a telephone number
(555) 406-5555

Format number to include commas
123,456,789.127700


Date Examples
08/05/30
07/05/2030
```

(continues on next page)

(continued from previous page)

```
Monday August 05, 2030
Mon Aug 05, 2030
August 05, 2030 08:07:09 pm
August 05, 2030 20:07:09
----122.33
------2.33
--88882.56
----000.33
------0.33
----0.3333


Press Enter To Quit


*/
```

End Of Topic

CHAPTER FOURTEEN

ITERATION / LOOPS

14.1 Key Ideas

- Do Loop
- While Loop
- For Loop
- For Each Loop
- Loop Pattern for Menus / Programming Structure
- Loops for grids

14.2 Readings

https://books.trinket.io/thinkjava2/chapter6.html

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/while.html

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/while.html

14.3 Lecture

Definition

Iteration / Loops: the repetition of a sequence of computer instructions a specified number of times or until a condition is met. Merriam Webster

14.3.1 Java Loops

Iteration has several purposes:

- **Reusability** - loops allow us to repeat code as often as we wish in the form that we can write it once and run it many times.
- Loops allow us to traverse the items in a list, array, or data structure.

14.4 Types Of Loops

Java has four main types of loops

- **Do Loop** - This loop will *always* run once
- **While Loop** - This loop runs while a condition is true
- **For Loop** - This loop runs a specific amount of times
- **For Each Loop** - This loop is a form of the **for-loop**. It is set up to easily iteration of each item in an array, list, or data structure.

14.5 Example Basic Loops

```
/*
Loop Lecture Code
Programmer: James Goudy
 */



public class Loops {

    public static void main(String[] args) {

        int cntr = 0;

        //Do Loop This loop will always run once
        do
        {
            System.out.println(cntr + " do loop");

            //We have to increment a counter
            //otherwise our while statement will never
            //go to false and we will be in a endless loop.
            cntr++;
        }while(cntr < 2000);


        //while loop - This loop will always check for true before running
        cntr = 0;

        while (cntr < 2000)
        {
            System.out.println(cntr + " while loop");
```

(continues on next page)

(continued from previous page)

```
            cntr++;
            //We have to increment a counter
            //otherwise our while statement will never
            //go to false and we will be in a endless loop.
        }

        //for statments are excellent if we have
        //a known number of cycles to loop.
        //Also we create a cntr, check for true and increment
        //the counter all on one line.
        for(int cntr2 = 0; cntr2 < 2000; cntr2++)
        {
            System.out.println(cntr2 + " for loop");
        }

    }

}
```

14.6 Loop To Control Menu

- Note this is using a **while-loop** and a **switch statement**.
- This is a programming style that fits 90% of your programs. Note that the functions are *not* calling the *menu() function*.

```
/* Switch /
Program: Menu Lecture
Programmer: Jim Goudy
This is a demo program to demonstrate a design pattern for creating a menu system in␣
↪the console environment.
:)----  Bob Ross Coding ---- :)

 */
package menu_lecture;

import java.util.Scanner;

public class Menu_Lecture {

    //Happy Function 1
    static void HappyFunction1() {
        System.out.println("\n\nThis is happy function 1");
    }

    //Happy Function 2
    static void HappyFunction2() {
        System.out.println("\n\nThis is happy function 2");
    }

    //Happy Function 3
    static void HappyFunction3() {
        System.out.println("\n\nThis is happy function 3");
```

(continues on next page)

(continued from previous page)

```
    }

    //This function creates a menu system.
    static void menu() {
        //variables
        int choice = 0;     //hold user choice

        //Scanner for user input
        Scanner myScan = new Scanner(System.in);

        //display the user choices
        System.out.println("\nMENU\n"
                + "\n1. Happy Function 1"
                + "\n2. Happy Function 2"
                + "\n3. Happy Function 3"
                + "\nPlease choose a function 1,2 or 3");
        choice = Integer.parseInt(myScan.nextLine());

        //take the choice stored in choice and use it in the switch
        switch (choice) {
            case 1:
                HappyFunction1();
                break;
            case 2:
                HappyFunction2();
                break;
            case 3:
                HappyFunction3();
                break;
            default:
                //default is used if the user entered any
                //number that was note 1,2 or 3
                System.out.println("\nThat wasn't a choice");
        }
    }

    public static void main(String[] args) {

        //variables
        String quit = "n";

        //Scanner for user input
        Scanner myScan = new Scanner(System.in);

        //quit has to be initialized with the value of n
        //so the loop will run at least once. Putting the menu
        //call in a while statement will allow the user to run
        //the program as many times as they wish.
        while (quit.equals("n")) {

            //call the menu
            menu();
            //prompt the user if they want to quit
            System.out.println("\nWould you like to quit -  y or n");
            quit = myScan.nextLine().toLowerCase();
        }
```

(continues on next page)

(continued from previous page)

```
        //Inform the user the program is done executing
        System.out.println("Good Bye");
    }
}
```

14.7 Loops For Printing A Grid

```
/*
Program: Loop Lecture Grid
Programmer: James Goudy
This program demonstrates how to write a grid
using loops.  The important note is that the
counters in the loops act as coordinate to the row and columns
of the x's that are being printed out.

 */
package loops_lecture_grid;

public class Loops_Lecture_Grid {

    public static void main(String[] args) {

        //Variables
        String xx = "x";

        int Colcntr = 0;     //This is our column counter
        int ColStop = 5;     //This is the actual number of
        //columns for the grid

        int Rowcntr = 0;     //This is our row counter
        int RowStop = 5;     //This is the actual number of
        //rows for the grid

        //loop to control the number of rows
        while (Rowcntr < RowStop) {
            //loopt to control the number of columns
            while (Colcntr < ColStop) {
                //Note the print. If a println is used
                //all of the starts will print down / "vertical"
                System.out.print("x");

                //Increment the column counter to the next column
                Colcntr++;
            }

            //this represents the end of the row
            //the cursor needs to be put on the next row
            //so a println will be used.
            System.out.println("");

            //Next the Colcntr needs to be reset to zero
            Colcntr = 0;
```

(continues on next page)

(continued from previous page)

```
            //Advance the rowcnter to the next row
            Rowcntr++;
        }

    }

}
```

End Of Topic

CHAPTER

FIFTEEN

EXCEPTION HANDLING

15.1 Key Ideas

- Try / Catch

15.2 Readings

https://books.trinket.io/thinkjava2/chapter15.html#sec187

Definition

Try and Catch - This statement allows to so encapsultate a part of code that has the possibility to error or crash. The try / catch will prevent the program from crashing.

```
/*
Try and Catch Demo
by James Goudy
 */


import java.util.Scanner;

public class J1_Try_Catch {

    public static void main(String[] args) {

        //Variables
        char xquit = 'n';
        String zz;
        String xinput;
        char ww;
        double xx;


        //create a scanner
        Scanner scan = new Scanner(System.in);

        zz = "Top";

```

(continues on next page)

(continued from previous page)

```
        // try and catch statments allow us to try code that could
        // possibly fail.  We put the code that could fail in the "try" part.
        // If it fails, we then catch the error in the "catch"
        // Exception e is the actual error message.
        try {
            //the word only has 3 letters and we are looking for the 10 letter
            // which it doesn not have.
            ww = zz.charAt(9);
            System.out.println("The char is " + ww);

        } catch (Exception e) {
            //if there is an error, then this code will rund
            System.out.println(e);
            System.out.println("You had an error - cant you count - try again");
        } finally {
            //finally is optional and it will always run
            System.out.println(" any code in the finally section  "
                    + " will always run.");
        }

        //Example 2
        //This also demonstrates how we can use a char
        //and use it in a while statement.
        while (xquit != 'y') {
            //this try and catch will catch any errors if anything
            //but a number is entered.
            try {
                System.out.println("Enter a number");
                xx = Double.parseDouble(scan.nextLine());
                System.out.println("You entered " + xx);

            } catch (Exception e) {
                System.out.println("You  did not enter a number");
            }

            System.out.println("\nWould you like to quit y / n ");
            xinput = scan.nextLine().toLowerCase();
            //notice how we are taking one letter out and making it a char
            //remember that positions that at the number 0
            xquit = xinput.charAt(0);
        }

       System.out.println("\n\nGoodbye");
    }

}
```

End Of Topic

CHAPTER SIXTEEN

FILE, FOLDER - CREATION AND DELETION

16.1 Key Ideas

- Objectives
- Global Variables
- Creating directories
- Deleting Directories
- Creating files
- Reading Files
- Deleting Files

16.2 Lecture Code

```
/*
 * Programmer: James Goudy
 * Project: File, folder creation and deletion
 */
package io_demo;

/**
 *
 * @author jgoudy
 */
/*
 * Objectives
 * Global Variables
 * Creating directories
 * Deleting Directories
 * Creating files
 * Reading Files
 * Deleting Files
 */
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;
```

(continues on next page)

(continued from previous page)

```
import java.nio.file.Paths;
import java.nio.file.StandardOpenOption;

public class IO_Demo {

    //Gloabal variable - used by all functions
    static String xStrPath;
    //static double[][] myArray;

    //Create a directory
    static void ioCreateDirectory() {

        boolean checkDir;

        //import java.nio.file Path
        //import java.nio.file Paths - helper class
        //Path sets a path for a file or directory
        Path xPath = Paths.get(xStrPath);

        //true if the file does exist;
        //false if the file does not exists or its existence cannot be determined
        checkDir = Files.exists(xPath);

        if (checkDir) {
            System.out.println("Directory Already Exists");
            return;
        }

        try {
            //Creates a directory - note that
            //it uses the path that was just created
            //Note that Files requires it to be in
            //try - catch statement
            Files.createDirectory(xPath);
        } catch (Exception e) {
            System.out.println("Could not create directory");
        }
    }

    //Delete a directory
    //Note: in order to delete a directory it must be empty first
    static void ioDeleteDirectory() {
        Path xDeletePath = Paths.get(xStrPath);

        try {
            Files.delete(xDeletePath);
        } catch (Exception e) {
            System.out.println("Could not delete the direcotry");
        }
    }

    //Write a text file
    static void ioCreateTextFile() {
        xStrPath = "c:\\zJavaTemp\\myFile1.txt";
        String text1;
        String text2;
```

(continues on next page)

(continued from previous page)

```
        boolean fileCheck;

        Path pathNewFile = Paths.get(xStrPath);

        fileCheck = Files.isRegularFile(pathNewFile)
                & Files.isReadable(pathNewFile)
                & Files.isExecutable(pathNewFile);

        try {

            BufferedWriter fw = null;

            if (fileCheck) {
                fw
                        = Files.newBufferedWriter(pathNewFile,
                                StandardCharsets.UTF_8,
                                StandardOpenOption.APPEND);
            } else {
                fw
                        = Files.newBufferedWriter(pathNewFile,
                                StandardCharsets.UTF_16,
                                StandardOpenOption.CREATE);
            }

            /*
             * Charatersets
             * StandardCharsets.UTF_8;
             * StandardCharsets.US_ASCII;
             * StandardCharsets.UTF_16;
             *
             * Opening and deleting files
             * StandardOpenOption.CREATE_NEW;
             * StandardOpenOption.APPEND;
             * StandardOpenOption.DELETE_ON_CLOSE;
             * StandardOpenOption.TRUNCATE_EXISTING;
             * StandardOpenOption.DELETE_ON_CLOSE;
             */

            //Note for windows \r\n has to be used in combination
            //For universal newlines, use the newline method
            text1 = "xx This is how you write a file. ";
            text2 = "(You store user input in a variable then write it. \r\n";

            for (int cntr = 0; cntr < 10; cntr++) {
                fw.write(text1);
                fw.newLine();
                fw.write(text2);

                fw.flush();
            }

            fw.close();
        } catch (Exception e) {
            System.out.println("Could not write the file");
        }
```

(continues on next page)

(continued from previous page)

```
    }

    //Read a text file
    static void ioReadFile() {
        xStrPath = "c:\\zJavaTemp\\myFile1.txt";
        String intext1;
        String text2;
        boolean fileCheck;

        Path pathOpenFile = Paths.get(xStrPath);

        //check if the files is real,
        //the file is readable, the file is not locked
        fileCheck = Files.isRegularFile(pathOpenFile)
                & Files.isReadable(pathOpenFile)
                & Files.isExecutable(pathOpenFile);

        if (!fileCheck) {
            System.out.println("File could not be opened");
            return;
        }

        try {

            //We will use the newBufferedReader
            //to read in our text file
            BufferedReader fr = Files.newBufferedReader(
                    pathOpenFile,
                    StandardCharsets.UTF_8);

            while ((intext1 = fr.readLine()) != null) {
                System.out.println(intext1);
            }

            fr.close();
        } catch (Exception e) {
            System.out.println("Could not read the file");
        }

    }

    static void ioDeleteFile() {

        xStrPath = "c:\\zJavaTemp\\myFile1.txt";

        //Get the path of the file to delet
        Path xPath = Paths.get(xStrPath);

        try {
            Files.delete(xPath);
            System.out.println("File successfully deleted");
        } catch (Exception e) {
            System.out.println("File not deleted");
        }

    }
```

(continues on next page)

(continued from previous page)

```
    public static void main(String[] args) {

        char xdeleteDir = 'n';

        //Note that this is a global variable
        xStrPath = "c:\\zJavaTemp";

        ioCreateDirectory();

        xdeleteDir = 'n';
        if (xdeleteDir == 'y') {
            ioDeleteDirectory();
        }

        //Creating and and Reading a Textfile
        ioCreateTextFile();
        ioReadFile();

        // Toggle the comment to delete the file
        // ioDeleteFile();
    }
}
```

End of Topic

CHAPTER

SEVENTEEN

IO READ CSV FILE

17.1 Key Ideas

- Read a comma-separated file

17.2 Lecture Code

```java
/*
 * Programmer: James Goudy
 * Project: IO_CSV_Demo Rev 2 20220630
 */
package com.company.my.j1_io_csv_demo_rev2;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;


/**
 *
 * @author jgoudy
 */

class readCSV_BufferedReader {

    BufferedReader br;
    FileReader fr;

    int numCols;
    int numRows;


    String filePath;

    public readCSV_BufferedReader(String filePath) {

        this.filePath = filePath;

        // creating a buffered reader in its own function
        // this will allow the resetting of the buffered reader
        createBufferedReader(filePath);
```

(continues on next page)

(continued from previous page)

```
    }

    private void createBufferedReader(String thefilePath) {

        // BufferedReader requires a try and catch
        try {
            br = new BufferedReader(new FileReader(thefilePath));
        } catch (IOException ex) {
            System.out.println(ex.getMessage());
        }

    }




    private void getArrayDimensions() {

        // this function is used only for calculating the
        // dimension of an array

        String inputLine = "";

        try {

            // skip the header row
            br.readLine();

            // count rows
            while ((inputLine = br.readLine()) != null) {
                numRows++;
            }

            //reset buffered reader
            createBufferedReader(this.filePath);

            // calculate columns
            inputLine = br.readLine();
            String[] inputArrary = inputLine.split(",");
            numCols = inputArrary.length;

            // optional prints to verify Rows and cols
            // System.out.println("Rows = " + numRows);
            // System.out.println("Cols = " + numCols);

        } catch (IOException ex) {
            System.out.println(ex.getMessage());
        }

    }

    public String[][] readDataIntoArray() {

        // this function will return a 2 dimensional array
        // not this can be modified to read in data and save
```

(continues on next page)

(continued from previous page)

```
        // it to a linked list.  In that case, all the dataArray
        // code can be eliminated. One we still need to do the
        // input array code.


        getArrayDimensions();

        // instantiate the new array
        String[][] dataArray = new String[numRows][numCols];
        String inputLine;

        int rowCount = 0;

        // reset buffered reader
        createBufferedReader(this.filePath);

        try {

            // skip the header row
            br.readLine();

            // br.readline reads the input file line
            // and stores it in inputline within the while statement

            while ((inputLine = br.readLine()) != null) {

                // split the inputline into a string array
                String[] inputArrary = inputLine.split(",");

                // add the data from the inputArray to the dataArray
                for(int i = 0; i < inputArrary.length; i++)
                {
                    dataArray[rowCount][i] = inputArrary[i];
                }
                System.out.println("");

                rowCount++;
            }

        } catch (IOException ex) {
            System.out.println(ex.getMessage());
        }

        return dataArray;
    }

    public int getNumCols() {
        return numCols;
    }

    public int getNumRows() {
        return numRows;
    }


}  // end of class
```

(continues on next page)

(continued from previous page)

```
public class J1_IO_CSV_Demo_Rev2 {

    static String[][] theArray;

    public void printArray(int rows, int cols)
    {
       for (int r = 0; r < cols; r++) {
            for(int c = 0; c < cols; c++)
            {
                System.out.print(theArray[r][c] + " ");
            }
            System.out.println("");
        }

    }

    public static void main(String[] args) {

        // file location
        String dataFile = "c:\\z\\citydata.csv";

        // instantiate a new object
        readCSV_BufferedReader br = new readCSV_BufferedReader(dataFile);


        theArray = new String[br.getNumRows()][br.getNumCols()];

        try {
           // This will over write the current array
           // with the returned array with the new dimensions
           theArray = br.readDataIntoArray();

        } catch (Exception e) {

            System.out.println(e.getMessage());
        }

        // print the array
        printArray(br.getNumRows(),br.br.getNumCols());

        // exit
        System.out.println("\nbye");
    }
}


/*
 citydata.csv
 Key,Town,State,Population
 1,New York,New York,18713220
 2,Los Angeles,California,12750807
 3,Chicago,Illinois,8604203
 4,Miami,Florida,6445545
 5,Dallas,Texas,5743938
 6,Philadelphia,Pennsylvania,5649300
```

(continues on next page)

(continued from previous page)

```
7,Houston,Texas,5464251
8,Atlanta,Georgia,5449398
9,Washington,District of Columbia,5379184
10,Boston,Massachusetts,4688346
11,Phoenix,Arizona,4219697
12,Seattle,Washington,3789215
13,San Francisco,California,3592294
14,Detroit,Michigan,3506126
15,San Diego,California,3220118
16,Minneapolis,Minnesota,2977172
17,Tampa,Florida,2908063
18,Denver,Colorado,2876625
19,Brooklyn,New York,2559903
20,Queens,New York,2230722
21,Riverside,California,2107852
22,Baltimore,Maryland,2106068
23,Las Vegas,Nevada,2104198
24,Portland,Oregon,2074775
25,San Antonio,Texas,2049293
26,St. Louis,Missouri,2024074
27,Sacramento,California,1898019
28,Orlando,Florida,1822394
29,San Jose,California,1798103
30,Cleveland,Ohio,1710093
31,Pittsburgh,Pennsylvania,1703266
32,Austin,Texas,1687311
33,Cincinnati,Ohio,1662691
34,Kansas City,Missouri,1636715
35,Manhattan,New York,1628706
36,Indianapolis,Indiana,1588961
37,Columbus,Ohio,1562009
38,Charlotte,North Carolina,1512923
39,Virginia Beach,Virginia,1478868
40,Bronx,New York,1418207
41,Milwaukee,Wisconsin,1365787
42,Providence,Rhode Island,1203230
43,Jacksonville,Florida,1181496
44,Salt Lake City,Utah,1098526
45,Nashville,Tennessee,1081903
46,Richmond,Virginia,1075798
47,Memphis,Tennessee,1066967
48,Raleigh,North Carolina,1038738
49,New Orleans,Louisiana,1020886
50,Louisville,Kentucky,1005654
*
*/
```

End of Topic

CHAPTER

EIGHTEEN

ARRAYS

18.1 Key Ideas

- Storing sets of data
- Arrays

18.2 Readings

https://books.trinket.io/thinkjava2/chapter7.html

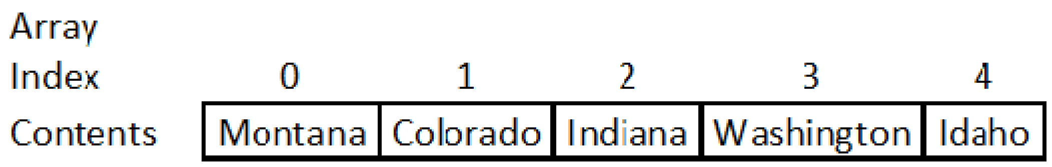
https://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html

Definition

Array - An array is a data structure that contains a group of elements of a fixed size. They are used to store sets of data like golf scores, budgets, etc. where data is stored like a “grid”.

18.3 Concepts

18.3.1 Visualization



Index - the index identifies the location of the data.

Element - the box itself is referred to as an element

```
// This is a one dimensional array
String[] states = new String[5];

// set values - include the index and value
```

(continues on next page)

(continued from previous page)

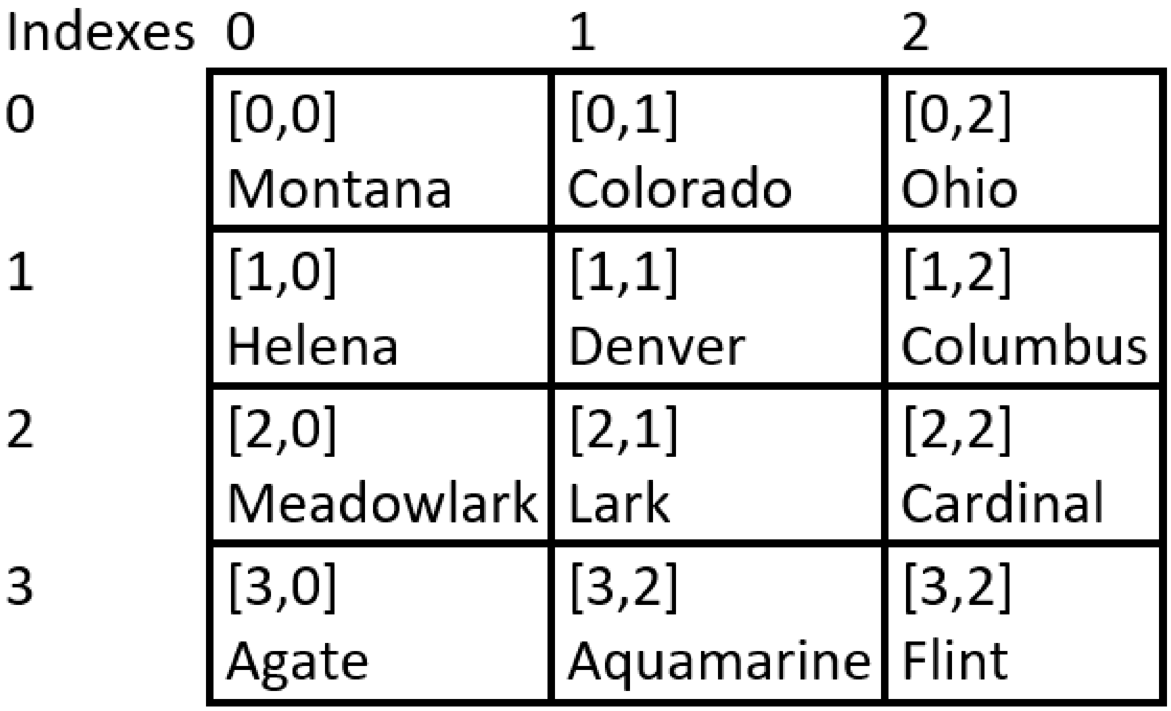
```
states[0] = "Montana";
states[1] = "Colorado";
states[2] = "Indiana";
states[3] = "Washington";
states[4] = "Idaho";

//alternative method
String[] states2 = {"Montana","Colorado","Indiana","Washington","Idaho"};

// retrieve value
String astate;
astate = states[2];
System.out.print(astate);
// output is Indiana
```

Tip: In the majority of programming languages, the first index starts at 0.

18.3.2 2 Dimensional Array

Assume the following array. It is tracking states, their capitals, state bird, and state rock. It is named stateData.

The "[ ]" are there for reference only. The squares "[ ]" indicate the coordinate system. They are not part of the actual data - they are there for reference only. Note that the coordinates are [row, column].



Indexes	0	1	2
0	[0,0] Montana	[0,1] Colorado	[0,2] Ohio
1	[1,0] Helena	[1,1] Denver	[1,2] Columbus
2	[2,0] Meadowlark	[2,1] Lark	[2,2] Cardinal
3	[3,0] Agate	[3,2] Aquamarine	[3,2] Flint

```
// create the 2 dimensional array
String[][] stateData =
```

(continues on next page)

(continued from previous page)

```
{
    {"Montana", "Colorado", "Ohio"},
    {"Helena", "Denver", "Columbus"},
    {"Meadolark", "Lark", "Cardinal"},
    {"Agate", "Aquamarine", "Flint"}
};

// output some data
String myData = "";
myData = stateData[1][1];
System.out.println("Capital of Colorado is " + myData);

myData = stateData[2][0];
System.out.println("The state bird of Montana is " + myData);

myData = stateData[0][2];
System.out.println("The state is " + myData);

// change Montana's state bird to hawk
myData = "Hawk";
stateData[2][0] = myData;
myData = stateData[2][0];
System.out.println("The state bird of Montana is " + myData);

// to iterate through the 2 dimensional array
int Rows = 4;
int Cols = 3;

        // this for statement controls the rows
        // note that r will keep track of the index of the rows
        for(int r = 0; r < Rows; r++)
    {
               // this for statement controls the columns
                // note that c will keep track of the index of the columns
        for(int c = 0; c < Cols; c++)
        {
            System.out.print(stateData[r][c] + " ");
        }
        // code is now at the end of a "row"
        // move the cursor down
        System.out.println();
    }


/*
Output
Capital of Colorado is Denver
The state bird of Montana is Meadolark
The state is Ohio
The state bird of Montana is Hawk
*/
```

18.4 Lecture Code

```
/*
Programmer: James Goudy
Project:  Array Lecture Code
ANT project
 */


import java.util.Arrays;


public class Array_Lecture {

    public static void main(String[] args) {

        //Variables

        //declare an arrays
        double[] Array1;
        String[] DogName;
        int[] Scores;
        double[][] Grid3by2;

        //Variables
        String ADogName;
        String stryy;


        //Variables for printing grid
        int Colcntr = 0;
        int ColStop = 2;

        int Rowcntr = 0;
        int RowStop = 3;

        //Initalize the array
        //This is where we set how many elements we need.
        //Consider an element as a box to store the data
        Array1 = new double[5];  //We can store 5 doubles
        DogName = new String[3];   //Here we can store 3 names
        Scores = new int[4];  //Here we can store  4 scores
        Grid3by2 = new double[3][2];


        //Here is an example of how we can
        //create and initialize an array in one step
        String[] colors = {"Red", "Green", "Blue"};
        String[][] names = {{"Mr. ", "Mrs. ", "Ms. "}, {"Smith", "Jones", "Rodes"}};

        //How we enter values into an array
        DogName[0] = "Fido";
        DogName[1] = "Spot";
        DogName[2] = "Barky";

        //Assign an element to a variable:
```

(continues on next page)

(continued from previous page)

```
        ADogName = DogName[1];  // This assigns the value of Spot to ADogName

        System.out.println("The second dog is " + ADogName);
        //Print the name out by referencing the array element
        System.out.println("The second dog is " + DogName[1]);


        //Here is how we enter data into Grid3by2
        //Here is the data in the grid formataion
        //   10 | 15
        //   -- | --
        //   20 | 25
        //   -- | --
        //   30 | 35

        Grid3by2[0][0] = 10;
        Grid3by2[0][1] = 15;
        Grid3by2[1][0] = 20;
        Grid3by2[1][1] = 25;
        Grid3by2[2][0] = 30;
        Grid3by2[2][1] = 35;


        System.out.println("\n----------------\n");

        //Loop to control rows
        while (Rowcntr < RowStop) {

            //Loop to control columns
            while (Colcntr < ColStop) {
                System.out.print(Grid3by2[Rowcntr][Colcntr] + "   ");
                Colcntr++;
            }
            Colcntr = 0;
            System.out.println();
            Rowcntr++;
        }

        System.out.println("\n----------------\n");

        //using a for  to loop through an array
        for (String d : DogName) {
            System.out.println(d);
        }

        //An example of using a for statement to print
        // an array - note that in this example we are storing
        //the array value in myValue and then printing it.
        //

        System.out.println("\n----------------\n");

        double myValue;

        // Note how the counters row and col will always increment
        // to the coordinates of current element
```

(continues on next page)

(continued from previous page)

```
        for (int row = 0; row < 3; row++) {
            for (int col = 0; col < 2; col++) {
                myValue = Grid3by2[row][col];
                System.out.print(myValue + " ");
            }
            System.out.println();
        }

        System.out.println("\n------------------\n");

        //Short hand version of using a for statement
        for (double[] dx : Grid3by2) {
            for (double dy : dx) {
                System.out.print(dy);
            }
            System.out.println();
        }


        ////////////////////////////////////////
        //Sort an Array
        ////////////////////////////////////////
        System.out.println("\n------------------\n");

        int[] Numbers1 = {5, 2, 4, 3, 0, 7, 1, 6};

        for (int NumCntr = 0; NumCntr < Numbers1.length; NumCntr++) {
            System.out.print(Numbers1[NumCntr] + " ");
        }

        Arrays.sort(Numbers1);

        System.out.print("\n\n");
        for (int NumCntr = 0; NumCntr < Numbers1.length; NumCntr++) {
            System.out.print(Numbers1[NumCntr] + " ");
        }

        ////////////////////////////////////////
        //Search an Array
        ////////////////////////////////////////
        System.out.println("\n------------------\n");
        String SearchItem;
        int FoundItem;

        String[] vehicles = {"bicycle", "truck", "car", "atv",
            "segway", "motor home", "skate board",
            "airplane", "boat", "canoe"};
        SearchItem = "segway";

        Arrays.sort(vehicles);

        //Binary Search searches for the item
        //If found, it will return the index
        //If not found, it will return a -1
        //Note Arrays must be sorted first in order to
        //return accurate results
```

(continues on next page)

(continued from previous page)

```
        FoundItem = Arrays.binarySearch(vehicles, SearchItem);

        if (FoundItem > -1) {
            System.out.println("Founditem is " + vehicles[FoundItem]);
        } else {
            System.out.println("Vehicle not found");
        }


        ////////////////////////////////////////
        //Fill an Array
        ////////////////////////////////////////
        System.out.println("\n----------------\n");

        //Replaces all the values with the value 99
        Arrays.fill(Numbers1, 99);

        for (int NumCntr = 0; NumCntr < Numbers1.length; NumCntr++) {
            System.out.print(Numbers1[NumCntr] + " ");
        }

        System.out.println("\n\n");
        //Fill the first three spots withthe number 42
        Arrays.fill(Numbers1, 0, 3, 42);

        for (int NumCntr = 0; NumCntr < Numbers1.length; NumCntr++) {
            System.out.print(Numbers1[NumCntr] + " ");
        }

        System.out.println("\n\n");

        Arrays.fill(vehicles, "");

        for (int NumCntr = 0; NumCntr < vehicles.length; NumCntr++) {
            System.out.print(vehicles[NumCntr] + " ");
        }
        System.out.println("\n----------------\n");

    }
}
```

End Of Topic

CHAPTER

NINETEEN

ARRAYLISTS

19.1 Key Ideas

- ArrayLists

19.2 Readings

https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html

https://www.geeksforgeeks.org/arraylist-in-java/

19.3 Concepts

Definition

ArrayList is a resizable one-dimensional array that can store any data type in any element.

Tip

The best practice is to make all the data the same type. This is done using the classifier.

```
    // only accept data that is a double
    ArrayList<Double> myArrayList = new ArrayList();

    // only accept data that is an int
    ArrayList<Integer> myArrayList = new ArrayList();

    // only accept data that is a String
    ArrayList<String> myArrayList = new ArrayList();

    // accepts any type of data including mixed
    ArrayList myArrayList = new ArrayList();
```

19.4 Lecture Code

19.4.1 Observations:

- We create an ArrayList with the following syntax:

```
ArrayList<datatype> arraylistname = new ArrayList();
```

- *.add(value)* method is used to add a data to the ArrayList
- *.remove(index)* method will remove data by position
- *.remove(value)* method will remove data by value
- *.clear()* method will clear the ArrayList of all elements - "emptying it"
- We iterate/loop through the array using a for statement:

```
/*
Project: ArrayLists
Programmer: James Goudy
 */

import java.util.ArrayList;
import java.util.Collections;


public class Arraylist_Lecture {

    public static void main(String[] args) {
        // Variables
        String stryy;


        ////////////////////////////////////////////////
        //
        // Array List
        //
        ////////////////////////////////////////////////

        // create the Array List
        // Note: <String> is telling the arraylist to only
        // accept data that is a String
        ArrayList<String> cars = new ArrayList();

        // use the add method to add data
        cars.add("Ford");
        cars.add("Chevy");
        cars.add("Scion");
        cars.add("Honda");

        //Retrieve one item - use the .get(x) where x is an int index location
        System.out.println(cars.get(3));

        //print out all items
        for (int CarCntr = 0; CarCntr < cars.size(); CarCntr++) {
            System.out.println("This is " + cars.get(CarCntr));
```

(continues on next page)

(continued from previous page)

```
        }

        System.out.println("\n-------Print With ForEach -----------");

        // print items using forEach
        cars.forEach((n)->{System.out.println(n);});

        System.out.println("\n------------------");


        ////////////////////////////////////////
        // Sort an Arraylist
        ////////////////////////////////////////

        // To sort an Arraylist, you must do it through the collections
        // library.  Add imports java.util.collections

        Collections.sort(cars);
        // print out all items
        System.out.println("\n\nCars sorted list ");
        for (int CarCntr = 0; CarCntr < cars.size(); CarCntr++) {
            System.out.println("This is " + cars.get(CarCntr));
        }

        System.out.println("\n----------------\n");
        // Remove Scion by referencing the index
        cars.remove(3);

        for (int CarCntr = 0; CarCntr < cars.size(); CarCntr++) {
            System.out.println("This is " + cars.get(CarCntr));
        }

        System.out.println("\n----------------\n");

        // Remove Chevy by using the actual data
        stryy = "Chevy";
        cars.remove(stryy);

        for (int CarCntr = 0; CarCntr < cars.size(); CarCntr++) {
            System.out.println("This is " + cars.get(CarCntr));
        }

        System.out.println("\n----------------\n");

        // insert an item - here we are inserting Dodge at the beginning of the list
        cars.add(0, "Dodge");

        for (int CarCntr = 0; CarCntr < cars.size(); CarCntr++) {
            System.out.println("This is " + cars.get(CarCntr));
        }

        // To empty or clear the arraylist
        cars.clear();
        System.out.println("The number of elements is cars is " +
        cars.size());
```

(continues on next page)

(continued from previous page)

```
    }
}
```

End Of Topic

CHAPTER

TWENTY

OVERLOADING

Definition

Overloading Overloading a function is to use the same function name, but give it different parameters.

20.1 Lecture Code

Note in the lecture code below how *funct1()* has different parameters passed to it. It first has *String* data passed to it, then *double* data, followed by *int* data, and then both *String* and *int* data. Same function name with different data types.

Overloading a function is used mostly in creating JAVA constructors when building *classes*.

```java
public class Overload_demo {


    //func1 with no parameters
    static void func1() {
        System.out.println("This is function 1 with no arguments");
        return;
    }
    //func1 with a string parameter
    static void func1(String mydata) {
        System.out.println("This is function 1 with a string");
        return;
    }
    //func1 with a double parameter
    static void func1(double mydata) {
        System.out.println("This is function 1 with a double");
        return;
    }
    //func1 with an int parameter
    static void func1(int mydata) {
        System.out.println("This is function 1 with a double");
        return;
    }
    //func1 with a string and int parameter
    //note that this one also returns a string
    //where the other one did not.
    static String func1(String mystring, int mydata) {
        String Ans = "";
        Ans = ("This is function 1 with a string and an int");
```

(continues on next page)

(continued from previous page)

```
        return Ans;
    }
    //add2 - add two doubles
    static double add2(double num1, double num2)
    {
        return (num1 + num2);
    }
    //add2 - concatenate two strings
    static String add2(String text1,String text2)
    {
        return (text1 + " " + text2 + " - is great!");
    }

    public static void main(String[] args) {
        //func1 examples
        func1();
        func1("Bubba");
        func1(22.5);
        func1(999);
        System.out.println(func1("Bob", 59));

        //add2 examples
        System.out.println(add2(100.00,6000.00));
        System.out.println(add2("hot","dog"));

    }
}
```

End Of Topic

CHAPTER TWENTYONE

CLASSES AND OBJECTS

21.1 Key Ideas

- Classes
- Objects
- Constructors
- Methods

21.2 Readings

https://books.trinket.io/thinkjava2/chapter11.html

https://www.geeksforgeeks.org/classes-objects-java/

21.3 Concepts

21.3.1 Class Definition

Definition

Class is a blueprint or template that is used to create objects. That template is considered to be a "data type" and can include functions/methods.

21.3.2 Attributes

Definition

Attributes This is the data that describes our class.

For example, if we wanted to create a class to make dogs, we might describe a dog by having a breed, a name, and a color. The attributes would be breed, name, and color.

21.3.3 Constructors

Definition

Constructor A constructor is a function(s) that allows us to set initial values to our attributes. It can be overloaded pending the values we need to set at the time of creating the object. Note that the constructor matches the name of the class and has no return value.

```java
class Dog{

    // Define the attributes of a dog

    String breed;
    String name;
    String Color;

    // NOTE: we can have as many constructors as we need.
    // If one is not defined,
    // the a "default" constructor is generated behind the scenes.
    // It is always good form to include constructors

    // if no constructor is defined, then this one is the default
    public Dog()
    {
    }

    // constructor if we know the breed and name at the time of creation
    // optional
    public Dog(String breed, String name)
    {
        this.breed = breed;
        this.name = name;
    }

    // constructor if we know the breed, name, color at the time of creation
    // optional
    public Dog(String breed, String name, String Color)
    {
        this.breed = breed;
        this.name = name;
        this.Color = Color;
    }

}
```

21.4 Setters and Getters

Definition

Setters and Getters - *Setters* are methods that allow a programmer to set the value of an attribute of an object. Setters allow the programmer to check a value before changing its value. *Getters* are methods that allow a programmer to retrieve the value of an attribute of an object.

```
class Dog{

    // Define the attributes of a dog

    // --------- ******** ----------
    // Values can initially be set by the constructor
    // and changed via the Set function

    String breed;
    String name;
    String color;

    // NOTE: we can have as many constructors as we need.
    // If one is not defined,
    // the a "default" constructor is generated behind the scenes.
    // It is always good form to include constructors

    // if no constructor is defined, then this one is the default
    public Dog()
    {
    }

    // constructor if we know the breed and name at the time of creation
    // optional
    public Dog(String breed, String name)
    {
        this.breed = breed;
        this.name = name;
    }

    // constructor if we know the breed, name, color at the time of creation
    // optional
    public Dog(String breed, String name, String Color)
    {
        this.breed = breed;
        this.name = name;
        this.color = color;
    }
    // ----------------------------------------------

    // Setters and Getters

    public String getBreed()
    {
        return breed;
    }
```

(continues on next page)

(continued from previous page)

```java
    public void setBreed(String breed)
    {
        this.breed = breed;
    }

    public String getName()
    {
        return name;
    }

    public void setName(String name)
    {
        this.name = name;
    }

    public String getColor()
    {
        return Color;
    }

    public void setColor(String Color)
    {
        this.Color = Color;
    }

    // --------------------------------

}
```

21.5 Methods

Defintion

Method This is a public function that can be accessed by the object. It allows the object to excecute code pertanent to the object.

In our dog example, if we want to make the dog bark we could write a method to do so.

```java
class Dog{

    // Define the attributes of a dog

    // --------- ******** ----------
    // Values can initially be set by the constructor
    // and changed via the Set function

    String breed;
    String name;
    String color;

    // NOTE: we can have as many constructors as we need.
    // If one is not defined,
```

(continues on next page)

(continued from previous page)

```
    // the a "default" constructor is generated behind the scenes.
    // It is always good form to include constructors

    // if no constructor is defined, then this one is the default
    public Dog()
    {
    }

    // constructor if we know the breed and name at the time of creation
    // optional
    public Dog(String breed, String name)
    {
        this.breed = breed;
        this.name = name;
    }

    // constructor if we know the breed, name, color at the time of creation
    // optional
    public Dog(String breed, String name, String Color)
    {
        this.breed = breed;
        this.name = name;
        this.color = color;
    }
    // ---------------------------------------------

    // Setters and Getters

    public String getBreed()
    {
        return breed;
    }

    public void setBreed(String breed)
    {
        this.breed = breed;
    }

    public String getName()
    {
        return name;
    }

    public void setName(String name)
    {
        this.name = name;
    }

    public String getColor()
    {
        return Color;
    }

    public void setColor(String Color)
    {
        this.Color = Color;
```

(continues on next page)

(continued from previous page)

```java
    }

    // -------------Methods-------------------
    public void Bark()
    {
        System.out.println("Woof Woof");
    }

}
```

21.6 Dog Completed Code

```java
/*
 * Programmer: James Goudy
 * Project: Dog Example
 *
 * Show how we can make multiple dogs using classes
 * Name the project: DogExample
 */

// Create a blueprint of a dog
// The characteristics(attributes) of a dog is:
// Breed, Name, Color
class Dog{

    // Define the attributes of a dog
    String breed;
    String name;
    String color;

    // Constructor
    // A constructor is a function(s)allows us to set initial values
    // to our attributes. It can be overloaded pending the values
    // we need to set at the time of creating the object. Note that
    // the constructor matches the name of the class and has no
    // return value.

    // NOTE: can have as many constructors as we need.  If one is not defined,
    // the a "default" constructor is generated behind the scenes.
    // It is always good form to include constructors when you write a class.

    // if no constructor is defined, then this one is the default
    public Dog()
    {
    }

    // constructor if we know the breed and name at the time of creation
    public Dog(String breed, String name)
    {
        this.breed = breed;
        this.name = name;
    }
```

(continues on next page)

(continued from previous page)

```
    // constructor if we know the breed, name, color at the time of creation
    public Dog(String breed, String name, String color)
    {
        this.breed = breed;
        this.name = name;
        this.color = color;
    }

    // Setters and Getters

    public String getBreed()
    {
        return breed;
    }

    public void setBreed(String breed)
    {
        this.breed = breed;
    }

    public String getName()
    {
        return name;
    }

    public void setName(String name)
    {
        this.name = name;
    }

    public String getcolor()
    {
        return color;
    }

    public void setcolor(String color)
    {
        this.color = color;
    }

    // --------- methods -------

    public void Bark()
    {
        System.out.println("Woof Woof");
    }

    public String allInfo()
    {
        String info = "";
        info = this.breed + " " + this.name + " " + this.color;
        return info;
     }

}
```

(continues on next page)

(continued from previous page)

```java
public class DogExample {

    public static void main(String[] args) {

        // Create a dog
        Dog myDog1 = new Dog();

        myDog1.setBreed("Collie");
        myDog1.setName("Fido");
        myDog1.setcolor("Brown");

        //Use getters to output information
        System.out.println(myDog1.getBreed() + " "
                            + myDog1.getName() + " "
                            + myDog1.getcolor());

        // use a method to output information
        System.out.println(myDog1.allInfo());

        // make the dog bark
        myDog1.Bark();

        // creat another dog
        // note this time, the breed, name, and color are known
        // at the time of creation

        Dog myDog2 = new Dog("Shepard", "Spot", "White");

        //Use getters to output information
        System.out.println(myDog2.getBreed() + " "
                            + myDog2.getName() + " "
                            + myDog2.getcolor());

        // use a method to output information
        System.out.println(myDog2.allInfo());

        // make the dog bark
        myDog2.Bark();
    }
}
```

21.7 In-class Exercise Suggestion

- Make cars

End Of Topic

CHAPTER TWENTYTWO

CLASSES - INHERITANCE

22.1 Key Ideas

- Inheritance
- *Multiple Classes*

22.2 Readings

https://books.trinket.io/thinkjava2/chapter14.html

https://www.geeksforgeeks.org/inheritance-in-java/

https://www.mygreatlearning.com/blog/polymorphism-in-java/

CHAPTER

TWENTYTHREE

CONCEPTS

Definition

Inheritance It is the ability in JAVA where *one* class is allowed to inherit the fields and methods of another class. It allows the programmer to build upon existing classes

Definition

Encapsulation In object-oriented computer programming (OOP) languages, the notion of encapsulation (or OOP Encapsulation) refers to the bundling of data, along with the methods that operate on that data, into a single unit. Many programming languages use encapsulation frequently in the form of classes. By Sumo Logic

Definition

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.

See Reference Reading

23.1 Lecture Code

```
import java.util.ArrayList;
import java.util.Scanner;

/*
 * Programmer: James Goudy
 * Project: PersonWorker Voter Class Lecture
 */



/*
    A class is like a blueprint.  It allows programes to
    create an "object" based on a class.  It follows the following
    naming schema.

        In main or a class other than itself.
```

(continues on next page)

(continued from previous page)

```
    classname  yourObjectname = new classname();

    example in main or other class other than PersonWorker

            PersonWorker myPerson = new PersonWorker();

    Classes usually start with a Capital Letter.

 */
class Person {

    /*
    These are private members -
    these variables can only be used
    by the functions within the class.
     */
    private String xFname;
    private String xLname;
    private String xCity;
    private String xState;

    /*
    Constructor
    This is used by the class in order to create a class.
    The constructor can also used to set defaults default values.

    All constructors are public and they match the name of the function.
     */
    public Person() {
        //This is the default construct.
    }

    //If the name of the of the person is known when we create the class
    //a constructor can be written to set it at the time of creation.
    public Person(String firstName, String lastName) {
        xFname = firstName;
        xLname = lastName;

    }

    //Note we can have as many constructors as we need, just as long
    //as we change the parameters of the function.
    public Person(String firstName, String lastName, String City, String State) {
        xFname = firstName;
        xLname = lastName;
        xCity = City;
        xState = State;
    }

    /*
        These are getters and setters.
        Setters allow other programs to set the values of the class object.
        Getters allow other programs to retrieve the values of the class object.
     */
    public String getFirstName() {
```

(continues on next page)

(continued from previous page)

```
        return xFname;
    }

    public void setFirstName(String FirstName) {
        xFname = FirstName;
    }

    public String getLastName() {
        return xLname;
    }

    public void setLastName(String LastName) {
        xLname = LastName;
    }

    public String getCity() {
        return xCity;
    }

    public void setCity(String City) {
        xCity = City;
    }

    public String getState() {
        return xState;
    }

    public void setState(String State) {
        xState = State;
    }

    //---------------------- We can include methods in our functions-----------
    /* Methods  members can be of three types.
        1. public -     can be used anywhere. Visible to all classes.
        2. private -    used only within the class.
                        Visible only within the class.
        3. protected -  used only within the package.
                        Visible only within package.
     */
    //Here is a method for concatenating the first and last name;
    public String getFullName() {
        String xFullName;
        xFullName = xFname + " " + xLname;
        return xFullName;
    }

    //Here is a method to calculate the value of income
    public double IncomeCalculateor(double NumberOfYears, double YearlySalary) {
        double ans = 0;
        try {
            ans = NumberOfYears * YearlySalary;
        } catch (Exception e) {
            System.out.println("\n\n****"
                    + "You had a error, please try again"
                    + "\n****\n");
        }
```

(continues on next page)

(continued from previous page)

```
        return ans;
    }

    //Here is a method for printing a mailing label
    public void MailingAddress()
    {
        System.out.println(xFname + " " + xLname + "\n"
                           + xCity + ", " + xState);

    }

}

// Example of inheritance
class Worker extends Person {

    //private members
    private String xCompany;
    private String xJobTitle;

    private double xSalary;
    //--------------------------------------------------

    //constructors
    //default constructor
    public Worker() {

    }

    public Worker(String FirstName, String LastName,
            String Company, String Title) {

        //notice that we use this to call the function
        //of the inherited class PersonWorker
        this.setFirstName(FirstName);
        this.setLastName(LastName);

        //set company name and title
        xCompany = Company;
        xJobTitle = Title;
    }

    //--------------------------------------------------
    //setters and getters
    public void setCompanyName(String CompanyName) {
        xCompany = CompanyName;
    }

    public String getCompanyName() {
        return xCompany;
    }

    public void setSalary(double Salary) {
        xSalary = Salary;
    }
```

(continues on next page)

(continued from previous page)

```
    public double getSalary() {
        return xSalary;
    }



}


public class PersonWorker {

    public static void main(String[] args) {
         // TODO code application logic here
        String full1, full2 = "";
        Scanner sc = new Scanner(System.in);
        double ans = 0;

        //Create A person
        Person p1 = new Person();
        //Add values to the person
        p1.setFirstName("Jim");
        p1.setLastName("Smith");
        ans = p1.IncomeCalculateor(5, 1000000.0);

        //Create a second person
        Person p2 = new Person();
        p2.setFirstName("Axle");
        p2.setLastName("Rod");

        full1 = p1.getFullName() + " " + ans;
        full2 = p2.getFullName();
        System.out.println(full1 + "\n" + full2);



        //Storing Objects in  an arraylist
        ArrayList<Person> myAl = new ArrayList();

        /* Notice how <Person> Here we are casting the
            arraylist to only take the datatype Person. This
            can also be done with every other data type. For example-
            Arraylist<double> myAl = new ArrayList();   //only accept doubles
            Arraylist<String> myAl = new ArrayList();   //only accept Strings
            Arraylist<int> myAl = new ArrayList();      //only accept ints
        */

        for (int cnt = 0; cnt < 4; cnt++) {
            Person xx = new Person();           //create a new person

            xx.setFirstName("Bob" + cnt);       //concantenate the counter
                                                //to the name so all the
                                                //Bobs will be unique

            xx.setLastName("Smith" + cnt);      //concantenate the counter
                                                //to the name so all the
                                                //Smiths will be unique
```

(continues on next page)

(continued from previous page)

```
            myAl.add(xx);                       //Add person xx to arraylist
        }


        for (int cntrx = 0; cntrx < myAl.size(); cntrx++) {

            System.out.println(  myAl.get(cntrx).getFullName());

        }


        Worker w1 = new Worker();
        w1.setFirstName("Bubbba");
        w1.setLastName("Smith");
        w1.setCompanyName("Police Academy");
        w1.setSalary(65000);
        full1 = w1.getFullName() + " " + w1.getSalary();
        System.out.println(full1);
        //</editor-fold>

        //create a worker using a constructor
        Worker w2 = new Worker("Jim", "Rodes", "Mickey's Brews", "Brewer");
        System.out.println("\n\n" + w2.getFullName() + "\n\n");



        //Create an array of workers
        Worker[] workers = new Worker[3];

        //Data for one worker in the array
        workers[0] = new Worker();
        workers[0].setFirstName("Billy");
        workers[0].setLastName("TheKid");
        System.out.println("Name = " + workers[0].getFullName());

        //Input workers in to the array by
        //asking the user for information
        //Note this can be done the same way for an arraylist
        for (int cnt = 0; cnt < workers.length; cnt++) {
            workers[cnt] = new Worker();
            System.out.println("Enter First Name");
            workers[cnt].setFirstName(sc.nextLine().toString());

            System.out.println("Enter Last Name");
            workers[cnt].setLastName(sc.nextLine());

            System.out.println("Enter Company Name");
            workers[cnt].setCompanyName(sc.nextLine());

            try {
                System.out.println("Enter Salary");
                workers[cnt].setSalary(sc.nextDouble());
                sc.nextLine();
            } catch (Exception e) {
                System.out.println("Error with salary - data not entered");
```

(continues on next page)

(continued from previous page)

```
                sc.nextLine();
            }

        }

        //Print out the contents of the array
        for (int cnt = 0; cnt < workers.length; cnt++) {
            full1 = workers[cnt].getFullName() + " "
                    + workers[cnt].getSalary() + " "
                    + workers[cnt].getCompanyName();

            System.out.println(full1 + "\n");
        }

    }

}
```

End Of Topic

CHAPTER TWENTYFOUR

ABSTRACT CLASSES

24.1 Key Ideas

- Abstract Classes

24.2 Readings

Abstract Classes by JournalDev

Using an Interface vs. Abstract Class in Java by Baeldung

Definition

Abstract Class - this is a class that is designated by the *abstract* keyword. This is a class that can only be used when it is inherited. The abstract class is mostly used to provide a base for subclasses.

24.3 Lecture Code

```
/*
 * Project: Abstract Classes Lecture Code
 * Programmer: James Goudy
 *
 */

// abstract class cannot be instantiated
// it must be inherited
abstract class Person {

    // Note that the properties are public
    // meaning they do not need a setter or getter and
    // can be accessed directly.
    public String firstName;
    public String lastName;

    // constructor
    public Person(String firstName, String lastName) {
        this.firstName = firstName;
```

(continues on next page)

(continued from previous page)

```
        this.lastName = lastName;
    }

}

class Worker extends Person {

    // Note that the properties are public
    // meaning they do not need a setter or getter and
    // can be accessed directly.
    public String jobTitle;
    public String company;

    // constructor
    public Worker(String jobTitle, String company,
            String firstName, String lastName) {
        super(firstName, lastName);
        this.jobTitle = jobTitle;
        this.company = company;
    }

    // super is telling the class to include the constructor
    // parent class - thus we can initalize firstName and lastName
    // in the constructor of the child class.

    public String allInfo()
    {
        return(firstName +" " + lastName +" " + jobTitle +" " +company);
    }
}

public class J1_Abstract_Classes {

    public static void main(String[] args) {

        System.out.println("\n\"Person myPerson = "
                + "new Person(\"John\",\"Doe\");\"");
        System.out.println("Will not compile since it is an "
                + "abstract class\n");

        // Person myPerson = new Person("John","Doe");
        // This will not compile since it is an abstract class

        // Worker will compile since it inherits Person
        Worker myWorker = new Worker("Programmer","XYZ Company","Joe", "Doe");
        System.out.println(myWorker.allInfo());

        System.out.println("\n\nbye\n");
    }
}
```

End Of Topic

CHAPTER

TWENTYFIVE

LAMBDA FUNCTIONS

25.1 Key Ideas

- Lambda functions

25.2 Readings

Lambda Expressions

Lambda Expression in Java 8

Definition

Lambda functions are intended as a shorthand for defining functions that can come in handy to write concise code without wasting multiple lines defining a function. They are also known as anonymous functions, since they do not have a name unless assigned one. From *What's in a Lambda?*

Definition

Interface - In its most common form, an interface is a group of related methods with empty bodies. The programmer is then responsible for writing the functions

Note

In most other languages, lambda functions do not need an interface and can be written as just as an anonmyous function. They all follow the same structure:

( ) -> { }

25.3 Lecture Code

```
/*

* Program: Lambda Examples
* Programmer: James Goudy

 */



// note that the interface can also be put in their own file
// note that a function is defined (which the programmer can name)
// in the interface.  The actual functin can then be written as
// a global function or within a function.

interface InchtesToFeet{
    // Input is a double - inches
    // Output is a double executed by the function name "calculate"
    double calculate(double inches);
}

interface FtoC{
    // Input is a double - farh
    // Output is a double executed by the function name "calculate"
    double calculate(double farh);

}

interface GalaxayTip{
    // Input is nothing
    // Output is a string executed by the function name "calculate"
    String run();
}

interface myFormula{
    // Input is two doubles
    // Output is a string executed by the function name "tabulate"
    String tabulate(double num1,double num2);

}

public class Lambda_Lecture {

    //Global Lamda functions
    static InchtesToFeet itf =(i)->{return i/12.0;};
    static GalaxayTip  tip = ()->{return "Always bring a towel";};

    // multiple 2 numbers, add 50 then concantenate string
    // note this lambda function has multiple lines of code
    static myFormula mf = (n1,n2) ->{

        // define local variable
        double ans = 0;

        ans = n1* n2 + 50;
```

(continues on next page)

(continued from previous page)

```
        return ans+ " units";
    };


    public static void main(String[] args) {

        int i = 36;
        double ft = 212;

        double xx = 10;
        double yy = 20;


        System.out.println(itf.calculate(i) + " feet");

        FtoC ftc = (f) -> {return (f-32.0) * 5.0/9.0;};
        System.out.println(ftc.calculate(ft) + " C");
        System.out.println(ftc.calculate(392.0) + " C");

        System.out.println(tip.run());

        System.out.println(mf.tabulate(xx, yy));

    }
}

/*
3.0 feet
100.0 C
200.0 C
Always bring a towel
250.0 units
 */
```

End of Topic

CHAPTER TWENTYSIX

RECURSION

26.1 Key Topics

- Recursion
- Tail Recursion

26.2 Readings

- Recursion - Geeks For Geeks
- Recursion - Wikipedia
- Tail vs Non-Tail Recursion by Baeldung
- Recursive by Computer Hope

CHAPTER

TWENTYSEVEN

VIDEOS

https://youtu.be/ngCos392W4w

Definition

Recursion A function that calls itself.

Definition

Tail Recursion A tail recursion is also a kind of recursion but it will make the return value of the recursion call as the last statement of the method.

Warning: Java does not optimize for tail recursion

Tip: In order to trap a stack over flow error, it has to be specified in the catch statment as shown below:

```java
        try {
          // posible code that could stack overflow
        } catch (StackOverflowError e) {

            System.out.println("RECURSION: STACK OVERFLOW ERROR");
            System.out.println(e.getMessage());
        }
```

27.1 Lecture Code

Any recursive function can always be written as a loop. Note there is an example of what a tail-recursive function would look like if Java supported it. Many languages do not support tail-recursion. Scala, Haskell, and some others do support it. This is important to know because sometimes this is asked in interview questions.

```java
/*
 *
 * Project: Recursion Lecture Code
```

(continues on next page)

(continued from previous page)

```
 * Programmer: James Goudy
 *
 */


public class Ds_recursion {

    static double sumNums_Loop(int n) {

        double sum = 0;

        for (int i = n; i > 0; i--) {
            sum = sum + i;
        }

        return sum;
    }

    // This will eventually cause a stack overflow error for big "n"'s.
    // Every time the function is called recursively and
    // intermediate n is saved in the program stack which
    // will result in running out of resources / stackoverflow error

    static double sumNums_Recursive(int n) {

        if (n == 0) {
            return n;
        }

        return n + sumNums_Recursive(n - 1);

    }

    // TAIL Recursion
    // NOTE: Java as of JDK 17 does not optimization for tail recursion
    // If it did, it would look like the following:
    // The reason this is a tail recursion is the sum
    // is being calculated everytime and the function
    // does not have to store any temp values to retreive the calculated answer

    static double sumNums_TailRecursive(int n, double sum) {

        if (n <= 1) {
            return sum;
        }

        return sumNums_TailRecursive(n - 1, sum + 1);

    }

    public static void main(String[] args) {

        int n = Integer.MAX_VALUE;
        double ans = 0;

        n = 200;
```

(continues on next page)

(continued from previous page)

```
        System.out.println("Max integer: " + Integer.MAX_VALUE);
        System.out.println("n = " + n + "\n");

        System.out.println("Loop: Sum of all numbers " + n
                + " = " + sumNums_Loop(n));

        System.out.println("Recursive: Sum of all numbers " + n
                + " = " + sumNums_Recursive(n));


        System.out.println("\n *************\n");

        // set a very big "n"
        n = 200000000;

        System.out.println("n = " + n + "\n");

        System.out.println("Loop: Sum of all numbers " + n
                + " = " + sumNums_Loop(n));

        // The following will fail recursively.
        //  This will error because it will run out of resources
        //  on the program stack.
        // Note how we are catching specifically for a StackOverFlow error.

        try {
            System.out.println("Recursive: Sum of all numbers " + n
                    + " = " + sumNums_Recursive(n));
        } catch (StackOverflowError e) {

            System.out.println("RECURSION: STACK OVERFLOW ERROR");
            System.out.println(e.getMessage());
        }

        System.out.println("\nbye\n");
    }
}


/*
Max integer: 2147483647
n = 200

Loop: Sum of all numbers 200 = 20100.0
Recursive: Sum of all numbers 200 = 20100.0

 *************

n = 200000000

Loop: Sum of all numbers 200000000 = 2.00000000174137712E16
RECURSION: STACK OVERFLOW ERROR
null

bye
 */
```

27.2 Lecture Code II

Note that in this example a recursive function as another recursive function called within it. Note that it will print the output of the first recursive function first all at once. Then it goes back for each time and prints the outcome separately starting with the last iteration of the "top" recursive function.

```
/*
 * Programmer: James Goudy
 * Project: Lecture Code Recursion II
 */


/**
 *
 * @author jgoudy
 */
public class DS_Recursion_Example_II {

    static String alp = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghicklmnopqrstuvwxyz";
    static int stop = 4;

    static void recursiveLetter(int a) {
        if (a > stop) {
            return;
        }

        System.out.println("a = " + alp.charAt(a) + " - " + a);
        a++;
        recursiveLetter(a);


    }

    static void recursiveCount(int n) {
        if (n > stop) {

            return;
        }

        System.out.println("n = " + n);
        n++;

        recursiveCount(n);
        System.out.println("----------");
        recursiveLetter(n);

    }

    public static void main(String[] args) {

        System.out.println("L = " + alp.length());
        System.out.println("");

        recursiveCount(0);

        System.out.println("\n****** Call recursive letters only *********\n");
```

(continues on next page)

(continued from previous page)

```
        recursiveLetter(0);

    }
}
/*

Output

L = 52

n = 0
n = 1
n = 2
n = 3
n = 4
----------
----------
a = E - 4
----------
a = D - 3
a = E - 4
----------
a = C - 2
a = D - 3
a = E - 4
----------
a = B - 1
a = C - 2
a = D - 3
a = E - 4

****** Call recursive letters only *********

a = A - 0
a = B - 1
a = C - 2
a = D - 3
a = E - 4
------------
 */
```

End Of Topic

CHAPTER

TWENTYEIGHT

REGEX - USING JAVA MATCHES

Definition

A **regular expression** (shortened as regex or regexp; sometimes referred to as rational expression) is a sequence of characters that specifies a search pattern in text. Usually such patterns are used by string-searching algorithms for "find" or "find and replace" operations on strings, or for input validation. *https://en.wikipedia.org/wiki/Regular_expression*

28.1 From Stack Overflow

https://stackoverflow.com/questions/8923398/regex-doesnt-work-in-string-matches

Welcome to Java's misnamed `.matches()` method… It tries and matches ALL the input. Unfortunately, other languages have followed suit :(

If you want to see if the regex matches an input text, use a `Pattern`, a `Matcher` and the `.find()` method of the matcher:

```
Pattern p = Pattern.compile("[a-z]");
Matcher m = p.matcher(inputstring);
if (m.find())
    // match
```

If what you want is indeed to see if an input only has lowercase letters, you can use `.matches()`, but you need to match one or more characters: append a + to your character class, as in `[a-z]+`. Or use `^[a-z]+$` and `.find()`.

28.2 Lecture Code

```
/*
 * Programmer: James Goudy
 * Project: Using Regex to determine valid characters in a string
 */
package com.mycompany.java_regexexpressions;

import java.util.regex.Matcher;
import java.util.regex.Pattern;

```

(continues on next page)

(continued from previous page)

```
/**
 *
 * @author jgoudy
 */
public class Java_RegExExpressions {

    public static void main(String[] args) {
        boolean status = true;

        // Test - only want a string expression to only allow
        // the any capital letters of A B C D,
        // the any lowercase letters of l m n o p,
        // and any numbers of 4 5 6 7 8
        // The regex Expression looks like this ^[A-Dl-p4-8]+$

        // Note in order to work correctly in JAVA
        // it must start with the ^ and end with the $

        String regex_Expression = "^[A-Dl-p4-8]+$";
        System.out.println("Regex Expression = " + regex_Expression);

        // this string should test false since FF is not allowed
        String testPattern = "AAFFlmno78";
        System.out.println("\nFalse test pattern: " + testPattern);

        Pattern p = Pattern.compile(regex_Expression);
        Matcher m = p.matcher(testPattern);
        status = m.find();
        System.out.println("status = " + status);

        // this pattern should test true
        testPattern = "BBDDoooml55447";
        System.out.println("\nTrue test pattern: " + testPattern);

        p = Pattern.compile(regex_Expression);
        m = p.matcher(testPattern);
        status = m.find();
        System.out.println("status = " + status);

        // note that m.find() is a boolean and could be used
        // in if statements, loops, etc.
    }
}

/* Output

Regex Expression = ^[A-Dl-p4-8]+$

False test pattern: AAFFlmno78
status = false

True test pattern: BBDDoooml55447
status = true

 */
```

28.3 Regular Expression References

Regular Expressions - Java

Regular Expressions - Microsoft

Regular Expression - Google / Python

Regular Expressions - Mozilla

Endo Of Topic

CHAPTER

TWENTYNINE

REFERENCE READING

29.1 Additional supplemental topics.

Articles you find helpful

Unable To Run .JAR Files in Windows 10 / 11 ? Here's the Solution

29.2 Building Large Java Applications

Author: Ray Toal

Real Java applications are made up of hundreds, or thousands, of classes. Designing and building an application of this size isn't at all like building something tiny.

CONTENTS

Java Applications • Organizing Your Development Environment • Basics of Class Loading • Building Your Application

29.2.1 Java Applications

A Java application is made up of **classes** and **resources**. Resources can be videos, songs, images, text files, data files, etc. There can be thousands of classes and thousands of resource files in one application. Often you will write only some of the classes yourself, and simply use many classes written by others.

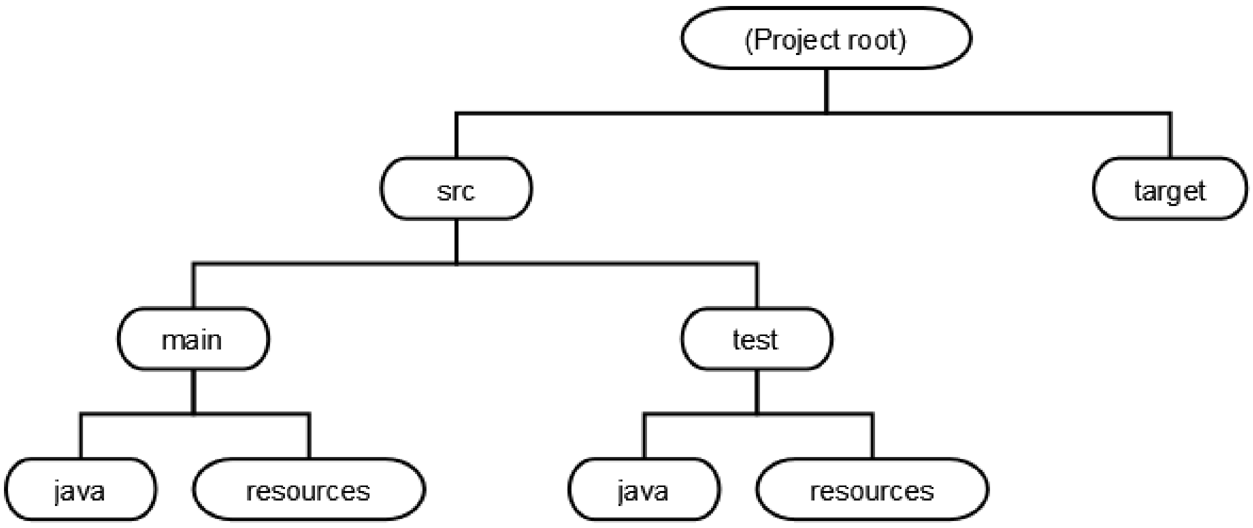
There are three main types of Java applications:

- Native applications, which are stored on, and run directly on, the user's desktop, laptop, or mobile device.
- Web applications, served by, and run mostly on, a web server.
- WebStart applications, which are stored on a server, but downloaded and run on demand.

Generally, all of the classes and resources are zipped up into a single file. For client apps and webstart apps, this is a JAR file; for webapps this is a WAR file. Both jars and wars use the popular zip format (nice and portable). Note: with webstart apps, you actually need two files, since the JNLP file has to live outside the war.

29.2.2 Organizing Your Development Environment

When dealing with large programs it is important that you keep your workspace organized. The following directory layout has become standard; you should follow it:

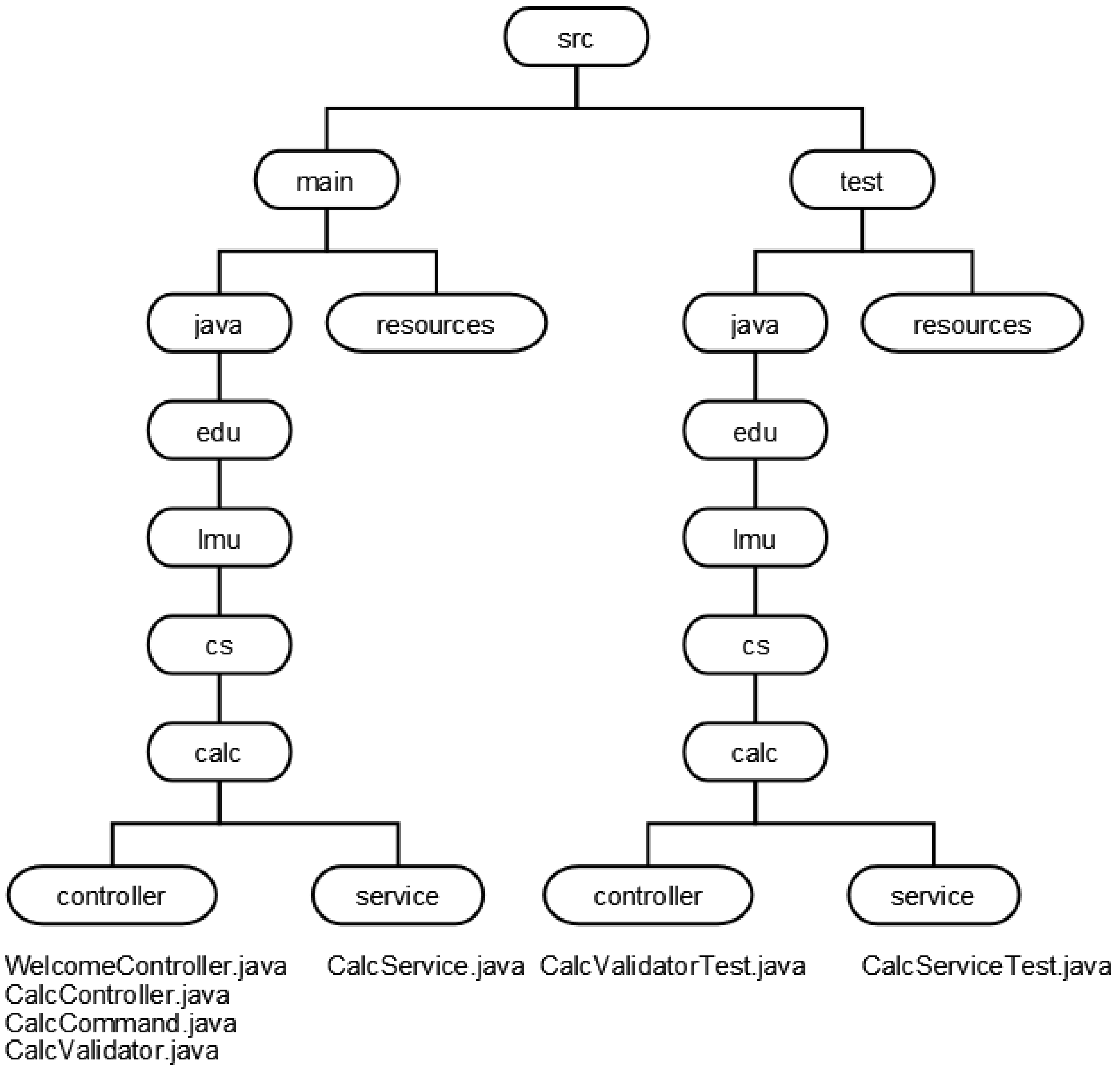


Why is this so great?

- Everything under **target** is "derived" and therefore does *not go into revision control*. For example you put your Java source code files under **src/main/java**. When these are compiled, the Java compiler places the resulting class files under **target**.
- Everything under **src/main** goes in your distributed application (you don't bundle your tests in your app).

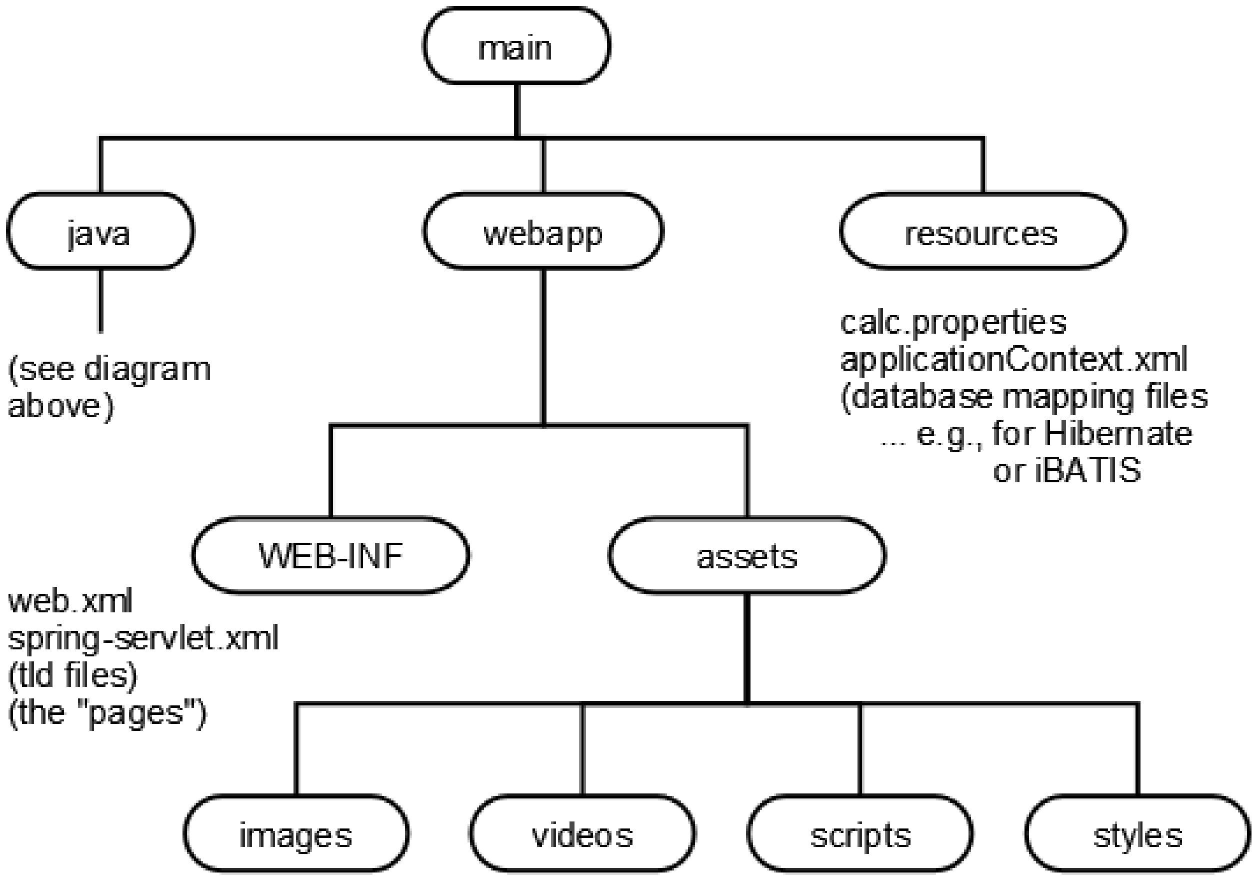
Since you may have hundreds of Java source files, you need to group them into packages. Make sure the initial part of the package name follows the convention of using your "reversed" Internet domain (such as **edu.lmu.cs**. The folder structure of your application must mirror the package names! Here is an example to explain:

Suppose we are writing a calculator web application. We would need Java classes for controllers, validators, and services, and perhaps some tests. Our project structure should be:



Of course a larger application might have many subpackages within an application, perhaps one per each functional subsystem of the project. And within each subsystem, we may need more than just **controller** and **service**; we might want separate packages for validators, domain classes, types, utility classes, factories, daos, property editors, etc.

Now if we were building a client application, we would probably place imeages, scripts, movies and so on into the **resources** folder, but the convention is to structure these things under **src/main/webapp** as shown below. The **resources** folder does keep properties files and other run-time configuration files (e.g., Hibernate mapping files). To make the example interesting, we'll assume that we are building our application using the Spring Framework.



Now let's see how Java applications are built and run. (In doing so, we'll see why the folder structure needs to match the package name structure.

29.2.3 Basics of Class Loading

So all Java applications are made up of a bunch of classes, grouped into packages. But when an application is being compiled and run, how does the system find all the classes? It uses **classloaders** to find and load the classes (and other resources like images, movies, and properties files) that the program requires.

The Java runtime starts with one classloader (the *bootstrap classloader*) that finds classes on the local filesystem. You can create your own classloaders to find classes from other places (such as over a network connection), to compose binary classfiles on the fly, or do things while loading classes (like decrypting them, verifying digital signatures, etc.). Classloaders are chained together so that each classloader except the bootstrap loader has a parent; if a classloader can't find a class it asks its parent to find it.

Every `Class` object contains a reference to the `ClassLoader` that defined it; you'll sometimes see code that says

```
x.getClass().getClassLoader()
```

The Bootstrap Classloader

Unless you are doing fairly fancy stuff, you never have to tell a classloader to load a class; just mentioning a class is good enough. For example, the class A below refers to:

- other classes in the same package
- classes from other packages
- classes from a library written by someone else
- classes from the Java Core API

src/main/java/edu/lmu/cs/scratch/A.java

```
package edu.lmu.cs.scratch;
public class A {
    org.bouncycastle.crypto.BlockCipher cipher =
        new com.citysearch.util.crypto.HagenRedmannTwofishEngine();

    void m(int x) {
        java.util.List y = new java.util.ArrayList();
        cipher.reset();
        B b = new B();
        edu.lmu.cs.math.Complex c;
        java.lang.System.out.println("okay");
    }
}
```

src/main/java/edu/lmu/cs/scratch/B.java

```
package edu.lmu.cs.scratch;
public class B {}
```

src/main/java/edu/lmu/cs/math/Complex.java

```
package edu.lmu.cs.math;
public class Complex {
    // Class body goes here...
}
```

So I've written classes A, B, and Complex myself, but these other classes (BlockCipher, HagenRedmannTwofishEngine, List, ArrayList, and System) already exist. But it doesn't really matter who wrote them; what matters is the full class names. A classloader needs to find:

```
edu.lmu.cs.scratch.A
edu.lmu.cs.scratch.B
edu.lmu.cs.math.Complex
org.bouncycastle.crypto.BlockCipher
com.citysearch.util.crypto.HagenRedmannTwofishEngine
java.util.List
java.util.ArrayList
java.lang.System
```

But how does the bootstrap class loader find all the classes? First, it turns the full class name into a filename — so the first class in the list above would correspond, on a Windows system, to

```
edu\lmu\cs\scratch\A.class
```

(If you're compiling, and the compiler can't find that class, it will look for

```
edu\lmu\cs\scratch\A.java
```

and compile that for you!!! How sweet.)

On almost every other system, the file names would be:

```
edu/lmu/cs/scratch/A.class
(or edu/lmu/cs/scratch/A.java)
```

Note: that filename is a *relative* path name! Relative to what? The bootstrap loader looks for it in this order:

1. The jar files in your JRE's lib directory (rt.jar, jsse.jar, jce.jar, charsets.jar)
2. The jar files in your *extensions directory* — by default this is the JRE's lib/ext directory. You can dump your own jars in this directory, or tell your tool to use a different directory.
3. Each entry in the current *classpath*, in order.

Classpath? What's that?

Classpaths

A classpath is simply a list of directories and jar files. The bootstrap class loader searches a classpath when it looks for the classes (or source files) it needs, **after searching the platform and extension locations**.

On Windows the classpath entries are separated with semicolons; on every other platform (I think) colons are used.

Example

```
c:\homework\stuff.jar;c:\other\crap;c:\mylibs\junit.jar
```

If you requested the class a.b.C from the bootstrap classloader, and that class was not found in rt.jar or in the extensions, it would look for, in this order:

1. a\b\C.class in c:\homework\stuff.jar
2. c:\other\crap\a\b\C.class
3. a\b\C.class in c:\mylibs\junit.jar

Specify the classpath when invoking a tool, for example

```
javac -cp c:\homework\stuff.jar;c:\other\crap;c:\mylibs\junit.jar *.java
```

or, less flexibly, set the CLASSPATH environment variable (which may seem like a timesaver but can cause headaches). It's suggested you leave this environment variable unset and use a specific classpath when you invoke a tool. If you really must know about this variable, consult Sun's online docs.

More on Classloaders, Classpaths, and Related Topics

See Sun’s documentation on How Classes are Found.

Also note: once you start writing your own classloaders or you start deploying applications with multiple classloaders, you’ll probably run into the case where a single class file is loaded by each classloader. Your application will think there are two distinct classes, and, well, some confusing things may start to happen. Just be aware.

29.2.4 Building Your Application

Developers should be familiar with all three main approaches to building applications

- Using Commandline Tools

  Everything you need to know is in Sun’s tool documentation.

- Using Build Files

  The defacto standard application for building Java applications is Maven. (You can also use Ant, which is older.)

- Using IDEs

  Once you understand how to build applications the hard way, you’re ready to fire up an IDE and use a nice tool to construct classpaths for your projects. Some IDEs just let you drag and drop jar files and directories into a window. However you do it, you need to understand what a classpath is. Note that you can use maven and ant plugins for modern IDEs, too!

Experienced programmers will get the most benefit out of an IDE and a Maven plugin. Beginners should probably pay their dues by using the command line first. This gives you the best “sense” of how things fit together and are supposed to work, enabling you to work much more efficiently with IDEs and plugins when you finally start using them.

End Of Topic

29.3 Ant Vs Maven Vs Gradle

Author: Baeldung

29.3.1 Introduction

In this article, we’ll explore three Java build automation tools that dominated the JVM ecosystem – Ant, Maven, and Gradle.

We’ll introduce each of them and explore how Java build automation tools evolved.

29.3.2 Apache Ant

In the beginning, Make was the only build automation tool available beyond homegrown solutions. Make has been around since 1976 and as such, it was used for building Java applications in the early Java years.

However, a lot of conventions from C programs didn't fit in the Java ecosystem, so in time Ant took over as a better alternative.

Apache Ant ("Another Neat Tool") is a Java library used for automating build processes for Java applications. Additionally, Ant can be used for building non-Java applications. It was initially part of Apache Tomcat codebase and was released as a standalone project in 2000.

In many aspects, Ant is very similar to Make, and it's simple enough so anyone can start using it without any particular prerequisites. Ant build files are written in XML, and by convention, they're called build.xml.

Different phases of a build process are called "targets".

Here is an example of a build.xml file for a simple Java project with the HelloWorld main class:

```xml
<project>
    <target name="clean">
        <delete dir="classes" />
    </target>

    <target name="compile" depends="clean">
        <mkdir dir="classes" />
        <javac srcdir="src" destdir="classes" />
    </target>

    <target name="jar" depends="compile">
        <mkdir dir="jar" />
        <jar destfile="jar/HelloWorld.jar" basedir="classes">
            <manifest>
                <attribute name="Main-Class"
                  value="antExample.HelloWorld" />
            </manifest>
        </jar>
    </target>

    <target name="run" depends="jar">
        <java jar="jar/HelloWorld.jar" fork="true" />
    </target>
</project>
```

This build file defines four targets: clean, compile, jar and run. For example, we can compile the code by running:

ant compile This will trigger the target clean first which will delete the "classes" directory. After that, the target compile will recreate the directory and compile the src folder into it.

The main benefit of Ant is its flexibility. Ant doesn't impose any coding conventions or project structures. Consequently, this means that Ant requires developers to write all the commands by themselves, which sometimes leads to huge XML build files that are hard to maintain.

Since there are no conventions, just knowing Ant doesn't mean we'll quickly understand any Ant build file. It'll likely take some time to get accustomed to an unfamiliar Ant file, which is a disadvantage compared to the other, newer tools.

At first, Ant had no built-in support for dependency management. However, as dependency management became a must in the later years, Apache Ivy was developed as a sub-project of the Apache Ant project. It's integrated with Apache Ant, and it follows the same design principles.

However, the initial Ant limitations due to not having built-in support for dependency management and frustrations when working with unmanagable XML build files led to the creation of Maven.

Apache Maven

Apache Maven is a dependency management and a build automation tool, primarily used for Java applications. Maven continues to use XML files just like Ant but in a much more manageable way. The name of the game here is convention over configuration.

While Ant gives flexibility and requires everything to be written from scratch, Maven relies on conventions and provides predefined commands (goals).

Simply put, Maven allows us to focus on what our build should do, and gives us the framework to do it. Another positive aspect of Maven was that it provided built-in support for dependency management.

Maven's configuration file, containing build and dependency management instructions, is by convention called pom.xml. Additionally, Maven also prescribes a strict project structure, while Ant provides flexibility there as well.

Here's an example of a pom.xml file for the same simple Java project with the HelloWorld main class from before:

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
      http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>baeldung</groupId>
    <artifactId>mavenExample</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <description>Maven example</description>

    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
</project>
```

However, now the project structure has been standardized as well and conforms to the Maven conventions:

> +—src | +—main | | +—java | | | —com | | | —baeldung | | | —maven | | | HelloWorld.java | | || | —resources | —test | +—java | —resources

As opposed to Ant, there is no need to define each of the phases in the build process manually. Instead, we can simply call Maven's built-in commands.

For example, we can compile the code by running:

mvn compile At its core, as noted on official pages, Maven can be considered a plugin execution framework, since all work is done by plugins. Maven supports a wide range of available plugins, and each of them can be additionally configured.

One of the available plugins is Apache Maven Dependency Plugin which has a copy-dependencies goal that will copy our dependencies to a specified directory.

To show this plugin in action, let's include this plugin in our pom.xml file and configure an output directory for our dependencies:

```
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-dependency-plugin</artifactId>
            <executions>
                <execution>
                    <id>copy-dependencies</id>
                    <phase>package</phase>
                    <goals>
                        <goal>copy-dependencies</goal>
                    </goals>
                    <configuration>
                        <outputDirectory>target/dependencies
                          </outputDirectory>
                    </configuration>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

This plugin will be executed in a package phase, so if we run:

mvn package We'll execute this plugin and copy dependencies to the target/dependencies folder.

There is also an existing article on how to create an executable JAR using different Maven plugins. Additionally, for a detailed Maven overview, have a look at this core guide on Maven, where some Maven's key features are explored.

Maven became very popular since build files were now standardized and it took significantly less time to maintain build files, comparing to Ant. However, though more standardized than Ant files, Maven configuration files still tend to get big and cumbersome.

Maven's strict conventions come with the price of being a lot less flexible than Ant. Goal customization is very hard, so writing custom build scripts is a lot harder to do, compared with Ant.

Although Maven has made some serious improvements regarding making application's build processes easier and more standardized, it still comes with a price due to being a lot less flexible than Ant. This lead to the creation of Gradle which combines the best of both worlds – Ant's flexibility and Maven's features.

Gradle

Gradle is a dependency management and a build automation tool that was built upon the concepts of Ant and Maven.

One of the first things we can note about Gradle is that it's not using XML files, unlike Ant or Maven.

Over time, developers became more and more interested in having and working with a domain-specific language – which, simply put, would allow them to solve problems in a specific domain using a language tailored for that particular domain.

This was adopted by Gradle, which is using a DSL based either on Groovy or Kotlin. This led to smaller configuration files with less clutter since the language was specifically designed to solve specific domain problems. Gradle's configuration file is by convention called build.gradle in Groovy, or build.gradle.kts in Kotlin.

Notice that Kotlin offers better IDE support than Groovy for auto-completion and error detection.

Here is an example of a build.gradle file for the same simple Java project with the HelloWorld main class from before:

apply plugin: 'java'

```
repositories {
    mavenCentral()
}

jar {
    baseName = 'gradleExample'
    version = '0.0.1-SNAPSHOT'
}

dependencies {
    testImplementation 'junit:junit:4.12'
}
```

We can compile the code by running:

Gradle classes At its core, Gradle intentionally provides very little functionality. Plugins add all useful features. In our example, we were using java plugin which allows us to compile Java code and other valuable features.

Gradle gave its build steps name "tasks", as opposed to Ant's "targets" or Maven's "phases". With Maven, we used Apache Maven Dependency Plugin, and it's a specific goal to copy dependencies to a specified directory. With Gradle, we can do the same by using tasks:

task copyDependencies(type: Copy) { from configurations.compile into 'dependencies' } We can run this task by executing:

gradle copyDependencies 5. Conclusion In this article, we presented Ant, Maven, and Gradle – three Java build automation tools.

Not surprisingly, Maven holds the majority of the build tool market today.

Gradle, however, has seen good adoption in more complex codebases, for the following reasons:

- Lots of open-source projects such as Spring are using it now
- It is faster than Maven for most scenarios, thanks to its incremental builds
- It offers advanced analysis and debugging services

However that Gradle seems to have a steeper learning curve, especially if you're not familiar with Groovy or Kotlin.

End Of Topic

29.4 Polymorphism in Java (With Examples)

By Great Learning Team

https://www.mygreatlearning.com/blog/polymorphism-in-java/

Polymorphism is the ability of an object to take on different forms. In Java, polymorphism refers to the ability of a class to provide different implementations of a method, depending on the type of object that is passed to the method.

To put it simply, polymorphism in Java allows us to perform the same action in many different ways. Any Java object that can pass more than one IS-A test is polymorphic in Java. Therefore, all the Java objects are polymorphic as it has passed the IS-A test for their own type and for the class Object.

This article also talks about two types of polymorphism in Java: compile time polymorphism and runtime polymorphism, Java polymorphism examples, method overloading, method overriding, why to use polymorphism in java, java programming, and many more.

Polymorphism is a feature of the object-oriented programming language, Java, which implies that you can perform a single task in different ways. In the technical world, polymorphism in Java allows one to do multiple implementations by defining one interface.

29.4.1 What is Polymorphism?

The derivation of the word Polymorphism is from two different Greek words- poly and morphs. “Poly” means numerous, and “Morphs” means forms. So, polymorphism means innumerable forms. Polymorphism, therefore, is one of the most significant features of Object-Oriented Programming.

29.4.2 What is Polymorphism in Java?

Polymorphism in Java is the task that performs a single action in different ways.

So, languages that do not support polymorphism are not ‘Object-Oriented Languages’, but, ‘Object-Based Languages’. Ada, for instance, is one such language. Since Java supports polymorphism, it is an Object-Oriented Language.

Polymorphism occurs when there is inheritance, i.e. there are many classes that are related to each other.

Inheritance is a powerful feature in Java. Java Inheritance lets one class acquire the properties and attributes of another class. Polymorphism in Java allows us to use these inherited properties to perform different tasks. Thus, allowing us to achieve the same action in many different ways.

29.4.3 Real-Life Examples of Polymorphism

An individual can have different relationships with different people. A woman can be a mother, a daughter, a sister, a friend, all at the same time, i.e. she performs other behaviours in different situations.

The human body has different organs. Every organ has a different function to perform; the heart is responsible for blood flow, lungs for breathing, brain for cognitive activity, and kidneys for excretion. So we have a standard method function that performs differently depending upon the organ of the body.

Polymorphism in Java Example

A superclass named "Shapes" has a method "area()". Subclasses of "Shapes" can be "Triangle", "circle", "Rectangle", etc. Each subclass has its way of calculating area. Using Inheritance and Polymorphism means, the subclasses can use the "area()" method to find the area's formula for that shape.

```java
class Shapes {
  public void area() {
    System.out.println("The formula for area of ");
  }
}
class Triangle extends Shapes {
  public void area() {
    System.out.println("Triangle is ½ * base * height ");
  }
}
class Circle extends Shapes {
  public void area() {
    System.out.println("Circle is 3.14 * radius * radius ");
  }
}
class Main {
  public static void main(String[] args) {
    Shapes myShape = new Shapes();  // Create a Shapes object
    Shapes myTriangle = new Triangle();  // Create a Triangle object
    Shapes myCircle = new Circle();  // Create a Circle object
    myShape.area();
    myTriangle.area();
    myShape.area();
    myCircle.area();
  }
}
```

Output:

The formula for the area of Triangle is ½ * base * height The formula for the area of the Circle is 3.14 * radius * radius

Also Read: **OOPs concepts in Java**

29.4.4 Types of Polymorphism

You can perform Polymorphism in Java via two different methods:

1. Method Overloading
2. Method Overriding

What is Method Overloading in Java?

Method overloading is the process that can create multiple methods of the same name in the same class, and all the methods work in different ways. Method overloading occurs when there is more than one method of the same name in the class.

Example of Method Overloading in Java

```java
class Shapes {
  public void area() {
    System.out.println("Find area ");
  }
public void area(int r) {
    System.out.println("Circle area = "+3.14*r*r);
  }

public void area(double b, double h) {
    System.out.println("Triangle area="+0.5*b*h);
  }
public void area(int l, int b) {
    System.out.println("Rectangle area="+l*b);
  }


}

class Main {
  public static void main(String[] args) {
    Shapes myShape = new Shapes();  // Create a Shapes object

    myShape.area();
    myShape.area(5);
    myShape.area(6.0,1.2);
    myShape.area(6,2);

  }
}
```

Output:

Find area Circle area = 78.5 Triangle area=3.60 Rectangle area=12

What is Method Overriding in Java?

Method overriding is the process when the subclass or a child class has the same method as declared in the parent class.

Example of Method Overriding in Java

```
class Vehicle{
  //defining a method
  void run(){System.out.println("Vehicle is moving");}
}
//Creating a child class
class Car2 extends Vehicle{
  //defining the same method as in the parent class
  void run(){System.out.println("car is running safely");}

  public static void main(String args[]){
  Car2 obj = new Car2();//creating object
  obj.run();//calling method
  }
}
```

Output:

```
Car is running safely
```

Also, Polymorphism in Java can be classified into two types, i.e:

1. Static/Compile-Time Polymorphism
2. Dynamic/Runtime Polymorphism

What is Compile-Time Polymorphism in Java?

Compile Time Polymorphism In Java is also known as **Static Polymorphism.** Furthermore, the call to the method is resolved at compile-time. Compile-Time polymorphism is achieved through **Method Overloading**. This type of polymorphism can also be achieved through **Operator Overloading**. However, Java does not support Operator Overloading.

Method Overloading is when a class has multiple methods with the same name, but the number, types, and order of parameters and the return type of the methods are different. Java allows the user freedom to use the same name for various functions as long as it can distinguish between them by the type and number of parameters.

Example of Compile-Time Polymorphism in Java

We will do addition in Java and understand the concept of compile time polymorphism using subtract()

```
package staticPolymorphism;
public class Addition
{
void sum(int a, int b)
{
int c = a+b;
System.out.println(“ Addition of two numbers :” +c); }
void sum(int a, int b, int e)
```

(continues on next page)

(continued from previous page)

```
{
int c = a+b+e;
System.out.println(“ Addition of three numbers :” +c); }
public static void main(String[] args)
{
Addition obj = new Addition();
obj.sum ( 30,90);
obj.sum(45, 80, 22);
}
}
```

The output of the program will be:

Sum of two numbers: 120

Sum of three numbers: 147

In this program, the sum() method overloads with two types via different parameters.

This is the basic concept of compile-time polymorphism in java where we can perform various operations by using multiple methods having the same name.

What is Runtime Polymorphism in Java?

Runtime polymorphism in Java is also popularly known as **Dynamic Binding or Dynamic Method Dispatch.** In this process, the call to an overridden method is resolved dynamically at runtime rather than at compile-time. You can achieve Runtime polymorphism via **Method Overriding**.

Method Overriding is done when a child or a subclass has a method with the same name, parameters, and return type as the parent or the superclass; then that function overrides the function in the superclass. In simpler terms, if the subclass provides its definition to a method already present in the superclass; then that function in the base class is said to be overridden.

Also, it should be noted that runtime polymorphism can only be achieved through functions and not data members.

Overriding is done by using a reference variable of the superclass. The method to be called is determined based on the object which is being referred to by the reference variable. This is also known as **Upcasting**.

Upcasting takes place when the Parent class’s reference variable refers to the object of the child class. For example:

```
class` `A{} ``class` `B ``extends` `A{} ``A a=``new` `B(); ``//upcasting
```

Examples of Runtime Polymorphism in Java

Example 1:

In this example, we are creating one superclass Animal and three subclasses, Herbivores, Carnivores, and Omnivores. Subclasses extend the superclass and override its eat() method. We will call the eat() method by the reference variable of Parent class, i.e. Animal class. As it refers to the base class object and the base class method overrides the superclass method; the base class method is invoked at runtime. As Java Virtual Machine or the JVM and not the compiler determines method invocation, it is, therefore, runtime polymorphism.

```
class Animal{
  void eat(){
System.out.println("Animals Eat");
```

(continues on next page)

(continued from previous page)

```
}
}
class herbivores extends Animal{
  void eat(){
System.out.println("Herbivores Eat Plants");
}
  }
class omnivores extends Animal{
  void eat(){
System.out.println("Omnivores Eat Plants and meat");
}
  }
class carnivores extends Animal{
  void eat(){
System.out.println("Carnivores Eat meat");
}
  }
class main{
  public static void main(String args[]){
    Animal A = new Animal();
    Animal h = new herbivores(); //upcasting
    Animal o = new omnivores(); //upcasting
    Animal c = new carnivores(); //upcasting
    A.eat();
    h.eat();
    o.eat();
    c.eat();

  }
}
```

Output:

Animals eat Herbivores Eat Plants Omnivores Eat Plants and meat Carnivores eat meat

Example 2:

In this example, we are creating one superclass Hillstations and three subclasses Manali, Mussoorie, Gulmarg. Subclasses extend the superclass and override its location() and famousfor() method. We will call the location() and famousfor() method by the Parent class', i.e. Hillstations class. As it refers to the base class object and the base class method overrides the superclass method; the base class method is invoked at runtime. Also, as Java Virtual Machine or the JVM and not the compiler determines method invocation, it is runtime polymorphism.

```
class Hillstations{
  void location(){
System.out.println("Location is:");
}
void famousfor(){
System.out.println("Famous for:");
}

}
class Manali extends Hillstations {
  void location(){
System.out.println("Manali is in Himachal Pradesh");
}
void famousfor(){
```

(continues on next page)

(continued from previous page)

```
System.out.println("It is Famous for Hadimba Temple and adventure sports");
}
  }
class Mussoorie extends Hillstations {
  void location(){
System.out.println("Mussoorie is in Uttarakhand");
}
void famousfor(){
System.out.println("It is Famous for education institutions");
}
  }
class Gulmarg extends Hillstations {
  void location(){
System.out.println("Gulmarg is in J&K");
}
void famousfor(){
System.out.println("It is Famous for skiing");
}
  }
class main{
  public static void main(String args[]){
    Hillstations A = new Hillstations();
    Hillstations M = new Manali();

    Hillstations Mu = new Mussoorie();

    Hillstations G = new Gulmarg();

    A.location();
A.famousfor();

M.location();
M.famousfor();

Mu.location();
Mu.famousfor();

G.location();
G.famousfor();
  }
}
```

Output:

Location is: Famous for: Manali is in Himachal Pradesh It is Famous for Hadimba Temple and adventure sports Mussoorie is in Uttarakhand It is Famous for education institutions Gulmarg is in J&K It is Famous for skiing

Example of run-time polymorphism in java

We will create two classes Car and Innova, Innova class will extend the car class and will override its run() method.

```
class Car
{
void run()
{
System.out.println(“ running”);
}
}
class innova extends Car
{
void run();
{
System.out.println(“ running fast at 120km”);
}
public static void main(String args[])
{
Car c = new innova();
c.run();
}
}
```

The output of the following program will be;

Running fast at 120 km.

Another example for run-time polymorphism in Java

Now, let us check if we can achieve runtime polymorphism via data members.

```
class car
{
int speedlimit = 125;
}
class innova extends car
{
int speedlimit = 135;
public static void main(String args[])
{
car obj = new innova();
System.out.println(obj.speedlimit);
}
```

The output of the following program will be :

125

This clearly implies we can’t achieve Runtime polymorphism via data members. In short, a method is overridden, not the data members.

Runtime polymorphism with multilevel inheritance

```
class grandfather
{
void swim()
{
System.out.println(“ Swimming”);
}
}
class father extends grandfather
{
void swim()
{
System.out.println(“ Swimming in river”);
}
}
class son extends father
{
void swim()
{
System.out.println(“ Swimming in pool”);
}
public static void main(String args[])
{
grandfather f1,f2,f3;
f1 =new grandfather();
f2 = new father();
f3 = new son();
f1.swim();
f2.swim();
f3.swim():
}
}
```

The output of the following program will be:

Swimming, *Swimming in river*, *Swimming in pool*

Another runtime polymorphism with multilevel inheritance example

```
class soundAnimal
{
public void Sound()
{
System.out.println("Different sounds of animal"); }
}
class buffalo extends soundAnimal
{
public void Sound()
{
System.out.println("The buffalo sound- gho,gho"); }
}
class snake extends soundAnimal
{
public void Sound()
{
System.out.println("The snake sound- his,his"); }
```

(continues on next page)

(continued from previous page)

```
}
class tiger extends soundAnimal
{
public void Sound()
{
System.out.println("The tiger sounds- roooo, rooo"); }
}
public class Animal Main
{
public static void main(String[] args)
{
soundAnimal Animal = new soundAnimal(); soundAnimal buffalo = new buffalo();
soundAnimal snake = new snake();
soundAnimal tiger = new tiger();
Animal.Sound();
buffalo.Sound();
snake.Sound();
tiger.Sound();
}
}
```

The output of the following program will be;

The buffalo sound- gho,gho

The snake sound- his,his

The tiger sound- roooo,roooo

We hope you got an idea about runtime and compile-time polymorphism.

29.4.5 Polymorphic Subtypes

Subtype basically means that a subtype can serve as another type's subtype, sounds a bit complicated?

Let's understand this with the help of an example:

Assuming we have to draw some arbitrary shapes, we can introduce a class named 'shape' with a draw() method. By overriding draw() with other subclasses such as circle, square, rectangle, trapezium, etc we will introduce an array of type 'shape' whose elements store references will refer to 'shape' subclass references. Next time, we will call draw(), all shapes instances draw () method will be called.

This Subtype polymorphism generally relies on upcasting and late binding. A casting where you cast up the inheritance hierarchy from subtype to a supertype is termed upcasting.

To call non-final instance methods we use late binding. In short, a compiler should not perform any argument checks, type checks, method calls, etc, and leave everything on the runtime.

29.4.6 What is Polymorphism in Programming?

Polymorphism in programming is defined usage of a single symbol to represent multiple different types.

29.4.7 What is Polymorphism Variables?

A **polymorphic variable** is defined as a variable that can hold values of different types during the course of execution.

29.4.8 Why use Polymorphism in Java?

Polymorphism in Java makes it possible to write a method that can correctly process lots of different types of functionalities that have the same name. We can also gain consistency in our code by using polymorphism.

Advantages of Polymorphism in Java

1. It provides reusability to the code. The classes that are written, tested and implemented can be reused multiple times. Furthermore, it saves a lot of time for the coder. Also, the one can change the code without affecting the original code.
2. A single variable can be used to store multiple data values. The value of a variable you inherit from the superclass into the subclass can be changed without changing that variable's value in the superclass; or any other subclasses.
3. With lesser lines of code, it becomes easier for the programmer to debug the code.

Characteristics of Polymorphism

Polymorphism has many other characteristics other than Method Overloading and Method Overriding. They include:

- Coercion
- Internal Operator Overloading
- Polymorphic Variables or Parameters

1. Coercion

Coercion deals with implicitly converting one type of object into a new object of a different kind. Also, this is done automatically to prevent type errors in the code.

Programming languages such as C, java, etc support the conversion of value from one data type to another data type. Data type conversions are of two types, i.e., implicit and explicit.

Implicit type conversion is automatically done in the program and this type of conversion is also termed coercion.

For example, if an operand is an integer and another one is in float, the compiler implicitly converts the integer into float value to avoid type error.

Example :

```
class coercion {

  public static void main(String[] args) {
    Double area = 3.14*5*7;
```

(continues on next page)

(continued from previous page)

```
System.out.println(area);
String s = "happy";
int x=5;
String word = s+x;
System.out.println(word);

  }
}
```

Output:

109.9 happy5

2. Internal Operator Overloading

In Operator Overloading, an operator or symbol behaves in more ways than one depending upon the input context or the type of operands. It is a characteristic of static polymorphism. Although Java does not support user-defined operator overloading like C++, where the user can define how an operator works for different operands, there are few instances where Java internally overloads operators.

Operator overloading is the concept of using the operator as per your choice. Therefore, an operator symbol or method name can be used as a 'user-defined' type as per the requirements.

For example, '+' can be used to perform the addition of numbers (same data type) or for concatenation of two or more strings.

In the case of +, can be used for addition and also for concatenation.

For example:

```
class coercion {

  public static void main(String[] args) {

String s = "happy";
String s1 = "world";
int x=5;
int y=10;

System.out.println(s+s1);
System.out.println(x+y);

  }
}
```

Output :

happyworld 15

Similarly, operators like**! &, and |** are also in the overload position for logical and bitwise operations. In both of these cases, the type of argument will decide how the operator will interpret.

3. Polymorphic Variables or Parameters

In Java, the object or instance variables represent the polymorphic variables. This is because any object variables of a class can have an IS-A relationship with their own classes and subclasses.

The Polymorphic Variable is a variable that can hold values of different types during the time of execution.

Parametric polymorphism specifies that while class declaration, a field name can associate with different types, and a method name can associate with different parameters and return types.

For example:

```java
class Shape
{
public void display()
{
System.out.println("A Shape.");
}
}
class Triangle extends Shape
{
public void display()
{
System.out.println("I am a triangle.");
}
}
class Main{
public static void main(String[] args)
{
Shape obj;
obj = new Shape();
obj.display();
obj = new Triangle();
obj.display();
}
}
```

Output:

A Shape. I am a triangle.

Here, the obj object is a polymorphic variable. This is because the superclass's same object refers to the parent class (Shape) and the child class (Triangle).

29.4.9 Problems with Polymorphism

With lots of advantages, there are also a few disadvantages of polymorphism.

- Polymorphism is quite challenging while implementation.
- It tends to reduce the readability of the code.
- It raises some serious performance issues in real-time as well.

Type Identification During Downcasting

Downcasting is termed as casting to a child type or casting a common type to an individual type.

So, we use downcasting whenever we need to access or understand the behaviour of the subtypes.

Example,

This is a hierarchical example

Food> Vegetable> Ladyfinger, Tomato

Here, tomato and ladyfinger are two subclasses.

In downcasting, we narrow the type of objects, which means we are converting common type to individual type.

Vegetable vegetable = new Tomato();

Tomato castedTomato = (Tomato) vegetable;

Here we are casting common type to an individual type, superclass to subclass which is not possible directly in java.

We explicitly tell the compiler what the runtime type of the object is.

Fragile base class problem

Fragile base class problem is nothing but a fundamental architectural problem.

Sometimes the improper design of a parent class can lead a subclass of a superclass to use superclass in some unpredicted ways.

The fragility of inheritance will lead to broken codes even when all the criteria is met.

This architectural problem is termed as a fragile base class problem in object-oriented programming systems and language.

Basically, the reason for the fragile base problem is that the developer of the base class has no idea of the subclass design. There is no solution yet for this problem.

29.4.10 Conclusion

We hope you must have got a basic idea of polymorphism in Java and how we use it as well as problems related to them.

End Of Topic

29.5 Unable To Run .JAR Files in Windows 10 / 11 ? Here's the Solution

May 28, 2022 By **Madhuparna**

https://thegeekpage.com/unable-to-run-jar-files-in-windows-10-heres-the-solution/#comment-161381

A JAR file is based on a Java archive file format that may include a Java program inside it. Although you can use a zip file extracting software like the 7zip to extract the .JAR files, it won't allow you to run a compete Java app based on .JAR.

So, if you are facing an issue while opening a .jar file, you can try the below methods.

Table of Contents

[TOC]

29.5.1 Method 1 – Download and run jarfix

Jatfix is a free utility which repairs jar files and let you run files which you are unable to run.

\1. Just go to this link and download jarfix.

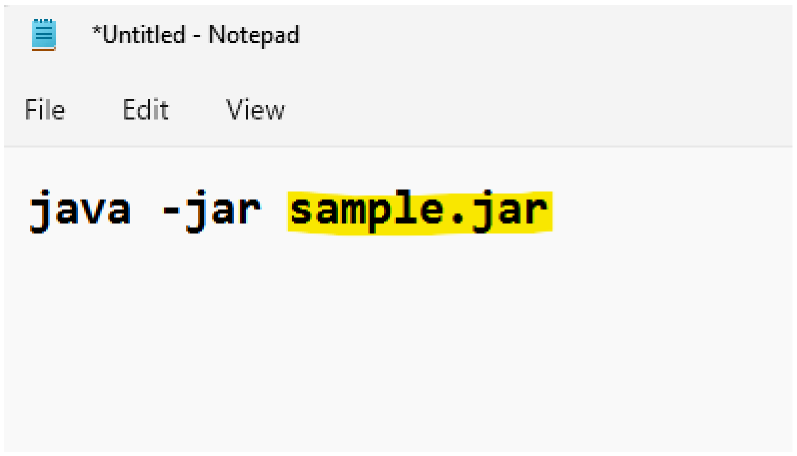
2.Now, Install it after downloading .

Now, try again.

29.5.2 Method 2: By Creating a .bat File

Step 1: Open **notepad** and type the below text on it:

```
java -jar sample.jar
```



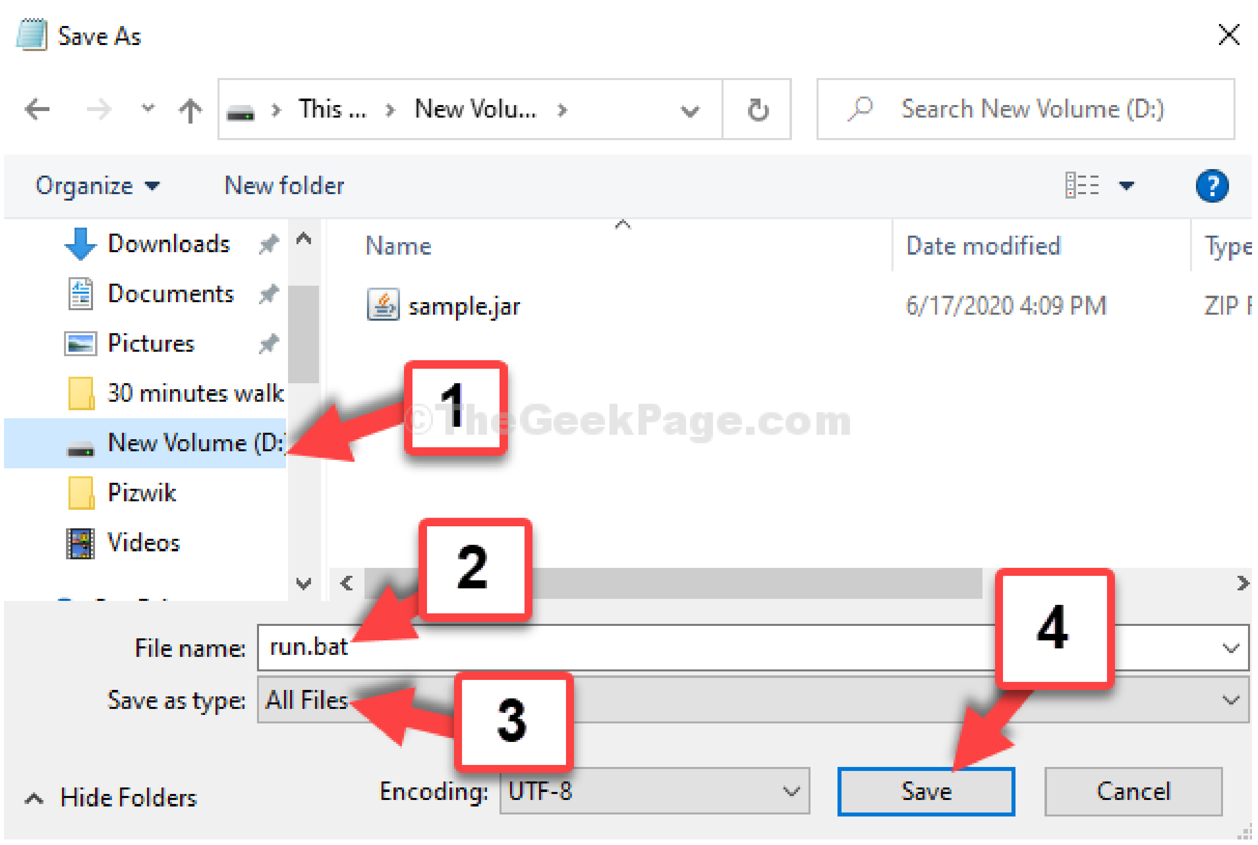
```
java -jar sample.jar
```

***Note –** Replace the highlighted portion with whatever is the name of your **.jar** file.

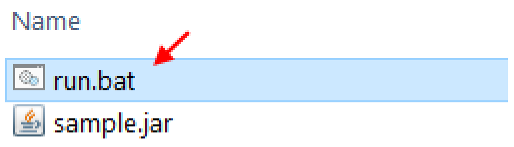
Step 2: Save the file with any name that suits you, in the same location where your **.jar** file is saved, followed by **.bat** extension.

For instance, here we named the file as **run.bat** and changed the **Save as type** to **All Files.**

Press the **Save** button.



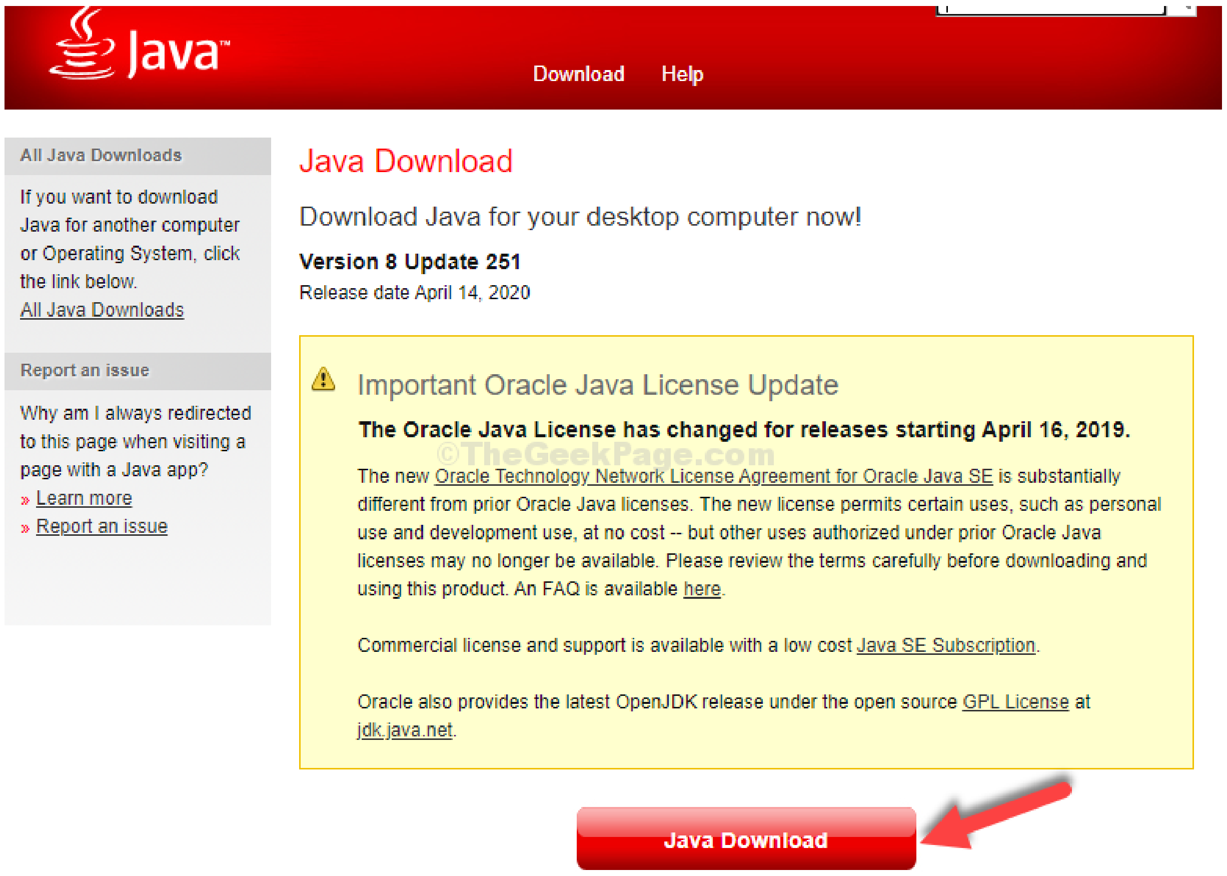
Now, double-click on the **run.bat** file and your **.jar** file will open smoothly.



29.5.3 Method 3: By Downloading Java

Step 1: Go to Java.com and download Java from the link

Step 2: On the website download page, click on the **Download** button in red to download the **Java** setup file.



All Java Downloads

If you want to download Java for another computer or Operating System, click the link below.
All Java Downloads

Report an issue

Why am I always redirected to this page when visiting a page with a Java app?
» Learn more
» Report an issue

Java Download

Download Java for your desktop computer now!

Version 8 Update 251
Release date April 14, 2020

Important Oracle Java License Update

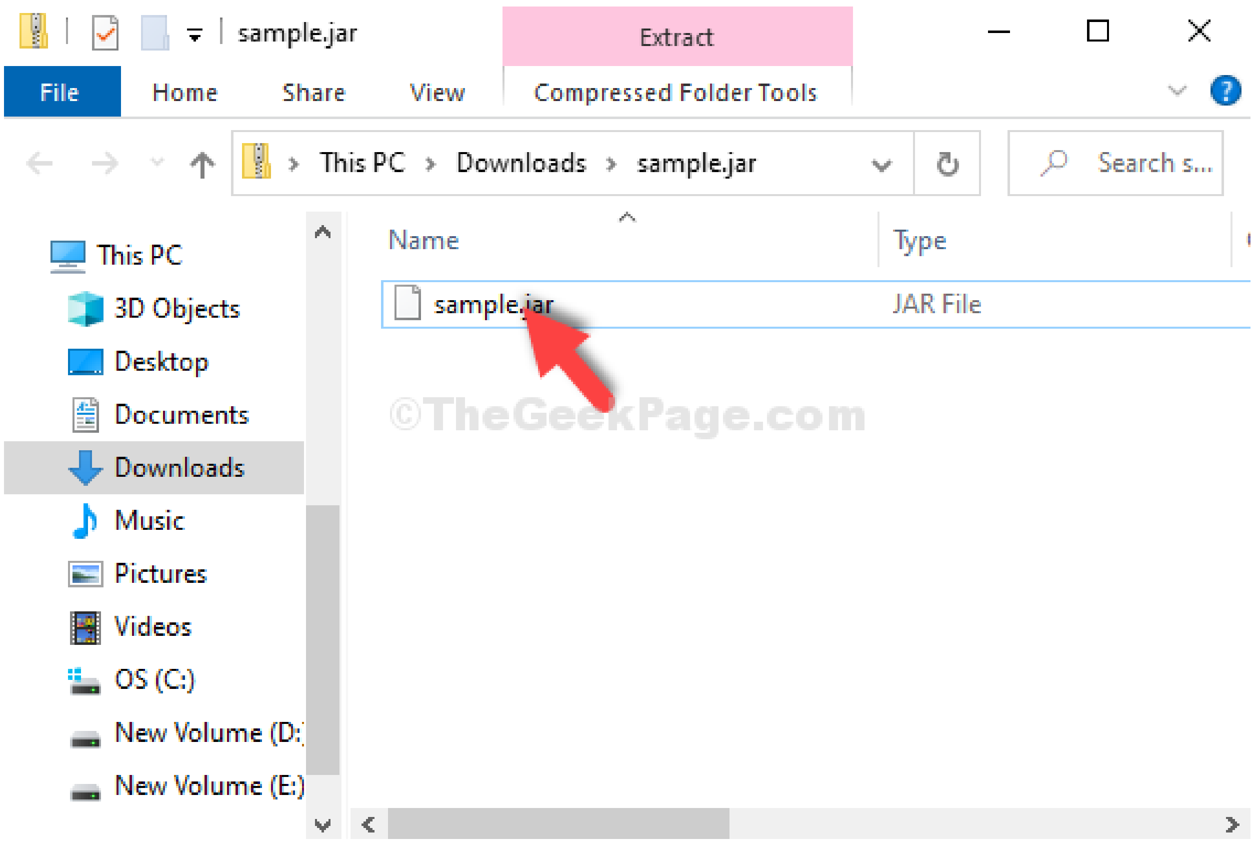
The Oracle Java License has changed for releases starting April 16, 2019.

The new Oracle Technology Network License Agreement for Oracle Java SE is substantially different from prior Oracle Java licenses. The new license permits certain uses, such as personal use and development use, at no cost -- but other uses authorized under prior Oracle Java licenses may no longer be available. Please review the terms carefully before downloading and using this product. An FAQ is available here.

Commercial license and support is available with a low cost Java SE Subscription.

Oracle also provides the latest OpenJDK release under the open source GPL License at jdk.java.net.

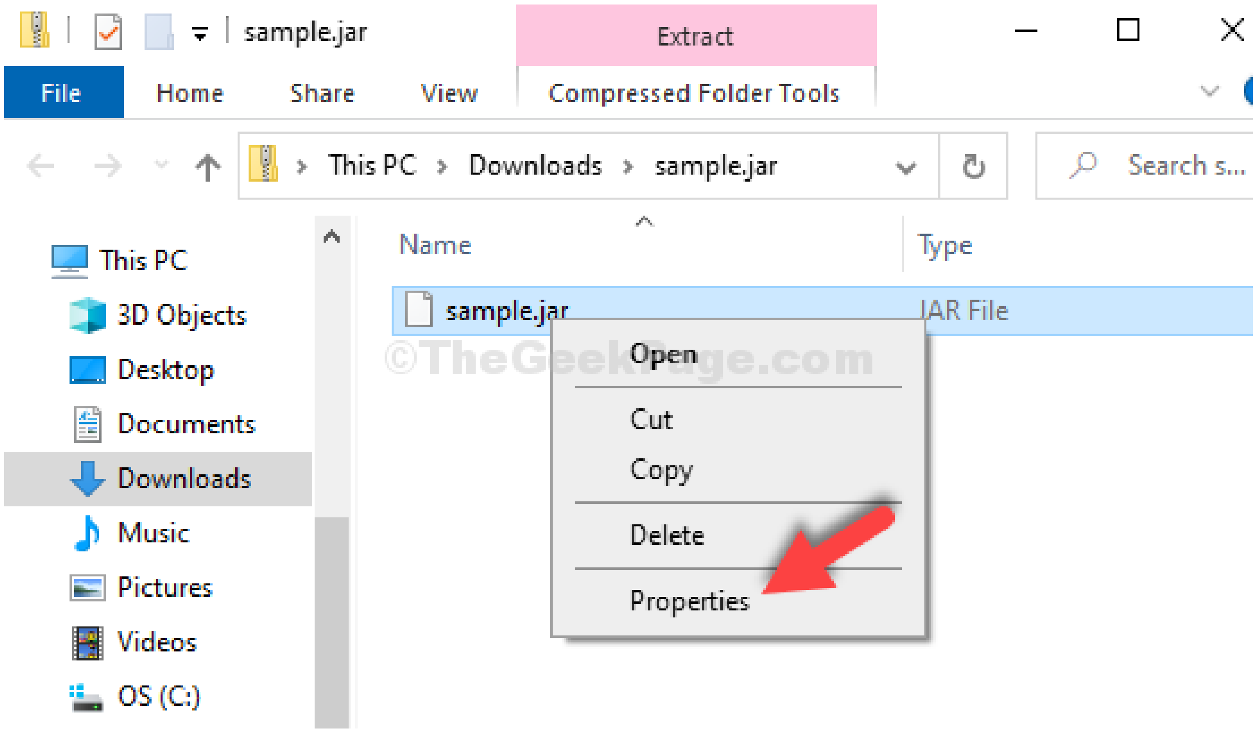
Step 3: Once the setup is complete, go to the location where the **.jar** file is saved and double-click on it to run the file.



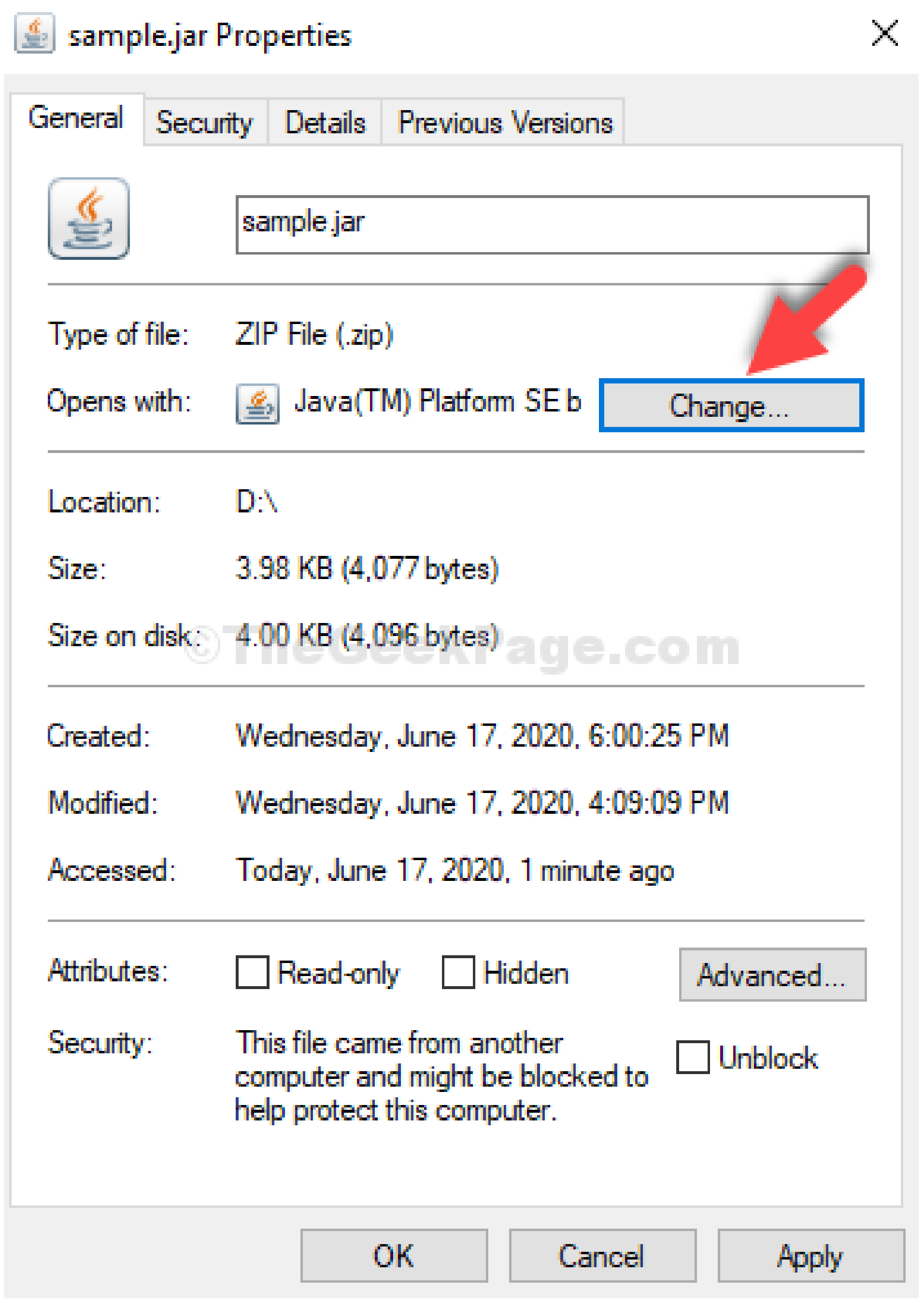
If it doesn't open up try the 2nd method.

29.5.4 Method 4: Through Properties

Step 1: Simply right-click on it and select **Properties** from the context menu.



Step 2: In the **Properties** window, click on **Change**.


sample.jar Properties

sample.jar

Type of file: ZIP File (.zip)

Opens with: Java(TM) Platform SE b

Location: D:\

Size: 3.98 KB (4,077 bytes)

Size on disk: 4.00 KB (4,096 bytes)

Created: Wednesday, June 17, 2020, 6:00:25 PM

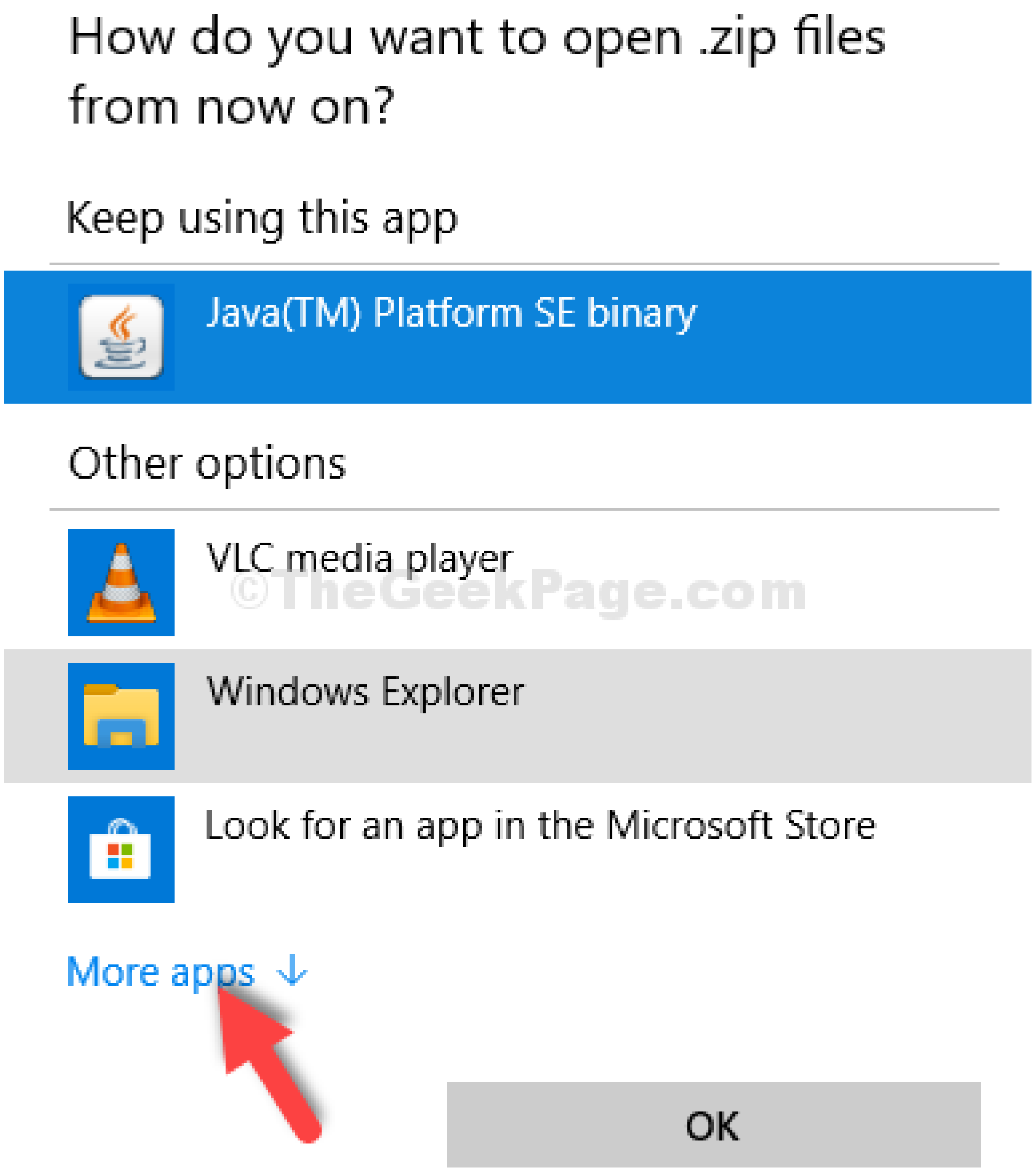
Modified: Wednesday, June 17, 2020, 4:09:09 PM

Accessed: Today, June 17, 2020, 1 minute ago

Attributes: ☐ Read-only ☐ Hidden

Security: This file came from another computer and might be blocked to help protect this computer. ☐ Unblock

Step 3: In the next window, click on **More apps.**



How do you want to open .zip files from now on?

Keep using this app

Java(TM) Platform SE binary

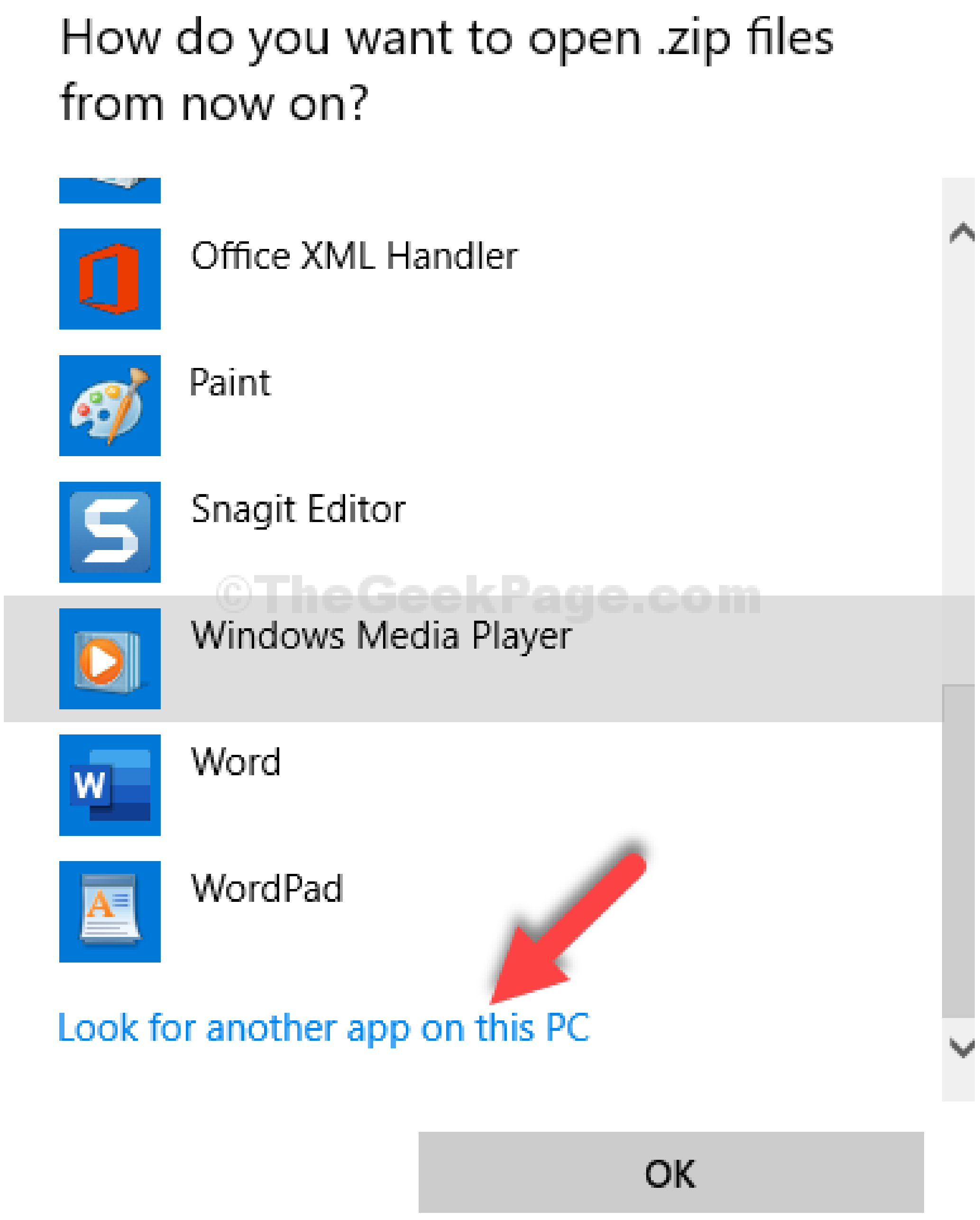
Other options

VLC media player

Windows Explorer

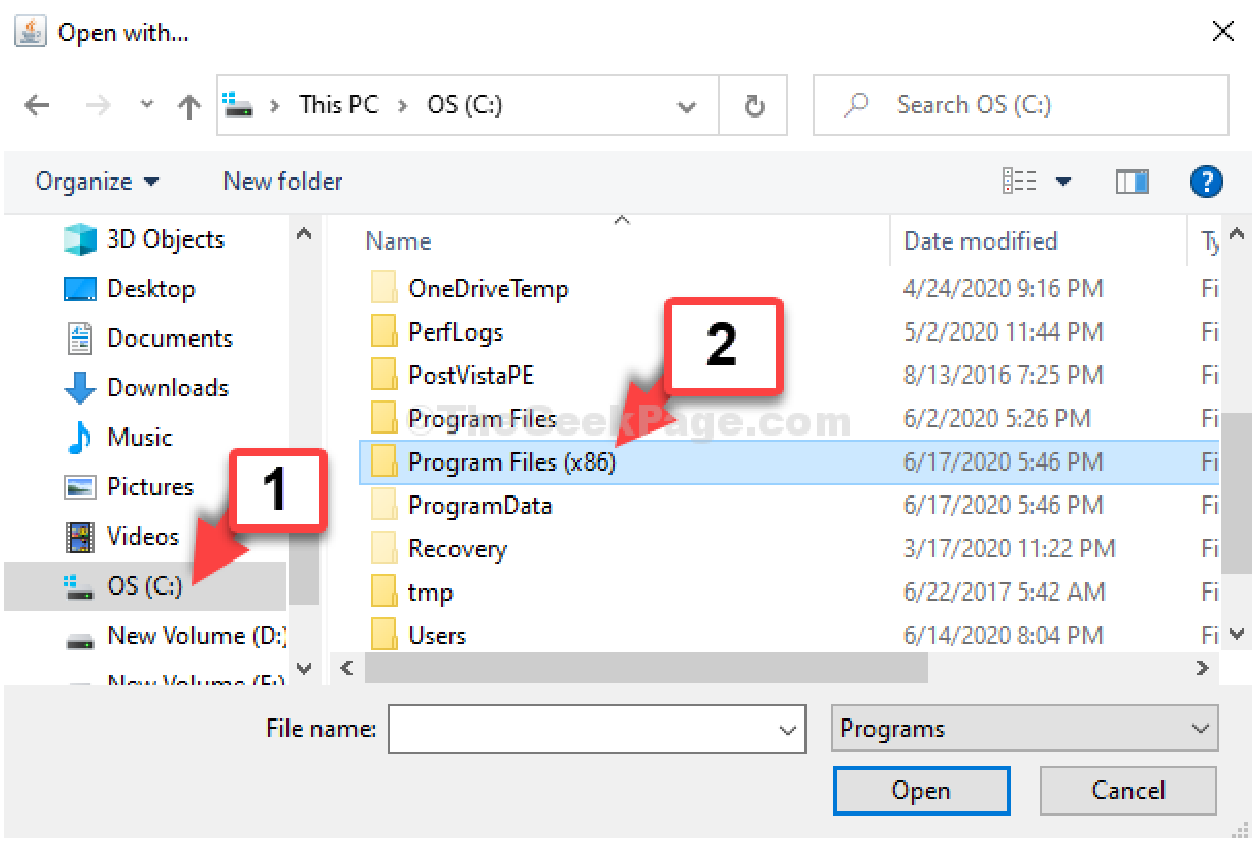
Look for an app in the Microsoft Store

Step 4: Scroll down and click on **Look for another app** on this PC.


How do you want to open .zip files from now on?

Step 5: It takes you to the **File Explorer**, click on the **C drive** shortcut and follow the below path step by step:

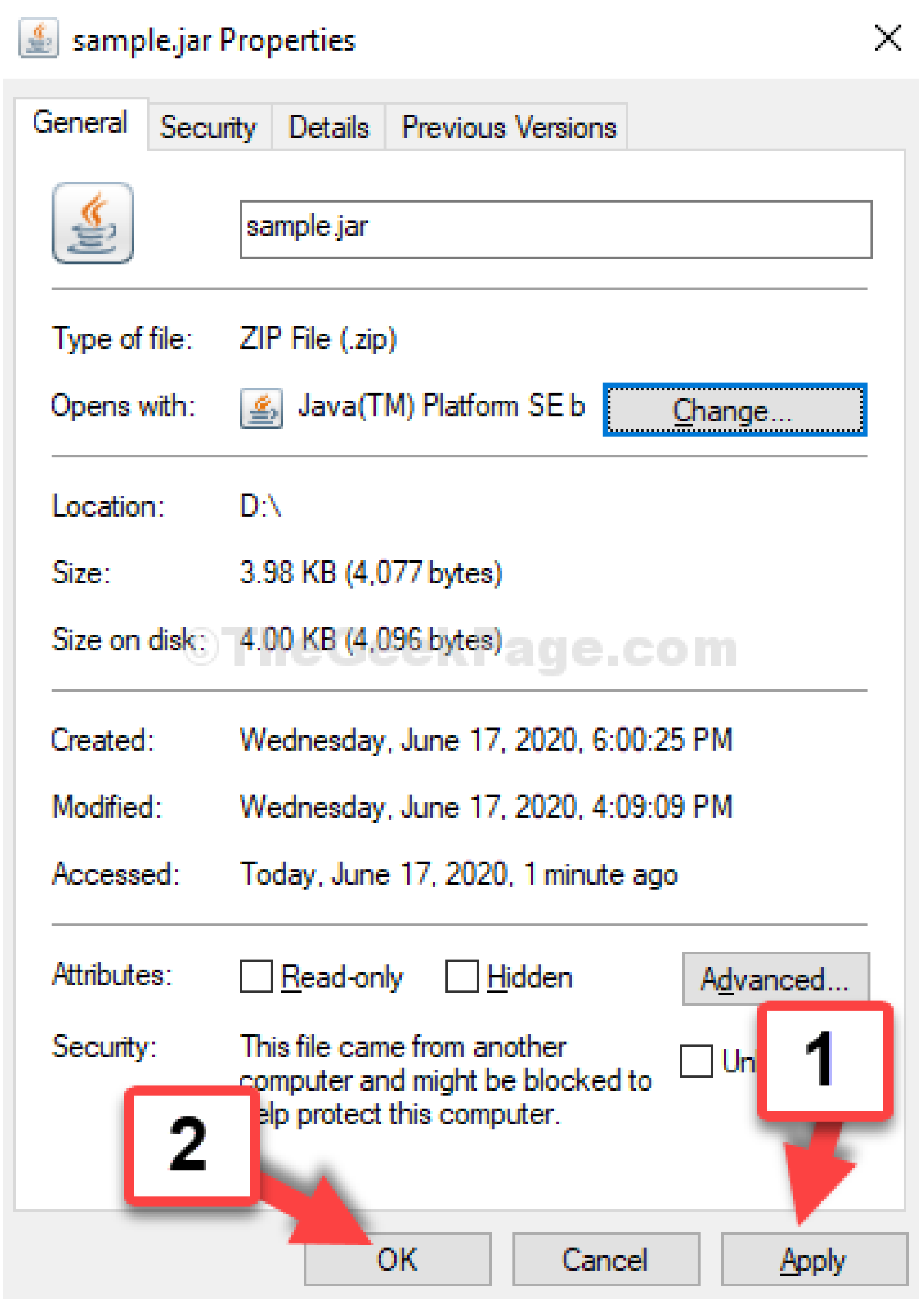
- Double-click on **Program Files (x86)**
- Double-click on **Java**
- Double-click on **jre1.8.0_251**
- Double-click on **bin**
- Select **javaw**



File name:

Name	Date modified	Ty
OneDriveTemp	4/24/2020 9:16 PM	Fi
PerfLogs	5/2/2020 11:44 PM	Fi
PostVistaPE	8/13/2016 7:25 PM	Fi
Program Files	6/2/2020 5:26 PM	Fi
Program Files (x86)	6/17/2020 5:46 PM	Fi
ProgramData	6/17/2020 5:46 PM	Fi
Recovery	3/17/2020 11:22 PM	Fi
tmp	6/22/2017 5:42 AM	Fi
Users	6/14/2020 8:04 PM	Fi

Step 6: As you double-click on **javaw**, it takes you back to the **Properties** window. Click on **Apply** and then **OK**.


sample.jar Properties

General | Security | Details | Previous Versions

sample.jar

Type of file: ZIP File (.zip)

Opens with: Java(TM) Platform SE b

Location: D:\

Size: 3.98 KB (4,077 bytes)

Size on disk: 4.00 KB (4,096 bytes)

Created: Wednesday, June 17, 2020, 6:00:25 PM

Modified: Wednesday, June 17, 2020, 4:09:09 PM

Accessed: Today, June 17, 2020, 1 minute ago

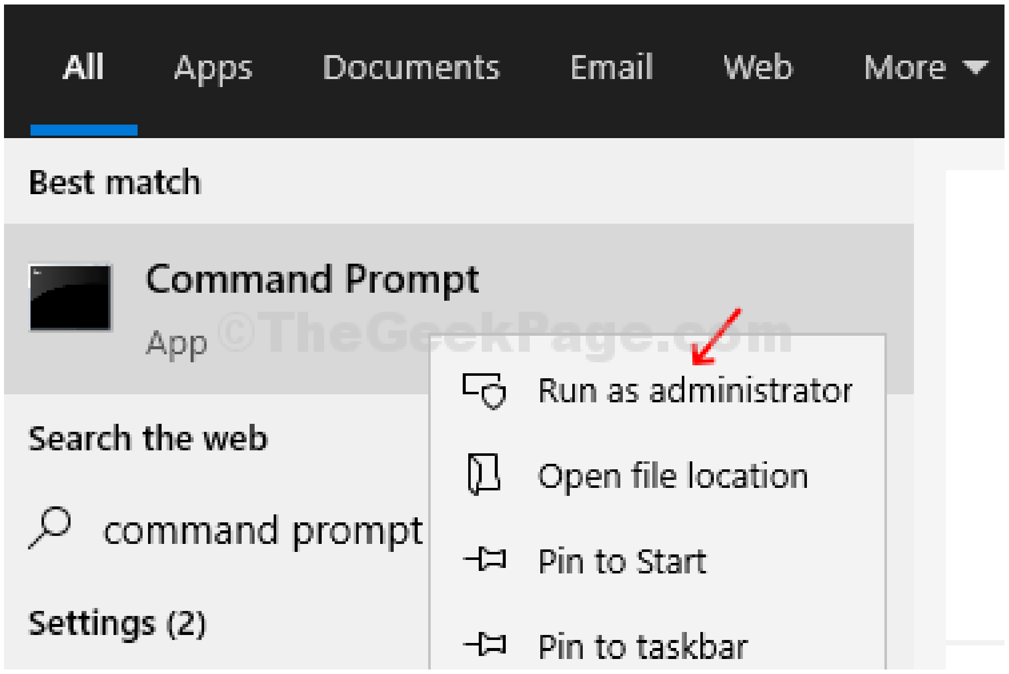
Attributes: Read-only, Hidden

Security: This file came from another computer and might be blocked to help protect this computer.

Your .jar file should open now. If it still doesn't open, try the 3rd method.

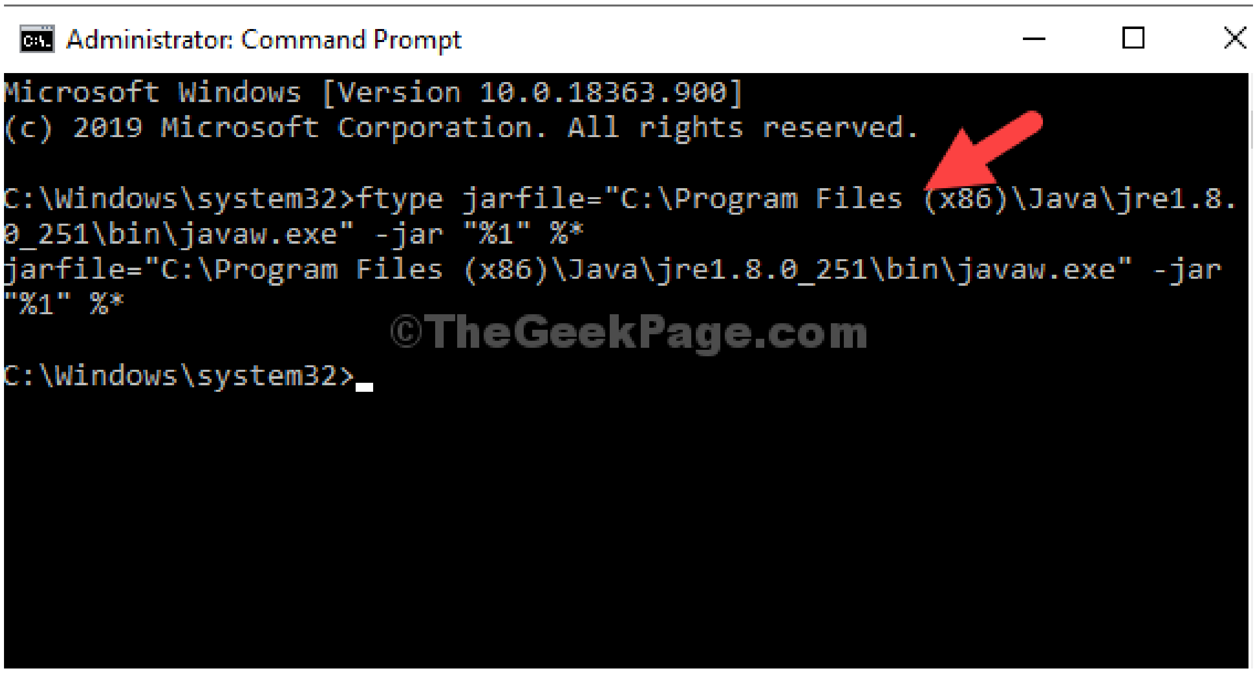
29.5.5 Method 5: Using Command Prompt

Step 1: Click on the **Start** button on your desktop and type **Command Prompt** in the search field. Right-click on the result and select **Run as administrator**.



Step 2: In the **Command Prompt** window, type the below command in the below format and hit **Enter**:

```
ftype jarfile="C:\Program Files (x86)\Java\jre1.8.0_251\bin\javaw.exe" -jar "%1" %*
```



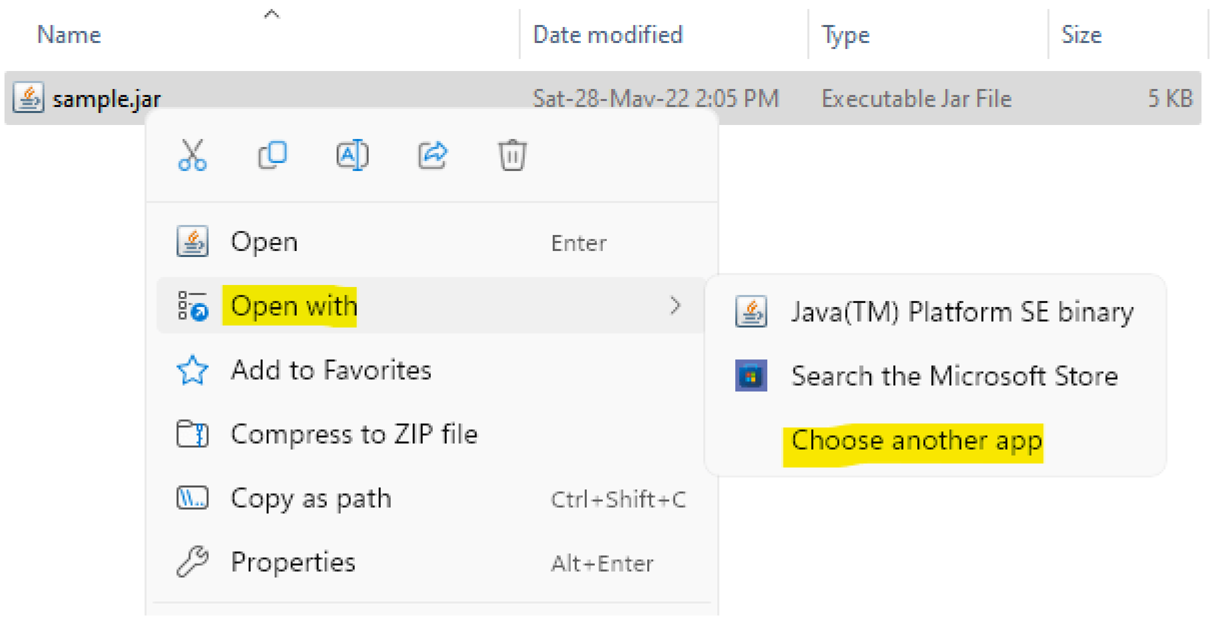
```
Microsoft Windows [Version 10.0.18363.900]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Windows\system32>ftype jarfile="C:\Program Files (x86)\Java\jre1.8.
0_251\bin\javaw.exe" -jar "%1" %*
jarfile="C:\Program Files (x86)\Java\jre1.8.0_251\bin\javaw.exe" -jar
"%1" %*

C:\Windows\system32>
```

***Note –** The path entered is the same path for **javaw** as illustrated in **Method 2**.

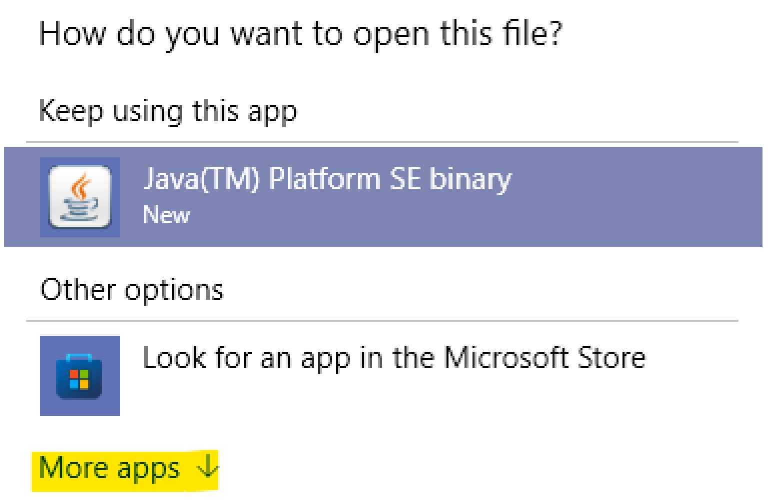
This should resolve your issue and you can open the .jar file. However, if problem persists you can try the 4th method.

29.5.6 Method 6: Set file association

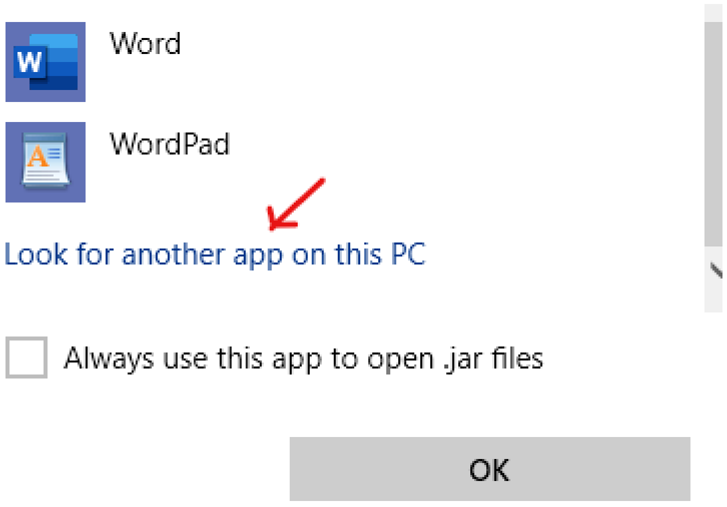
1 -Right click on the jar file and click on **Open with** and then click on **choose another app**.



2 – Click on **More apps**

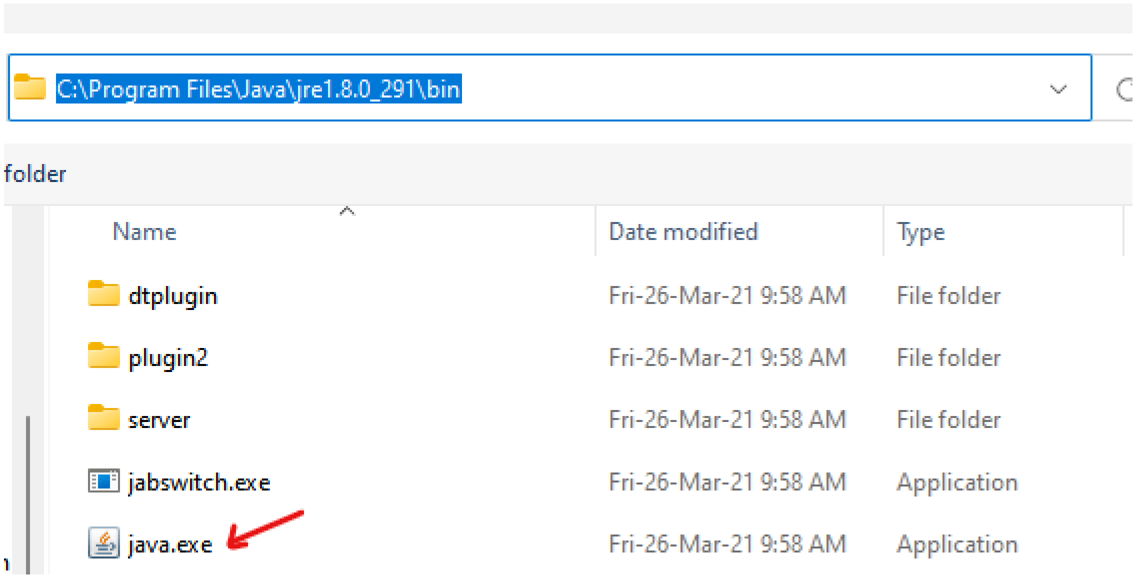

How do you want to open this file?
Keep using this app
Java(TM) Platform SE binary
New
Other options
Look for an app in the Microsoft Store
More apps ↓

3 – Click on **Look for another app on this PC**


Word
WordPad
Look for another app on this PC
Always use this app to open .jar files
OK

4 – Now, go to

- **C:**
- **Program Files**
- **Java**
- **jre-Version-Number**
- **bin**


C:\Program Files\Java\jre1.8.0_291\bin

Name	Date modified	Type
dtplugin	Fri-26-Mar-21 9:58 AM	File folder
plugin2	Fri-26-Mar-21 9:58 AM	File folder
server	Fri-26-Mar-21 9:58 AM	File folder
jabswitch.exe	Fri-26-Mar-21 9:58 AM	Application
java.exe	Fri-26-Mar-21 9:58 AM	Application

5 – Select java.exe and click **OK**.

If all of the above methods fail, search for **jarfix** program on **Google** and download it. This will instantly fix the issue.

29.6 End Of Section

Java Language

End Of Section

Part II

Data Structures

CHAPTER

THIRTY

DATA STRUCTURES AND ALGORITHMS

30.1 Textbooks and Online Sources

Data Structures and Algorithms in Java Second Edition Robert Lafore

Data Structures and Algorithms in Java 6th Edition Michael T. Goodrich

Algorithms (4th Edition) 4th Edition by Robert Sedgewick and Kevin Wayne

Open Data Structures |

End Of Topic

CHAPTER

THIRTYONE

ARRAY TECHNIQUES

End of Topic

31.1 Array Warp Around

31.1.1 Key Ideas

- Place data in an array.
  - If the data element is "full/occupied", look to the right for the next empty element and wrap around to the beginning element if at the end.

31.1.2 Lecture Code

```
/*
 * DS132SU_WrapAround
 *
 * Programmer: Jim Goudy
 * Project: Wrap Around Array
 This shows how to wrap around in an around.
 Meaning, the program looks to see if an array
 element is open. If it is not, then look to
 the right. Continue looking to the right,
 till the next available element is open.

 *
 */


import java.util.Random;
import java.util.Scanner;

public class DS_WrapAround {

    static int ArrayLength = 8;
```

(continues on next page)

(continued from previous page)

```
    static String[] myArray = new String[ArrayLength];
    static int maxVal = ArrayLength;

    static Scanner myScan = new Scanner(System.in);
        static Random RNG = new Random();

    static int myRNG() {
        // Generate a random number
        // within the number of array elements
        return RNG.nextInt(maxVal);
    }

    static void printArray() {
        // print the array

        for (int c = 0; c < myArray.length; c++) {
            if (myArray[c] == null) {
                System.out.print(" - |");
            } else {
                System.out.print(myArray[c] + " |");
            }
        }
    }

    public static void main(String[] args) {

        String run = "y";
        int aIndex = -1;
        boolean check = true;
        int boxCntr = 0;

        String quit = "y";

        // assign values to the array - go to right if occupied
        // this loops continues till the array is full
        while (run.equals("y")) {
            aIndex = myRNG();
            System.out.print("\nComputer chooses " + aIndex + "\n");

            // fill the array element
            while (check) {
                if (myArray[aIndex] == null) {
                    //array index(box) is empty
                    myArray[aIndex] = " X";

                    // this variable keeps track of
                    // the total number of elments that
                    // are occupied/filled
                    boxCntr++;

                    // sets variable to exit
                    // the inner while loop
                    check = false;

                    // check if all the array elements are filled
                    if (boxCntr == myArray.length) {
```

(continues on next page)

(continued from previous page)

```
                        run = "n";
                    }
                } else {
                    // array index (box) is not empty
                    aIndex++;
                    System.out.println("move right " + aIndex);

                    // if the index is at the end of the array
                    // wrap around to the first element 0
                    if (aIndex == myArray.length) {
                        aIndex = 0;
                        System.out.println("move right " + aIndex);
                    }
                }
            }

            check = true;
            printArray();
        }

    }
}
/*
Note: Since the computer use a random generator to pick numbers - individual results␣
↪will vary
Example Output:
Computer chooses 5
 - | - | - | - | - | X | - | - |
Computer chooses 6
 - | - | - | - | - | X | X | - |
Computer chooses 7
 - | - | - | - | - | X | X | X |
Computer chooses 7
move right 8
move right 0
 X | - | - | - | - | X | X | X |
Computer chooses 6
move right 7
move right 8
move right 0
move right 1
 X | X | - | - | - | X | X | X |
Computer chooses 3
 X | X | - | X | - | X | X | X |
Computer chooses 0
move right 1
move right 2
 X | X | X | X | - | X | X | X |
Computer chooses 5
move right 6
move right 7
move right 8
move right 0
move right 1
move right 2
move right 3
```

(continues on next page)

(continued from previous page)

```
move right 4
 X | X | X | X | X | X | X | X |
------------------------------------------------------------------------
*/
```

End Of Topic

31.2 Array Add and Delete Data

31.2.1 Key Ideas

- In working with arrays, keeping data continuous is a good practice the majority of the time. If data in an element is deleted, the data in the elements to the right should be shifted left to remove the empty space.
- Also, there may be times when data has to be inserted into an array that has continuous data elements. In this case, data is shifted to the right to create a space where the new data can be inserted.

Tip: In many cases, arrays are usually set to have more elements than is needed. Therefore, it is important to create a variable that will alway track the number of items in the array.

31.2.2 Lecture Code

```
/*
 *
 * Project: Add Delete In an Array
 * Programmer: J Goudy

 */


public class DS_ArraysAddDelete {

    static String[] arrString;
    static int arrStrDataCount = 0;

    static void loadStringArray() {
    // this is a helper function to setup our example array

        String[] names = {"Adam", "Bobby", "Howard", "Mary", "Zuzu"};

        //load names
        for (int c = 0; c < names.length; c++) {
            arrString[c] = names[c];
        }

                //set our number of data items
        arrStrDataCount = names.length;

```

(continues on next page)

(continued from previous page)

```
        // print number of items
        System.out.println("DataCount = " + arrStrDataCount);

        printArray(arrString, arrStrDataCount);

    }

    static void printArray(String[] theArray, int dataCount) {
    // This function prints the array.
    // Note that an array is being passed to it.

        // for spacing
        System.out.println();

        // iterate through the array and print the data
        for (int i = 0; i < dataCount; i++) {
            System.out.print(theArray[i] + " ");
        }

        System.out.println("\n------------\n");

    }

    static void insertStringByPos(int pos, String aName) {
        // ---------- Do some checks ---------------------

        // check if position is withing array bounds
        if (pos > arrStrDataCount) {
            System.out.println("Error out array bounds");

            // exit the function
            return;
        }

        // check if the array is full
        if (arrStrDataCount >= arrString.length) {
            System.out.println("Array is full");
            return;
        }

        // -------------- Insert Code --------------------------
        // shift to right
        // note that the loop is starting at the end and
        // working backwards to the insert spot (pos)
        for (int i = arrStrDataCount; i > pos; i--) {

            arrString[i] = arrString[i - 1];
        }

        // insert the new name
        arrString[pos] = aName;

        // increment our data Count
        arrStrDataCount++;

        // print the array to show data was inserted
```

(continues on next page)

(continued from previous page)

```java
        printArray(arrString, arrStrDataCount);

    }

    static void deleteByPos(int pos) {
        // check if there is contents
        if (arrStrDataCount <= 0) {
            System.out.println("Array is empty\n");
            return;
        }

        // check if the position to delete is out of bounds
        if (pos >= arrStrDataCount) {
            System.out.println("Pos is out of bounds\n");
            return;
        }

        // -----------  Delete Code ---------------------
        // shift loop
        // note that the loop starts at the element location
        // that is being deleted
        for (int i = pos; i < arrStrDataCount; i++) {
            arrString[i] = arrString[i + 1];
        }

        // decrease the item count by 1
        arrStrDataCount--;

        printArray(arrString, arrStrDataCount);

    }

    public static void main(String[] args) {

        // instantiate the data array
        arrString = new String[8];

        try {

            // setup the demo array
            loadStringArray();

            // insert "Bubba" in the third positon of the array
            insertStringByPos(2, "Bubba");

            // delete the data in the second
            // element/position of the array
            deleteByPos(1);

        } catch (Exception e) {
            System.out.println(e.getMessage());
        }

        System.out.println("\n\nbye\n");
    }
}
```

End Of Topic

CHAPTER

THIRTYTWO

BIG O NOTATION

32.1 Key Ideas

- Big O Notation

Definition

Big O notation is used in Computer Science to describe the performance or complexity of an algorithm. Big O specifically describes the worst-case scenario and can be used to describe the execution time required or the space used (e.g. in memory or on disk) by an algorithm. - Rob Bell

32.2 Reading

ON_Visualizing Big O Notation

32.3 Videos

https://youtu.be/v4cd1O4zkGw

https://youtu.be/Q_1M2JaijjQ

End Of Topic

32.4 Visualizing Big O notation

From https://mellowd.co.uk/ccie/?p=6122

Author: Darren O'Connor

I'm currently learning as much computer science as I can on the side. I've come across Big O notation a few times already, and while I understand it, I'm much more of a visual guy.

It's rather easy to use Python and matplotlib to graph out how a function's execution time grows as the size of the input grows. The important thing to note is not total execution time, but rather how the runtime of that function grows in relation to the input size. This can be plotted onto a graph which should give us a nice representation of Big O notations.

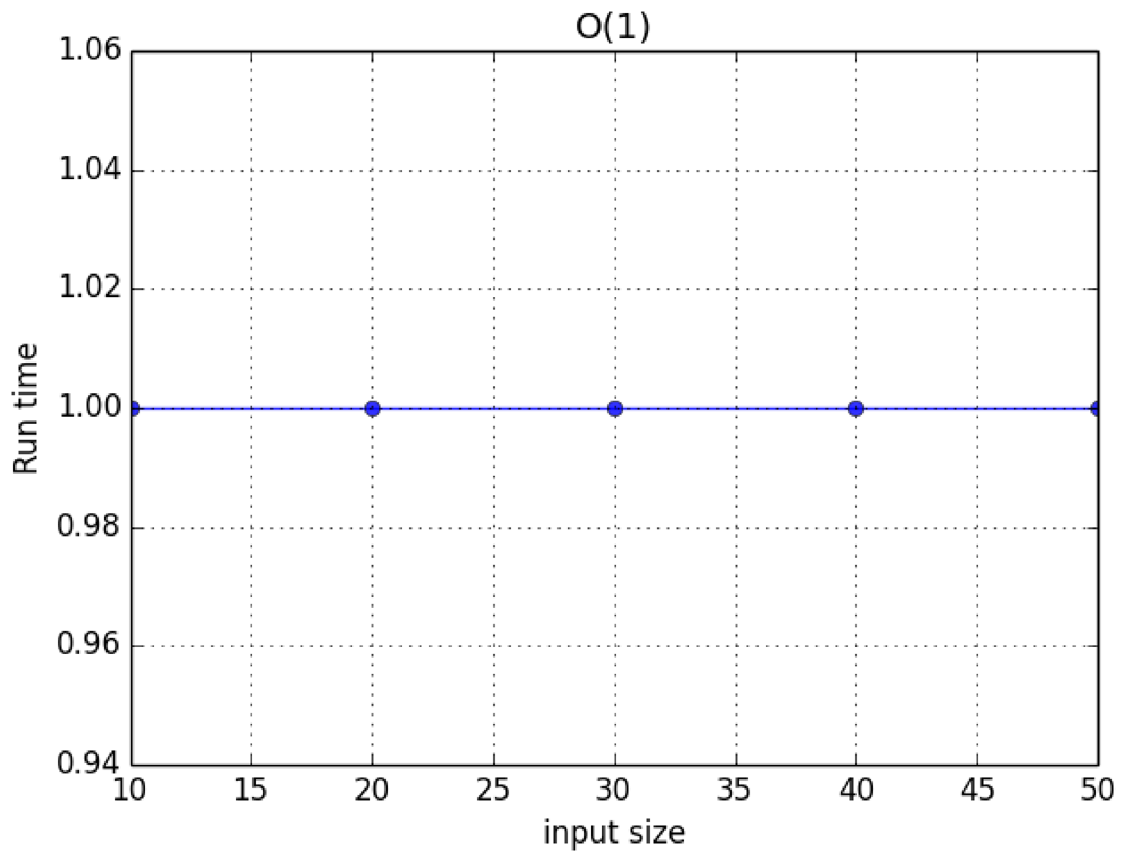
Note too that Big O notations always show the worst case. For this reason, I'll ensure to use values which the function will have to do the most work for.

32.4.1 O(1)

O(1) means constant time. No matter what size the input, the runtime will always be the same. A simple example is finding the middle number in a list. I'll ensure that all code return the amount of time a command was run in the function. This may make the code look just a bit bloated, but for a good reason.To find the center of a list we simply divide the length of the list in two, and return that number. It does not matter if a list has 10 elements or 100 elements, the same amount of steps is performed:

```
def O1(input):
    count = 0
    result = input[len(input) / 2]
    count += 1
    return count
```

I have created 5 lists. The first is length 10, the second the length 20, and so on. I'll get the returned values and plot them.



O(1) plot

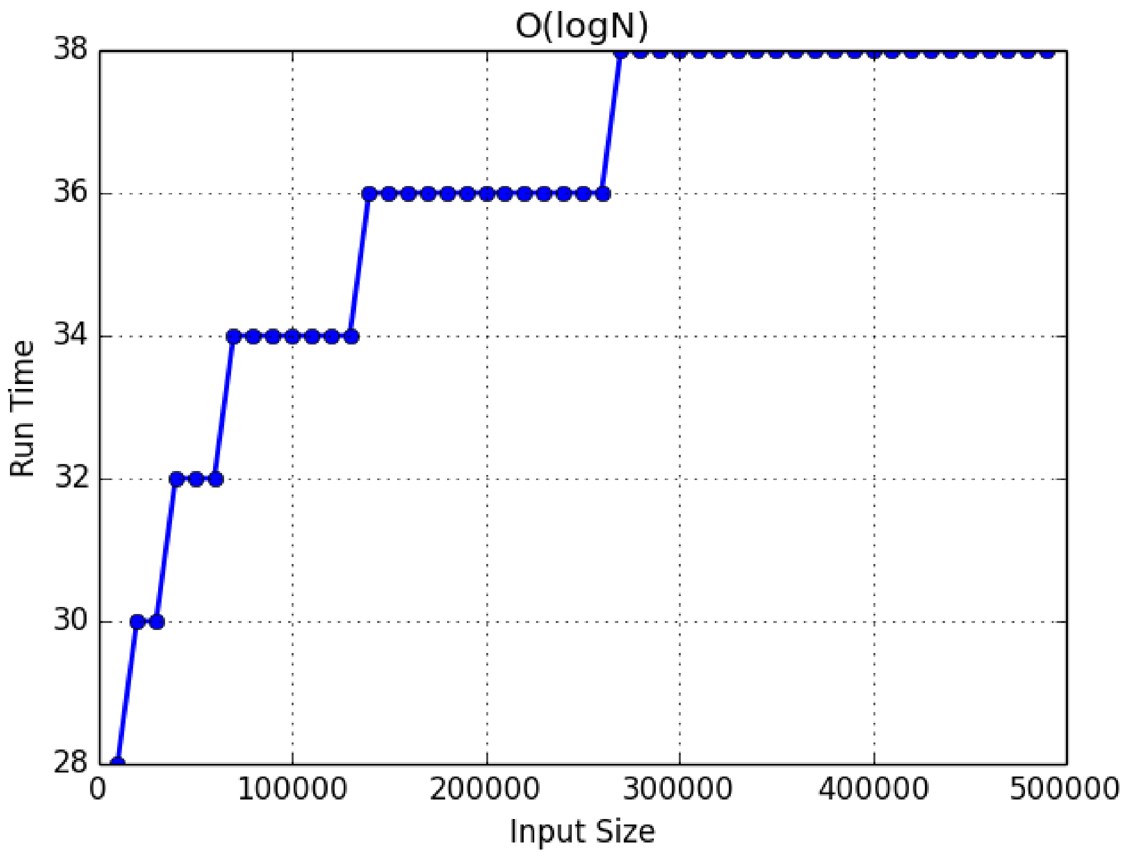
As can be seen, it doesn't matter the size of the input. It will always run at the same constant time.

32.4.2 O(logN)

O(logN) increases as the input size goes up. However, it goes up as a log of the input size. This means that you can exponentially increase your input size, without linearly increasing the processing time to match.

```python
def OlogN(input):
    def search(length, count):
        count += 1
        length /= 2
        if length == 1 or length == 0:
            return 1 + count
        else:
            return 1 + search(length, count)
    return 1 + search(len(input), 1)
```

O(logN) plot



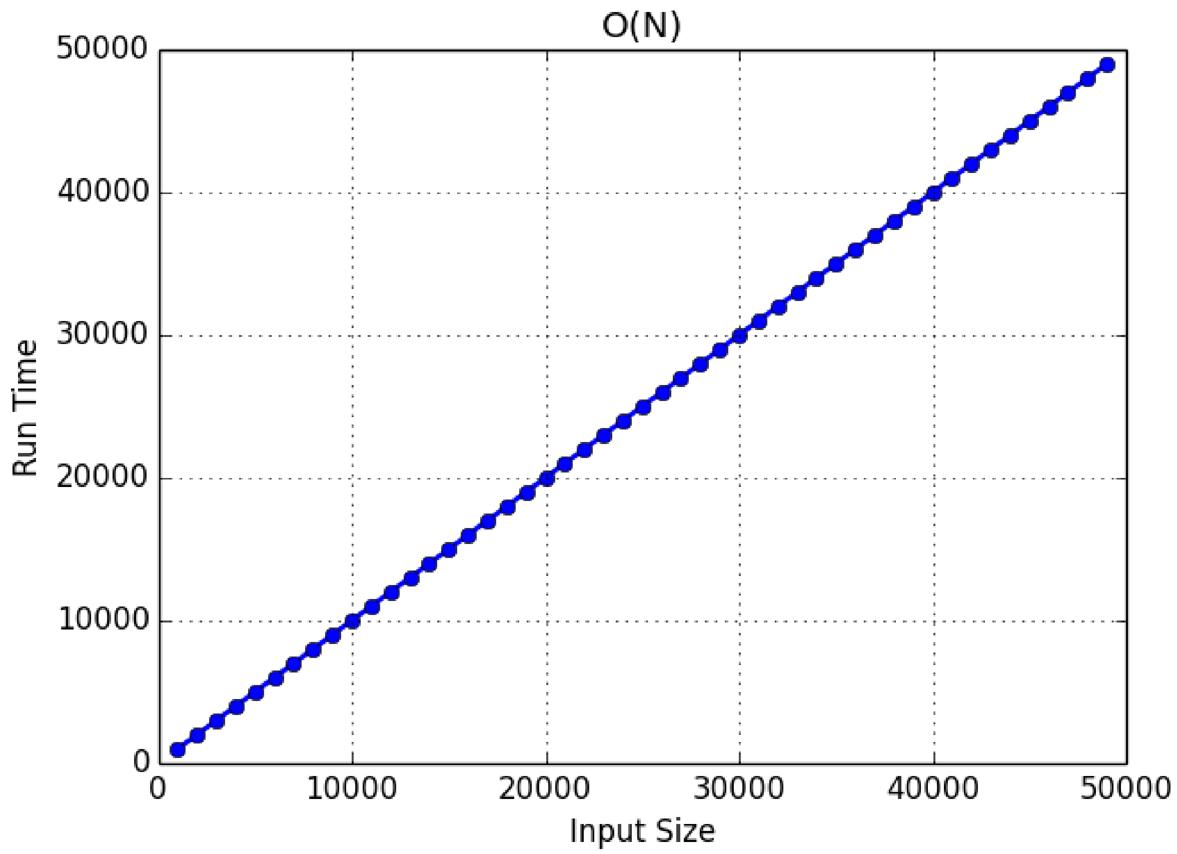
The run time is going up but look at the size of the inputs at the bottom. I start with 10,000 and move up to 500,000. The number of steps has increased, but not significantly.

32.4.3 O(N)

O(N) is linear. This means that the run time is linearly matched to the input size. They should increase at exactly the same rate.

```python
def ON(input, check):
    count = 0
    for number in input:
        count += 1
        if number == check:
            return 1 + count
```

O(N) plot



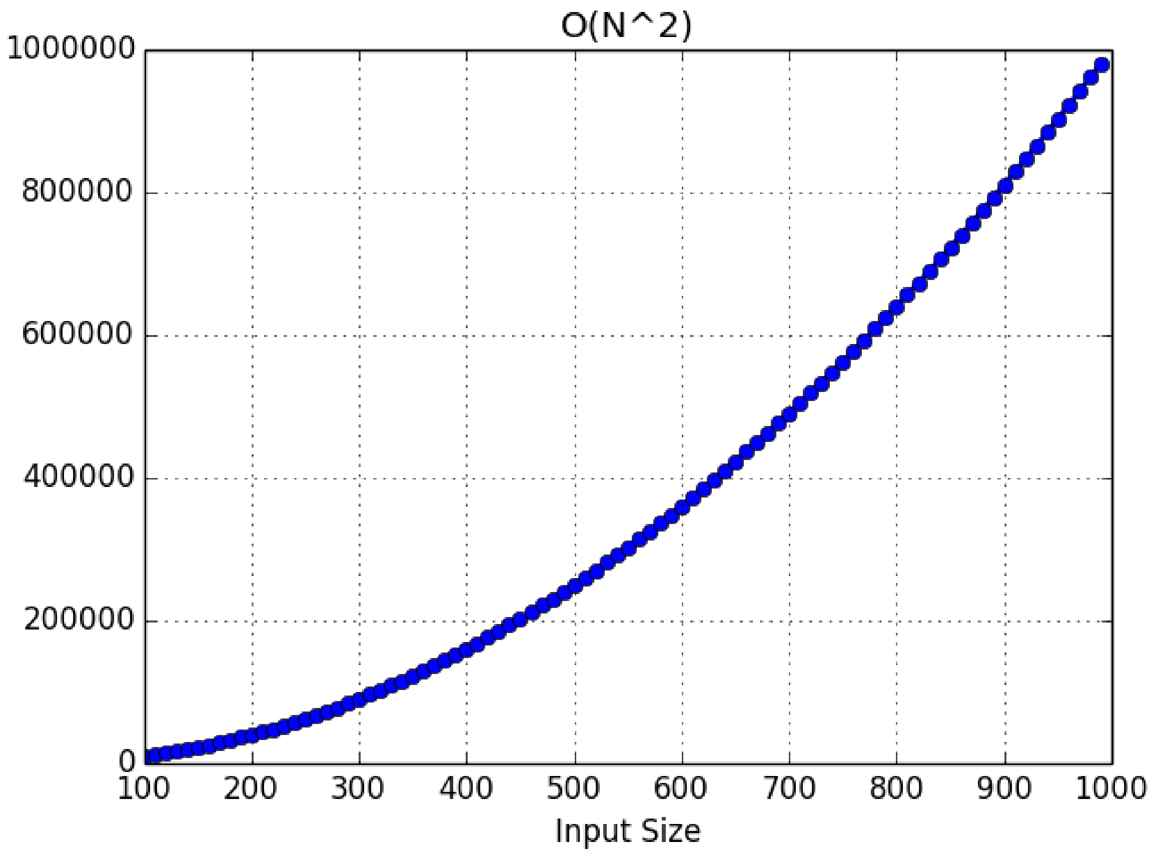
There is a 1:1 correlation between input size and run time. As expected this produces a linear graph.

32.4.4 O(N^2^)

O(N^2^)'s runtime will go up as a square of the input size. The runtime goes up faster than your input sizes, so processing time increases rapidly. This is usually when you iterate through multiple loops at the same time like so:

```
def ON2(input):
    count = 0
    for i in input:
        count += 1
        for j in input:
            count += 1
    return 1 + count
```

O(N^2^) plot



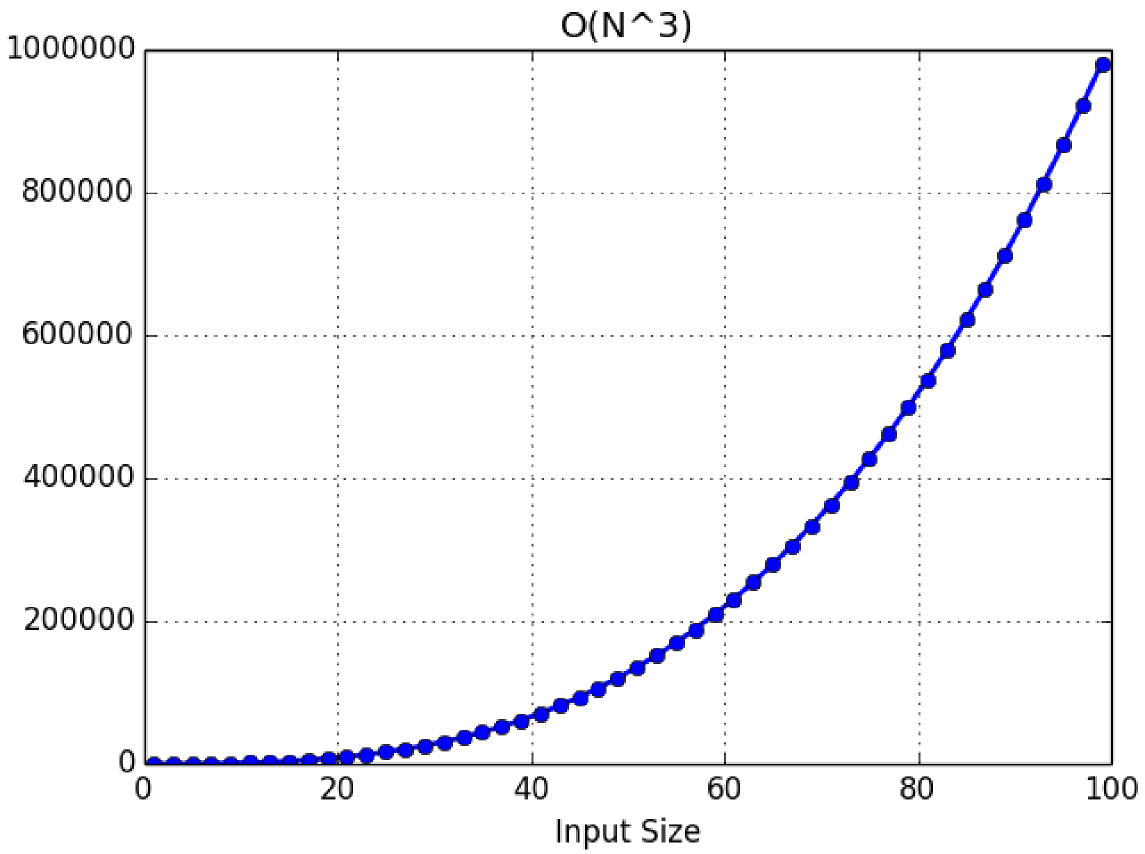
32.4.5 O(N^3^)

O(N^3^)is merely O(N^2^) with another exponent. I wanted to show the difference by simply changing the exponent.

```
def ON3(input):
    count = 0
    for i in input:
        count += 1
        for j in input:
            count += 1
            for k in input:
                count +=1
    return 1 + count
```

O(N^3^) plot

Graphs increase rapidly as the exponent increases.



32.4.6 Conclusions

I've not shown every single type of algorithm, as I just wanted to show the ones I have the most experience with. It's nice to have a visual representation of these things as it really drills down just how fast your runtime can increase with larger inputs.

You can find my code used over here. https://github.com/mellowdrifter/Blog_Code/tree/master/Big_O

End Of Topic

32.5 Big O Notation - Explained

Tip: Big O notation is generally referencing *worst case scenario* for the algorithm

32.5.1 O(1) - Constant Time

O(1) means that it takes a constant time to run an algorithm, regardless of the size of the input. In programming, a lot of operations are constant.

Here are some examples:

- math operations
- accessing an array via the index
- accessing a hash via the key
- pushing and popping on a stack
- insertion and removal from a queue
- returning a value from a function

```
int theArray = new int[Integer.MAX_VALUE]

for(int c = 0; c < Integer.MAX_VALUE; c++)
{
    // This step is O(1)
    theArray[c] = c;
}
```

32.5.2 O(n) - Linear Time

`O(n)` means that the run-time increases at the rate/size as the input.

```
// n is some integer

// The for statement is O(n)
// if n is small, it will take less time
// if n is very large number, it will take more time
```

(continues on next page)

(continued from previous page)

```
for(int c = 0; c < n; c++)
{
    // This step is O(1)
    System.out.println(c)
}
```

32.5.3 O(n^2) - Quadratic Time

O(n^2^) means that the calculation runs in quadratic time.

Examples of algorithms having worst-case run times of O(n^2^):

- Bubble Sort
- Insertion Sort
- Selection Sort

Tip: The general pattern is a for statement within a for statement

```
// Bubble Sort

/*
 * Programmer: James Goudy
 * Project: Bubble Sort
 */
package com.mycompany.bubblesort_lecturecode;

import java.util.Random;

class BubbleSort {

    // arrInt is an array of integers
    // numDataElements is the actual count
    // of elements of data in the array
    // algorithm assumes the data is contiguous
    int arrInt[];
    int numDataElments;

    public BubbleSort(int[] arrInt, int numDataElments) {
        this.arrInt = arrInt;
        this.numDataElments = numDataElments;
    }

    public void Sort() {
        //
        int n = numDataElments;

        for (int c = 0; c < n; c++) {
            for (int j = 1; j < (n); j++) {

                // check if the left element is
                // greater to the one on the right
                // "Bubble" the lowest to the left
```

(continues on next page)

(continued from previous page)

```
                if (arrInt[j - 1] > arrInt[j]) {
                    // swap left element arr[j-1]
                    // with the one on the right and arr[j]

                    // store left one in temp
                    int temp = arrInt[j - 1];
                    //copy the right into the left
                    arrInt[j - 1] = arrInt[j];
                    //copy the left into the right
                    arrInt[j] = temp;
                }
            }
        }
    }

}

public class BubbleSort_LectureCode {

    static int arrSize = 8;
    //static int theArray[] = new int[arrSize];
    static int theArray[] ={3,60,35,2,45,320,5,1};

    static void fillTheArray()
    {
        Random RNG  = new Random();
        for(int c = 0; c < arrSize; c++)
        {
            theArray[c] = RNG.nextInt(0,(arrSize*10));
        }
    }

    static void printArray(int anArray[], int numOfDataElements)
    {
        System.out.println("");
        for (int i = 0; i < numOfDataElements; i++) {
            System.out.print(anArray[i] + " ");
        }
        System.out.println("\n---------------\n");

    }

    public static void main(String[] args) {

        // option to randomly fill the array
        //fillTheArray();

        printArray(theArray, arrSize);

        BubbleSort bs = new BubbleSort(theArray, arrSize);
        bs.Sort();

        printArray(theArray, arrSize);

    }
```

(continues on next page)

(continued from previous page)

```
}

/*
3 60 35 2 45 320 5 1
--------------


1 2 3 5 35 45 60 320
--------------
*/
```

https://betterprogramming.pub/big-o-notation-a-simple-explanation-with-examples-a56347d1daca

---

End Of Topic

CHAPTER

THIRTYTHREE

LINKED LISTS

Definition

Linked List is a set of nodes where each node contains a data field(s) and a reference(link) to the next node in the list.

33.1 Benefits of a linked list

- Not limited to a specific data space amount.
- Easy to add and delete data

33.2 Disadvantages of a linked list

- The retrieval time of stored data is dependent on the size of the list and the position/node of the data in the list. O(n)

33.3 Types of linked lists

- Singly linked lists
- *Doubly linked lists*
- *Doubly linked lists with links as sub class*
- *Circular linked lists*

End of Topic

33.4 Singly Linked List

Ternary Operator

[Singly Linked List](# Singly-Linked-List-Code)

Si

Definition

singly linked list is a type of linked list that is *unidirectional*. It can be traversed in only one direction from the head to the last node (tail). The last node always points to null

33.4.1 Ternary Operator

Ternary Operator is an instruction that consists of three parts. *Condition Part* - test if something is true or false *True Part* - what is returned if the condition is true *False Part* - what is returned if the condition is false

x = **(**Conditional Part**) ** **?** True Part **:** False Part;

```
// Example
y = 3;
x = (y < 4) ? "y is less than four": "y is greater than four"
System.out.write(x);

// Output
// y is less than four
```

33.4.2 Lecture Code

Singly Linked List Code

The *Link* and the controlling *Linked List* are written as two separate classes

```
/*
 * Singly Linked List
 *
 * singlylinkedlists_rev3
 * Programmer: James Goudy
 *
 */
package singlylinkedlists_rev3;

class Link
{

    // Data goes here
    public String city = "";
    public int population = 0;

    // link to next node
    public Link next;
```

(continues on next page)

(continued from previous page)

```
    // constructor
    public Link(String city, int population)
    {
        this.city = city;
        this.population = population;
    }

    // display the link
    public void displayLink()
    {
        System.out.print("{" + city + ", " + population + "} ");
    }

}


class LinkedList
{

    // first is a reference / "address" of the first link
    private Link first;

    // constructor
    public LinkedList()
    {
        first = null;
    }

    //Check if the list is empty
    public boolean isEmpty()
    {
        return (first == null);
    }

    // insert at the front
    // of the list (front[left] -- to --> back[right])
    public void insertFirst(String city, int population)
    {

        // create a new link
        Link newLink = new Link(city, population);

        // the new link is to the left or in front of first
        // the new link will make first in the second spot
        // so newLink.next has to point to the address first
        newLink.next = first;

        // now that the newLink.next is looking at the seond spot
        // or the next spot, first can have the address of the newLink
        first = newLink;

    }

    //This function displays the linked list
    public void displayList()
```

(continues on next page)

(continued from previous page)

```
    {
        System.out.print("\nList\n(first --> last): ");

        //temp variable to hold first
        Link current = first;

        while (current != null)
        {
            current.displayLink();

            // move to the next  link
            current = current.next;
        }

        System.out.println();
    } //end of display list

    public boolean findCity(String city)
    {
        boolean found = false;

        Link current = first;
        // iterate through the loop
        while (current != null)
        {
            // check if current city matches search city
            // set found to true and break out of the loop
            if (current.city.equals(city))
            {
                found = true;
                break;
            }

            current = current.next;

        }

        return found;
    }

    // delete first
    public Link deleteFirst()
    {

        // temp variable to hold first address
        // temp is pointing to an address
        Link temp = first;

        if (!isEmpty())
        {
            // set variable to second spot
            first = first.next;
        }

        // return a link to object
        // in case calling program wants
```

(continues on next page)

(continued from previous page)

```
        // to retreive the deleted data
        return temp;
    }

    public void deleteCity(String city)
    {
        // assumes the data does not have duplicates

        //need to check if the city was found
        boolean found = false;

        Link current = first;
        Link prev = first;
        Link temp;

        // check if city is in the first node
        if (first.city.equals(city))
        {
            temp = first;
            first = first.next;
            temp = null;
            System.out.println("City was deleted");
            return;
        }

        // start at the beginning of list
        while (current != null)
        {
            // check if the current city matches the search city
            if (current.city.equals(city))
            {
                found = true;

                // break out of the while statement
                break;
            }

            // set the prev variable
            prev = current;

            // move to the next node
            current = current.next;
        }

        // check if the while statment made it to
        // the end of the loop
        if (found == false)
        {
            System.out.println("\n*** City not found - Nothing deleted");
            return;
        }

        // delete process
        prev.next = current.next;

        current = null;
```

(continues on next page)

(continued from previous page)

```
        System.out.println("\n*** City was successfully delete");

    }

    public void deleteList()
    {
        while (first != null)
        {
            deleteFirst();
        }

    }

    // city is the location of where the new data
    // will be inserted (after)
    // newCity and newPop are the new city and new population
    public void insertAfter(String city, String newCity, int newPop)
    {
        boolean found = false;
        Link current = first;

        //create a new link
        Link NewLink = new Link(newCity, newPop);

        try
        {
            // iterate through loop
            while (current != null)
            {
                // break out of the loop if found
                // stops the loop at the found city
                // sets found to true
                if (current.city.equals(city))
                {
                    found = true;
                    break;
                }

                // move to next node
                current = current.next;
            }

            // insert process
            NewLink.next = current.next;
            current.next = NewLink;

        } catch (Exception e)
        {
            System.out.println("**Error**\n" + e.getMessage() + "\n***\n");
        }

    }

} // end of class

public class SinglyLinkedLists_Rev3
```

(continues on next page)

(continued from previous page)

```
{

    public static void main(String[] args)
    {
        LinkedList theList = new LinkedList();


        theList.insertFirst("Kali", 32000);
        theList.insertFirst("Whitefish", 7700);
        theList.insertFirst("Polson", 20000);
        theList.insertFirst("Chicago", 13000000);
        theList.insertFirst("Convoy", 500);

        theList.displayList();

        System.out.println("Find Kali : " + theList.findCity("Kali"));

        // ternary operator
        String output = (theList.findCity("Polson") == true)
                ? "City Found" : "City Not Found";
        System.out.println("Polson: " + output);

        output = (theList.findCity("Somers") == true)
                ? "City Found" : "City Not Found";
        System.out.println("Somers: " + output);

        theList.deleteFirst();
        theList.displayList();

        theList.deleteCity("Chicago");
        theList.displayList();

        theList.insertAfter("Polson", "New York", 10000000);
        theList.displayList();

        theList.deleteList();
        theList.displayList();
    }

}
```

Singly Linked List - Nested Link Class

```
/*
 * Single Link List
 *
 * SingleLinkList_nested_Rev3
 * Programmer: James Goudy
 *
 */
package singlylinkedlists_nested_rev3;

class LinkedList
{
```

(continues on next page)

(continued from previous page)

```
    class Link
    {

        // Data goes here
        public String city = "";
        public int population = 0;

        // link to next node
        public Link next;

        // constructor
        public Link(String city, int population)
        {
            this.city = city;
            this.population = population;
        }

        // display the link
        public void displayLink()
        {
            System.out.print("{" + city + ", " + population + "} ");
        }

    }

    // first is a reference / "address" of the first link
    private Link first;

    // constructor
    public LinkedList()
    {
        first = null;
    }

    //Check if the list is empty
    public boolean isEmpty()
    {
        return (first == null);
    }

    // insert at the front
    // of the list (front[left] -- to --> back[right])
    public void insertFirst(String city, int population)
    {

        // create a new link
        Link newLink = new Link(city, population);

        // the new link is to the left or in front of first
        // the new link will make first in the second spot
        // so newLink.next has to point to the address first
        newLink.next = first;

        // now that the newLink.next is looking at the seond spot
        // or the next spot, first can have the address of the newLink
```

(continues on next page)

(continued from previous page)

```
        first = newLink;

    }

    //This function displays the linked list
    public void displayList()
    {
        System.out.print("\nList\n(first --> last): ");

        //temp variable to hold first
        Link current = first;

        while (current != null)
        {
            current.displayLink();

            // move to the next  link
            current = current.next;
        }

        System.out.println();
    } //end of display list

    public boolean findCity(String city)
    {
        boolean found = false;

        Link current = first;
        // iterate through the loop
        while (current != null)
        {
            // check if current city matches search city
            // set found to true and break out of the loop
            if (current.city.equals(city))
            {
                found = true;
                break;
            }

            current = current.next;

        }

        return found;
    }

    // delete first
    public Link deleteFirst()
    {

        // temp variable to hold first address
        // temp is pointing to an address
        Link temp = first;

        if (!isEmpty())
        {
```

(continues on next page)

(continued from previous page)

```
            // set variable to second spot
            first = first.next;
        }

        // return a link to object
        // in case calling program wants
        // to retreive the deleted data
        return temp;
    }

    public void deleteCity(String city)
    {
        // assumes the data does not have duplicates

        //need to check if the city was found
        boolean found = false;

        Link current = first;
        Link prev = first;
        Link temp;

        // check if city is in the first node
        if (first.city.equals(city))
        {
            temp = first;
            first = first.next;
            temp = null;
            System.out.println("City was deleted");
            return;
        }

        // start at the beginning of list
        while (current != null)
        {
            // check if the current city matches the search city
            if (current.city.equals(city))
            {
                found = true;

                // break out of the while statement
                break;
            }

            // set the prev variable
            prev = current;

            // move to the next node
            current = current.next;
        }

        // check if the while statment made it to
        // the end of the loop
        if (found == false)
        {
            System.out.println("\n*** City not found - Nothing deleted");
            return;
```

(continues on next page)

(continued from previous page)

```
        }

        // delete process
        prev.next = current.next;

        current = null;
        System.out.println("\n*** City was successfully delete");

    }

    public void deleteList()
    {
        while (first != null)
        {
            deleteFirst();
        }

    }

    // city is the location of where the new data
    // will be inserted (after)
    // newCity and newPop are the new city and new population
    public void insertAfter(String city, String newCity, int newPop)
    {
        boolean found = false;
        Link current = first;

        //create a new link
        Link NewLink = new Link(newCity, newPop);

        try
        {
            // iterate through loop
            while (current != null)
            {
                // break out of the loop if found
                // stops the loop at the found city
                // sets found to true
                if (current.city.equals(city))
                {
                    found = true;
                    break;
                }

                // move to next node
                current = current.next;
            }

            // insert process
            NewLink.next = current.next;
            current.next = NewLink;

        } catch (Exception e)
        {
            System.out.println("**Error**\n" + e.getMessage() + "\n***\n");
        }
```

(continues on next page)

(continued from previous page)

```java
    }

} // e

public class Singlylinkedlists_nested_rev3
{

    public static void main(String[] args)
    {

        LinkedList theList = new LinkedList();

        // how to create a Link from a nested class
        // note that the new link has to be created from the
        // outer link ('theList')
        LinkedList.Link aLink =  theList.new Link("Detroit", 2000000);

        // display the created link
        System.out.println("Created Inner Link");
        aLink.displayLink();
        System.out.println("\n\n");



        theList.insertFirst("Kali", 32000);
        theList.insertFirst("Whitefish", 7700);
        theList.insertFirst("Polson", 20000);
        theList.insertFirst("Chicago", 13000000);
        theList.insertFirst("Convoy", 500);

        theList.displayList();

        System.out.println("Find Kali : " + theList.findCity("Kali"));

        // ternary operator
        String output = (theList.findCity("Polson") == true)
                ? "City Found" : "City Not Found";
        System.out.println("Polson: " + output);

        output = (theList.findCity("Somers") == true)
                ? "City Found" : "City Not Found";
        System.out.println("Somers: " + output);

        theList.deleteFirst();
        theList.displayList();

        theList.deleteCity("Chicago");
        theList.displayList();

        theList.insertAfter("Polson", "New York", 10000000);
        theList.displayList();

        theList.deleteList();
        theList.displayList();
```

(continues on next page)

(continued from previous page)

```
    }

}
```

End Of Topic

33.5 Doubly Linked List

33.5.1 Key Ideas

- The doubly linked list allows the list to be traversed in both directions; forwards and backwards

33.5.2 Lecture Code

```
/*
 * Programmer: James Goudy
 * Project: Doubly Linked List
 */


class Link {

    Link first = null;
    Link last = null;

    // data
    String city = null;

    // link navigation
    Link next = null;
    Link prev = null;

    // constructor
    Link(String city) {
        this.city = city;
        this.next = null;
        this.prev = null;
    }

    public void displayNode() {
        System.out.print(city + " ");
    }
} // end of link


class Doubly {

    Link first = null;
    Link last = null;

```

(continues on next page)

(continued from previous page)

```
    // constructor
    public Doubly() {

        first = null;
        last = null;
    }

    // add link at the beginning of the list
    public boolean addFirst(String city) {
        Link newLink = new Link(city);

        if (first == null) {
            // if list is empty
            first = newLink;
            last = newLink;
        } else {
            newLink.next = first;
            first.prev = newLink;
            first = newLink;
        }

        return true;
    }

    // add link to the end of the list
    public boolean addLast(String city) {

        Link newLink = new Link(city);

        if (first == null) {
            // if list is empty
            first = newLink;
            last = newLink;
        } else {
            newLink.prev = last;
            last.next = newLink;
            last = newLink;
        }

        return true;
    }

    public boolean findCity(String citySearch) {

        if (first == null) {

            // if list is empty
            return false;
        } else {
            Link current = first;

            while (current != null) {
                if (current.city.equals(citySearch)) {
                    return true;
                }
                current = current.next;
```

(continues on next page)

(continued from previous page)

```
            }

            return false;
        }
    }

    public boolean insertAfter(String citySearch, String insertCity) {

        Link newLink = new Link(insertCity);

        if (first == null) {

            // list is empty - add the link
            first = newLink;
            last = newLink;

            // NOTE: there is an option not to insert
            // a link then the code above would be replaced
            // with return false
        } else {
            Link current = first;

            while (current != null) {
                if (current.city.equals(citySearch)) {

                    // check if last link
                    if (current.next == null) {
                        // check if last link
                        current.next = newLink;
                        newLink.prev = current;

                        last = newLink;

                    } else {
                        newLink.next = current.next;
                        newLink.prev = current;

                        current.next.prev = newLink;
                        current.next = newLink;
                    }

                    return true;
                }
                current = current.next;

            }
        }

        return false;

    }

    public boolean insertBefore(String citySearch, String insertCity) {

        Link newLink = new Link(insertCity);
```

(continues on next page)

(continued from previous page)

```
        if (first == null) {

            // list is empty - add the link
            first = newLink;
            last = newLink;

            // NOTE: there is an option not to insert
            // a link then the code above would be replaced
            // with return false
        } else {
            Link current = first;

            while (current != null) {
                if (current.city.equals(citySearch)) {

                    // check for first link
                    if (current.prev == null) {
                        // check if last link
                        current.prev = newLink;
                        newLink.next = current;

                        first = newLink;

                    } else {
                        newLink.next = current;
                        newLink.prev = current.prev;

                        current.prev.next = newLink;
                        current.prev = newLink;
                    }

                    return true;
                }
                current = current.next;

            }
        }

        return false;
    }

    public boolean deleteCity(String citySearch) {

        if (first == null) {
            return false;
        } else {
            Link current = first;

            while (current != null) {
                if (current.city.equals(citySearch)) {

                    if (current.prev == null) {
                        // first node
                        current.next.prev = null;
                        first = current.next;
                        current = null;
```

(continues on next page)

(continued from previous page)

```
                        return true;
                    } else if (current.next == null) {
                        // last node
                        current.prev.next = null;
                        last = current;
                        current = null;
                        return true;
                    } else {
                        // a center node
                        current.prev.next = current.next;
                        current.next.prev = current.prev;
                        current = null;

                        return true;
                    }
                }
                current = current.next;
            }

            return false;
        }
    } //end of function

    public void displayList() {
        Link current = first;

        System.out.println("");
        while (current != null) {
            current.displayNode();
            current = current.next;
        }
        System.out.println("");

    }
}

public class DS_DoublyLinkedList {

    static Doubly dl = new Doubly();

    public static void citySearch(String searchCity) {

        // note that findCity returns a boolean
        // so it can be used in an "if" statement
        if (dl.findCity(searchCity)) {
            System.out.println("\n" + searchCity + " is in list");
        } else {
            System.out.println("\n" + searchCity + " not found");
        }

    }

    public static void deleteCity(String searchCity) {

        // note that findCity returns a boolean
        // so it can be used in an "if" statement
```

(continues on next page)

(continued from previous page)

```
        if (dl.deleteCity(searchCity)) {
            System.out.println(searchCity + " was deleted");
        } else {
            System.out.println(searchCity + " was NOT deleted");
        }

    }

    public static void main(String[] args) {

        String searchCity = "";
        String insertCity = "";

        // insert data at front of list
        dl.addFirst("Kali");
        dl.addFirst("Polson");
        dl.addFirst("Missoula");
        dl.addFirst("Whitefish");

        // insert data at end of list
        dl.addLast("Chicago");
        dl.addLast("Denver");
        dl.addLast("Sandiego");

        dl.displayList();

        System.out.println("\n----- Find Examples------\n");

        searchCity = "Chicago";
        citySearch(searchCity);

        searchCity = "Bozeman";
        citySearch(searchCity);

        System.out.println("\n----- Delete Examples------\n");

        searchCity = "Polson";
        deleteCity(searchCity);

        searchCity = "Bozeman";
        deleteCity(searchCity);

        searchCity = "Sandiego";
        deleteCity(searchCity);

        searchCity = "Whitefish";
        deleteCity(searchCity);

        dl.displayList();

        System.out.println("\n----- Insert Examples------\n");

        searchCity = "Missoula";
        insertCity = "Dayton";
        dl.insertAfter(searchCity, insertCity);
```

(continues on next page)

(continued from previous page)

```
        searchCity = "Denver";
        insertCity = "Boulder";
        dl.insertAfter(searchCity, insertCity);

        dl.displayList();

        searchCity = "Chicago";
        insertCity = "Springfield";
        dl.insertBefore(searchCity, insertCity);

        searchCity = "Missoula";
        insertCity = "Libby";
        dl.insertBefore(searchCity, insertCity);

        dl.displayList();

        System.out.println("\nbye");
    }
}

/*
 * OUTPUT  *
 * Whitefish Missoula Polson Kali Chicago Denver Sandiego  *
 * ----- Find Examples------
 *
 *
 * Chicago is in list
 *
 * Bozeman not found
 *
 * ----- Delete Examples------
 *
 * Polson was deleted
 * Bozeman was NOT deleted
 * Sandiego was deleted
 * Whitefish was deleted
 *
 * Missoula Kali Chicago Denver  *
 * ----- Insert Examples------
 *
 *
 * Missoula Dayton Kali Chicago Denver Boulder  *
 * Libby Missoula Dayton Kali Springfield Chicago Denver Boulder  *
 * bye
 *
 */
```

End Of Topic

33.6 Doubly Linked List - Links as Sub Class

33.6.1 Key Ideas

- The doubly linked list allows the list to be traversed in both directions; forwards and backwards

Note: Not all languages support subclasses.

33.6.2 Lecture Code

```
/*
 * Programmer: James Goudy
 * Project: Doubly Linked List written
 * with  Links / Nodes as SubClass
 */

class Doubly {

    Link first = null;
    Link last = null;

    // ---------------------------------
    // sub class - Link / Nodes
    // NOTE: this can be a separate class as well

    class Link {

        Link first = null;
        Link last = null;

        // data
        String city = null;

        // link navigation
        Link next = null;
        Link prev = null;

        // constructor
        Link(String city) {
            this.city = city;
            this.next = null;
            this.prev = null;
        }

        public void displayNode() {
            System.out.print(city + " ");
        }
    } // end of link
    // ---------------------------------

    // constructor
    public Doubly() {
```

(continues on next page)

(continued from previous page)

```
        first = null;
        last = null;
    }

    // add link at the beginning of the list
    public boolean addFirst(String city) {
        Link newLink = new Link(city);

        if (first == null) {
            // if list is empty
            first = newLink;
            last = newLink;
        } else {
            newLink.next = first;
            first.prev = newLink;
            first = newLink;
        }

        return true;
    }

    // add link to the end of the list
    public boolean addLast(String city) {

        Link newLink = new Link(city);

        if (first == null) {
            // if list is empty
            first = newLink;
            last = newLink;
        } else {
            newLink.prev = last;
            last.next = newLink;
            last = newLink;
        }

        return true;
    }

    public boolean findCity(String citySearch) {

        if (first == null) {

            // if list is empty
            return false;
        } else {
            Link current = first;

            while (current != null) {
                if (current.city.equals(citySearch)) {
                    return true;
                }
                current = current.next;
            }

            return false;
```

(continues on next page)

(continued from previous page)

```
        }
    }

    public boolean insertAfter(String citySearch, String insertCity) {

        Link newLink = new Link(insertCity);

        if (first == null) {

            // list is empty - add the link
            first = newLink;
            last = newLink;

            // NOTE: there is an option not to insert
            // a link then the code above would be replaced
            // with return false

        } else {
            Link current = first;

            while (current != null) {
                if (current.city.equals(citySearch)) {

                    // check if last link
                    if (current.next == null) {
                        // check if last link
                        current.next = newLink;
                        newLink.prev = current;

                        last = newLink;

                    } else {
                        newLink.next = current.next;
                        newLink.prev = current;

                        current.next.prev = newLink;
                        current.next = newLink;
                    }

                    return true;
                }
                current = current.next;

            }
        }

        return false;

    }

    public boolean insertBefore(String citySearch, String insertCity) {

        Link newLink = new Link(insertCity);

        if (first == null) {
```

(continues on next page)

(continued from previous page)

```
            // list is empty - add the link
            first = newLink;
            last = newLink;

            // NOTE: there is an option not to insert
            // a link then the code above would be replaced
            // with return false

        } else {
            Link current = first;

            while (current != null) {
                if (current.city.equals(citySearch)) {

                    // check for first link
                    if (current.prev == null) {
                        // check if last link
                        current.prev = newLink;
                        newLink.next = current;

                        first = newLink;

                    } else {
                        newLink.next = current;
                        newLink.prev = current.prev;

                        current.prev.next = newLink;
                        current.prev = newLink;
                    }

                    return true;
                }
                current = current.next;

            }
        }

        return false;
    }

    public boolean deleteCity(String citySearch) {

        if (first == null) {
            return false;
        } else {
            Link current = first;

            while (current != null) {
                if (current.city.equals(citySearch)) {

                    if (current.prev == null) {
                        // first node
                        current.next.prev = null;
                        first = current.next;
                        current = null;
                        return true;
```

(continues on next page)

(continued from previous page)

```
                    } else if (current.next == null) {
                        // last node
                        current.prev.next = null;
                        last = current;
                        current = null;
                        return true;
                    } else {
                        // a center node
                        current.prev.next = current.next;
                        current.next.prev = current.prev;
                        current = null;

                        return true;
                    }
                }
                current = current.next;
            }

            return false;
        }
    } //end of function

    public void displayList() {
        Link current = first;

        System.out.println("");
        while (current != null) {
            current.displayNode();
            current = current.next;
        }
        System.out.println("");
    }

    public void displayListBackwards() {
        Link current = last;

        System.out.println("");
        while (current != null) {
            current.displayNode();
            current = current.prev;
        }
        System.out.println("");
    }

}

public class J2_SubClass {

    static Doubly dl = new Doubly();

    public static void citySearch(String searchCity) {

        // note that findCity returns a boolean
        // so it can be used in an "if" statement
        if (dl.findCity(searchCity)) {
            System.out.println("\n" + searchCity + " is in list");
```

(continues on next page)

(continued from previous page)

```
        } else {
            System.out.println("\n" + searchCity + " not found");
        }

    }

    public static void deleteCity(String searchCity) {

        // note that findCity returns a boolean
        // so it can be used in an "if" statement
        if (dl.deleteCity(searchCity)) {
            System.out.println(searchCity + " was deleted");
        } else {
            System.out.println(searchCity + " was NOT deleted");
        }

    }

    public static void main(String[] args) {

        String searchCity = "";
        String insertCity = "";

        // insert data at front of list
        dl.addFirst("Kali");
        dl.addFirst("Polson");
        dl.addFirst("Missoula");
        dl.addFirst("Whitefish");

        // insert data at end of list
        dl.addLast("Chicago");
        dl.addLast("Denver");
        dl.addLast("Sandiego");

        dl.displayList();

        System.out.println("\n----- Find Examples------\n");

        searchCity = "Chicago";
        citySearch(searchCity);

        searchCity = "Bozeman";
        citySearch(searchCity);

        System.out.println("\n----- Delete Examples------\n");

        searchCity = "Polson";
        deleteCity(searchCity);

        searchCity = "Bozeman";
        deleteCity(searchCity);

        searchCity = "Sandiego";
        deleteCity(searchCity);

        searchCity = "Whitefish";
```

(continues on next page)

(continued from previous page)

```
        deleteCity(searchCity);

        dl.displayList();

        System.out.println("\n----- Insert Examples------\n");

        searchCity = "Missoula";
        insertCity = "Dayton";
        dl.insertAfter(searchCity, insertCity);

        searchCity = "Denver";
        insertCity = "Boulder";
        dl.insertAfter(searchCity, insertCity);

        dl.displayList();

        searchCity = "Chicago";
        insertCity = "Springfield";
        dl.insertBefore(searchCity, insertCity);

        searchCity = "Missoula";
        insertCity = "Libby";
        dl.insertBefore(searchCity, insertCity);

        dl.displayList();

        System.out.println("\nbye");
    }
}

/*
 OUTPUT

 Whitefish Missoula Polson Kali Chicago Denver Sandiego

----- Find Examples------


Chicago is in list

Bozeman not found

----- Delete Examples------

Polson was deleted
Bozeman was NOT deleted
Sandiego was deleted
Whitefish was deleted

Missoula Kali Chicago Denver

----- Insert Examples------


Missoula Dayton Kali Chicago Denver Boulder

```

(continues on next page)

(continued from previous page)

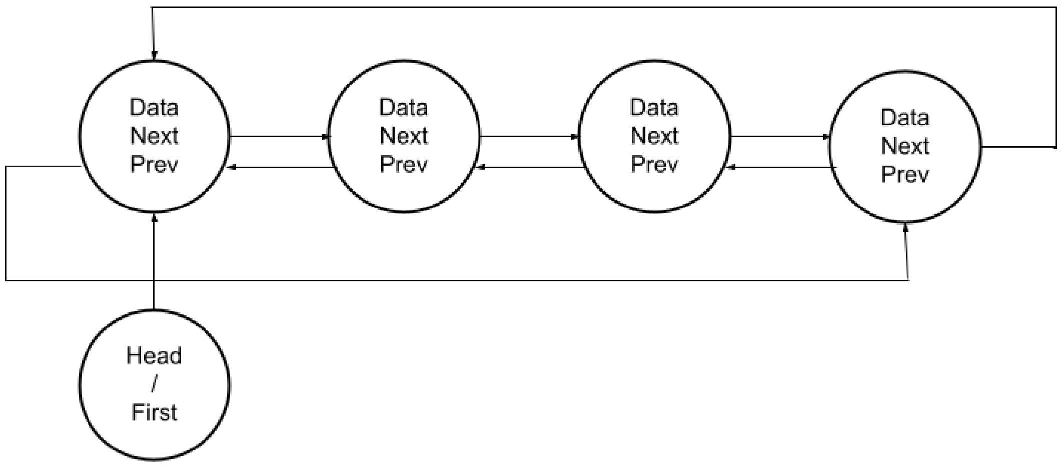
```
Libby Missoula Dayton Kali Springfield Chicago Denver Boulder

bye

 */
```

End Of Topic

33.7 Circular Linked List - Links as Sub Class

33.7.1 Key Ideas

- The doubly linked list allows the list to be traversed in both directions; forwards and backwards
- In a circular doubly-linked list the last node (next) points to the first. The first node (previous) points to the last node.
- The doubly linked list allows the list to be traversed in both directions; forwards and backwards



Note: Not all languages support subclasses.

33.7.2 Lecture Code

```
package com.mycompany.linkedlistcircular;

/*
 * Programmer: James Goudy
 * Project Circular LinkedList
 *
 * NOTE: last link is referenced as first.prev
 */
class CircularLinkedList {

    Link first = new Link("");

    // ---------------------------------
    // sub class - Link / Nodes
    // NOTE: this can be a separate class as well
    class Link {

        Link first = null;

        // data
        String city = null;

        // link navigation
        Link next = null;
        Link prev = null;

        // constructor
        Link(String city) {
            this.city = city;
            this.next = null;
            this.prev = null;
        }

        public void displayNode() {
            System.out.print(city + "  ");
        }
    } // end of link
    // ---------------------------------

    // constructor
    public CircularLinkedList() {

        first = null;
    }

    // add link at the beginning of the list
    public boolean addFirst(String city) {

        Link newLink = new Link(city);

        if (first == null) {
            // empty list
            newLink.next = newLink;
            newLink.prev = newLink;
```

(continues on next page)

(continued from previous page)

```
            first = newLink;

        } else {

            // connect the newLink references
            newLink.next = first;

            newLink.prev = first.prev;

            first.prev = newLink;

            // move first to the new link
            first = newLink;

            // point the last link to the new first
            first.prev.next = first;

        }

        return true;
    }

    // add link to the end of the list
    public boolean addLast(String city) {

        Link newLink = new Link(city);

        if (first == null) {
            //list is empty
            first = newLink;
        } else {
            // set new link references
            newLink.next = first;

            newLink.prev = first.prev;

            // last link is (first.prev)
            first.prev.next = newLink;

            first.prev = newLink;

        }

        return true;
    }

    public boolean findCity(String citySearch) {

        if (first == null) {

            // if list is empty
            return false;
        } else {
            Link current = first;

            do {
```

(continues on next page)

(continued from previous page)

```
                if (current.city.equals(citySearch)) {
                    return true;
                }
                current = current.next;

            } while (current != first);

            return false;
        }
    }

    public boolean insertAfter(String citySearch, String insertCity) {

        Link newLink = new Link(insertCity);

        if (first == null) {

            // list is empty - add the link
            first = newLink;

            // NOTE: there is an option not to insert
            // a link then the code above would be replaced
            // with return false
        } else {
            Link current = first;

            while (current != null) {
                if (current.city.equals(citySearch)) {

                    // check if last link
                    if (current.next == first.prev) {
                        // check if last link
                        current.next = newLink;
                        newLink.prev = current;

                        first.prev = newLink;

                    } else {
                        newLink.next = current.next;
                        newLink.prev = current;

                        current.next.prev = newLink;
                        current.next = newLink;
                    }

                    return true;
                }
                current = current.next;

            }
        }

        return false;

    }
```

(continues on next page)

(continued from previous page)

```java
    public boolean insertBefore(String citySearch, String insertCity) {

        Link newLink = new Link(insertCity);

        if (first == null) {

            // list is empty - add the link
            first = newLink;

            // NOTE: there is an option not to insert
            // a link then the code above would be replaced
            // with return false
        } else {
            Link current = first;

            while (current != null) {
                if (current.city.equals(citySearch)) {

                    // check for first link
                    if (current.prev == null) {
                        // check if last link
                        current.prev = newLink;
                        newLink.next = current;

                        first = newLink;

                    } else {
                        newLink.next = current;
                        newLink.prev = current.prev;

                        current.prev.next = newLink;
                        current.prev = newLink;
                    }

                    return true;
                }
                current = current.next;

            }
        }

        return false;
    }

    public boolean deleteCity(String citySearch) {

        if (first == null) {
            return false;
        } else {
            Link current = first;

            do {
                if (current.city.equals(citySearch)) {

                    if (current.prev == null) {
                        // first node
```

(continues on next page)

(continued from previous page)

```
                        current.next.prev = null;
                        first = current.next;
                        current = null;
                        return true;
                    } else if (current.next == null) {
                        // last node
                        current.prev.next = null;
                        first.prev = current;
                        current = null;
                        return true;
                    } else {
                        // a center node
                        current.prev.next = current.next;
                        current.next.prev = current.prev;
                        current = null;

                        return true;
                    }
                }

                current = current.next;
            } while (current != first);

            return false;
        }
    } //end of function

    public void displayList() {
        Link current = first;

        System.out.println("\n** Display List Forward To Back **");

        do {
            current.displayNode();
            current = current.next;
            if (current == first) {
                System.out.println("\n-----------\n");
                return;
            }

        } while (current.next != null);

    } // end of method

    public void displayList(String startCity) {

        Link current = first;
        Link start = null;

        System.out.println("\n ** Display List Forward To Back"
                + " starting at " + startCity + "**");

        // find the city in the list
        do {
            if (current.city.equals(startCity)) {
                break;
```

(continues on next page)

(continued from previous page)

```java
            }

            current = current.next;

            if (current == first) {
                System.out.println("City Not Found");
                return;
            }

        } while (current.next != null);

        System.out.println("");

        start = current;

        do {
            current.displayNode();
            current = current.next;
            if (current == start) {
                System.out.println("\n-----------\n");
                return;
            }

        } while (current.next != null);

    } // end of method

    public void displayListReverse() {

        Link current = first.prev;

        System.out.println("\n** Display List in Reverse **\n");

        do {
            current.displayNode();
            current = current.prev;
            if (current == first.prev) {
                System.out.println("\n-----------\n");
                return;
            }

        } while (current.prev != null);
    } // end of method

    public void displayListReverse(String startCity) {

        Link current = first;
        Link start = null;

        System.out.println("\n ** Display list in reverse"
                + " starting at " + startCity + "**");

        // find the city in the list
        do {
            if (current.city.equals(startCity)) {
                break;
```

(continues on next page)

(continued from previous page)

```
            }

            current = current.next;

            if (current == first) {
                System.out.println("City Not Found");
                return;
            }

        } while (current.next != null);

        System.out.println("");

        start = current;

        do {
            current.displayNode();
            current = current.prev;
            if (current == start) {
                System.out.println("\n-----------\n");
                return;
            }

        } while (current.prev != null);
    } // end of method

} // end of class

public class DS_LinkedListCircular {

    static CircularLinkedList dl = new CircularLinkedList();

    public static void citySearch(String searchCity) {

        // note that findCity returns a boolean
        // so it can be used in an "if" statement
        if (dl.findCity(searchCity)) {
            System.out.println("\n" + searchCity + " is in list");
        } else {
            System.out.println("\n" + searchCity + " not found");
        }

    }

    public static void deleteCity(String searchCity) {

        // note that findCity returns a boolean
        // so it can be used in an "if" statement
        if (dl.deleteCity(searchCity)) {
            System.out.println(searchCity + " was deleted");
        } else {
            System.out.println(searchCity + " was NOT deleted");
        }

    }
```

(continues on next page)

(continued from previous page)

```
    public static void main(String[] args) {

        String searchCity = "";
        String insertCity = "";

        // insert data at front of list
        dl.addFirst("Kali");
        dl.addFirst("Polson");
        dl.addFirst("Missoula");
        dl.addFirst("Whitefish");
        dl.addFirst("Plains");

        // insert data at end of list
        dl.addLast("Chicago");
        dl.addLast("Denver");
        dl.addLast("Sandiego");

        dl.displayList();
        dl.displayListReverse();

        dl.displayList("Missoula");
        dl.displayList("Wolfcreek");

        dl.displayListReverse();
        dl.displayListReverse("Missoula");

        System.out.println("\n----- Find Examples------\n");

        searchCity = "Missoula";
        citySearch(searchCity);

        searchCity = "Bozeman";
        citySearch(searchCity);

        System.out.println("\n----- Delete Examples------\n");

        searchCity = "Polson";
        deleteCity(searchCity);

        searchCity = "Bozeman";
        deleteCity(searchCity);

        searchCity = "Sandiego";
        deleteCity(searchCity);

        searchCity = "Whitefish";
        deleteCity(searchCity);

        dl.displayList();

        System.out.println("\n----- Insert After Examples------\n");

        searchCity = "Missoula";
        insertCity = "Dayton";
        dl.insertAfter(searchCity, insertCity);
```

(continues on next page)

(continued from previous page)

```
        searchCity = "Denver";
        insertCity = "Boulder";
        dl.insertAfter(searchCity, insertCity);

        dl.displayList();

        System.out.println("\n----- Insert After Examples------\n");

        searchCity = "Chicago";
        insertCity = "Springfield";
        dl.insertBefore(searchCity, insertCity);

        searchCity = "Missoula";
        insertCity = "Libby";
        dl.insertBefore(searchCity, insertCity);

        dl.displayList();
        System.out.println("\nbye");
    }
}
/*
 * output
 *
 ** Display List Forward To Back **
 * Plains Whitefish Missoula Polson Kali Chicago Denver Sandiego
 * -----------
 *
 *
 ** Display List in Reverse **
 *
 * Sandiego Denver Chicago Kali Polson Missoula Whitefish Plains
 * -----------
 *
 *
 ** Display List Forward To Back starting at Missoula**
 *
 * Missoula Polson Kali Chicago Denver Sandiego Plains Whitefish
 * -----------
 *
 *
 ** Display List Forward To Back starting at Wolfcreek**
 * City Not Found
 *
 ** Display List in Reverse **
 *
 * Sandiego Denver Chicago Kali Polson Missoula Whitefish Plains
 * -----------
 *
 *
 ** Display list in reverse starting at Missoula**
 *
 * Missoula Whitefish Plains Sandiego Denver Chicago Kali Polson
 * -----------
 *
 *
 * ----- Find Examples------
```

(continues on next page)

(continued from previous page)

```
 *
 *
 * Missoula is in list
 *
 * Bozeman not found
 *
 * ----- Delete Examples------
 *
 * Polson was deleted
 * Bozeman was NOT deleted
 * Sandiego was deleted
 * Whitefish was deleted
 *
 ** Display List Forward To Back **
 * Plains Missoula Kali Chicago Denver
 * -----------
 *
 *
 * ----- Insert After Examples------
 *
 *
 ** Display List Forward To Back **
 * Plains Missoula Dayton Kali Chicago Denver Boulder
 * -----------
 *
 *
 * ----- Insert After Examples------
 *
 *
 ** Display List Forward To Back **
 * Plains Libby Missoula Dayton Kali Springfield Chicago Denver Boulder
 * -----------
 *
 *
 * bye
 *
 */
```

End Of Topic

33.8 Reading Datafile Into Linked List

33.8.1 Lecture Code

```
/*
Programmer: James Goudy
Project: Reading a csv file into a linked list
 */
```

(continues on next page)

(continued from previous page)

```
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.util.logging.Level;
import java.util.logging.Logger;

class Link {

    Link first = null;
    Link last = null;

    // data
    String id = null;
    String firstname;
    String lastname;
    String pet;


    // link navigation
    Link next = null;
    Link prev = null;

    // constructor
    public Link(String id, String firstname, String lastname, String pet) {
        this.id = id;
        this.firstname = firstname;
        this.lastname = lastname;
        this.pet = pet;
    }


    public void displayNode() {
        System.out.println("{"+ firstname + " " + lastname + " "
                                    +pet+ " " + id +" } " );
    }
} // end of link




class Doubly {

    Link first = null;
    Link last = null;

    // constructor
    public Doubly() {

        first = null;
        last = null;
    }

    // add link at the beginning of the list
    public boolean addFirst(String id,String firstname,
            String lastname, String pet) {
        Link newLink = new Link( id,firstname,  lastname,  pet);
```

(continues on next page)

(continued from previous page)

```
        if (first == null) {
            // if list is empty
            first = newLink;
            last = newLink;
        } else {
            newLink.next = first;
            first.prev = newLink;
            first = newLink;
        }

        return true;
    }

    // add link to the end of the list
    public boolean addLast(String id,String firstname,
            String lastname, String pet) {
        Link newLink = new Link( id,firstname,  lastname,  pet);

        if (first == null) {
            // if list is empty
            first = newLink;
            last = newLink;
        } else {
            newLink.prev = last;
            last.next = newLink;
            last = newLink;
        }

        return true;
    }

    public boolean findId(String searchID) {

        if (first == null) {

            // if list is empty
            return false;
        } else {
            Link current = first;

            while (current != null) {
                if (current.id.equals(searchID)) {
                    return true;
                }
                current = current.next;
            }

            return false;
        }
    }

    public boolean insertAfter(String searchId, String id,String firstname,
            String lastname, String pet) {
        Link newLink = new Link( id,firstname,  lastname,  pet);
```

(continues on next page)

(continued from previous page)

```
        if (first == null) {

            // list is empty - add the link
            first = newLink;
            last = newLink;

            // NOTE: there is an option not to insert
            // a link then the code above would be replaced
            // with return false
        } else {
            Link current = first;

            while (current != null) {
                if (current.id.equals(searchId)) {

                    // check if last link
                    if (current.next == null) {
                        // check if last link
                        current.next = newLink;
                        newLink.prev = current;

                        last = newLink;

                    } else {
                        newLink.next = current.next;
                        newLink.prev = current;

                        current.next.prev = newLink;
                        current.next = newLink;
                    }

                    return true;
                }
                current = current.next;

            }
        }

        return false;

    }

    public boolean insertBefore(String searchId, String id,String firstname,
            String lastname, String pet) {

        Link newLink = new Link( id,firstname,  lastname,  pet);

        if (first == null) {

            // list is empty - add the link
            first = newLink;
            last = newLink;

            // NOTE: there is an option not to insert
            // a link then the code above would be replaced
            // with return false
```

(continues on next page)

(continued from previous page)

```
        } else {
            Link current = first;

            while (current != null) {
                if (current.id.equals(searchId)) {

                    // check for first link
                    if (current.prev == null) {
                        // check if last link
                        current.prev = newLink;
                        newLink.next = current;

                        first = newLink;

                    } else {
                        newLink.next = current;
                        newLink.prev = current.prev;

                        current.prev.next = newLink;
                        current.prev = newLink;
                    }

                    return true;
                }
                current = current.next;

            }
        }

        return false;
    }

    public boolean deleteId(String searchID) {

        if (first == null) {
            return false;
        } else {
            Link current = first;

            while (current != null) {
                if (current.id.equals(searchID)) {

                    if (current.prev == null) {
                        // first node
                        current.next.prev = null;
                        first = current.next;
                        current = null;
                        return true;
                    } else if (current.next == null) {
                        // last node
                        current.prev.next = null;
                        last = current;
                        current = null;
                        return true;
                    } else {
                        // a center node
```

(continues on next page)

(continued from previous page)

```
                        current.prev.next = current.next;
                        current.next.prev = current.prev;
                        current = null;

                        return true;
                    }
                }
                current = current.next;
            }

            return false;
        }
    } //end of function

    public void displayList() {
        Link current = first;

        System.out.println("");
        while (current != null) {
            current.displayNode();
            current = current.next;
        }
        System.out.println("");

    }
}

public class DS_ReadDataIntoLinkedList {

    static Doubly dl = new Doubly();

    public static void idSearch(String searchId) {

        // note that findCity returns a boolean
        // so it can be used in an "if" statement
        if (dl.findId(searchId)) {
            System.out.println("\n" + searchId + " is in list");
        } else {
            System.out.println("\n" + searchId + " not found");
        }

    }

    public static void deleteID(String searchId) {

        // note that findCity returns a boolean
        // so it can be used in an "if" statement
        if (dl.deleteId(searchId)) {
            System.out.println(searchId + " was deleted");
        } else {
            System.out.println(searchId + " was NOT deleted");
        }

    }
```

(continues on next page)

(continued from previous page)

```
    public static boolean addData(String filePath)
    {
        BufferedReader br;
        FileReader fr;
        String inputLine;

        try {
            // create buffered reader
            br = new BufferedReader(new FileReader(filePath));

            // skip the header line
            br.readLine();

            while((inputLine = br.readLine()) != null)
            {
                String[] inputArray = inputLine.split(",");
                dl.addLast(inputArray[0],
                           inputArray[1],
                           inputArray[2],
                           inputArray[3]);

            }

            br.close();

        } catch (FileNotFoundException ex) {
            System.out.println(ex.getMessage());
            return false;
        }catch (Exception e)
        {
            System.out.println(e.getMessage());
            return false;
        }


        return true;
    }


    public static void main(String[] args) {

        String searchID = "";
        String insertCity = "";

        String filePath = "c:\\z\\peoplePets.csv";

        addData(filePath);

        dl.displayList();

        System.out.println("\n----- Find Examples------\n");

        searchID = "10";
        idSearch(searchID);

        searchID = "AAA";
```

(continues on next page)

(continued from previous page)

```
        idSearch(searchID);

        System.out.println("\n----- Delete Examples------\n");

        searchID = "10";
        deleteID(searchID);

        searchID = "AA";
        deleteID(searchID);


        System.out.println("\n----- Insert Examples------\n");

        searchID = "140";
        dl.insertAfter(searchID, "1440","Duke","Ellington", "Phoenix");


        searchID = "145";
        dl.insertBefore(searchID, "1455","John","Coletrane", "Hydra");


        dl.displayList();

        System.out.println("\nbye");
    }
}

/* peoplePet.csv

id,firstname,lastname,pet
1,Wini,Gwinnett,Crowned hawk-eagle
2,Birgit,Tabb,Silver gull
3,Alfonso,Moyle,Ring-tailed possum
4,Ode,Buckwell,Leopard
5,Francene,Zanazzi,Wambenger
6,Willie,Hakking,Butterfly
7,Carlie,Pizey,Northern fur seal
8,Cobby,Chittock,Great white pelican
9,Ingamar,Cardenoza,Puma
10,Brady,Vowles,Red meerkat
11,Nonna,Betser,Lesser mouse lemur
12,Antonina,Dovey,Insect
13,Erastus,Crackett,Cow
14,Lucien,Cardenosa,Yellow baboon
15,Leon,Storm,Blue catfish
16,Reinaldos,Welberry,Common grenadier
17,Cherice,Coleson,Shrike
18,Elyn,Antill,Echidna
19,Dicky,Guppie,Puna ibis
20,Erasmus,Pauncefort,White-necked stork
21,Stewart,Pettifer,Starling
22,Brand,Tytcomb,Raccoon
23,Edwina,Cosens,Radiated tortoise
24,Martelle,Barkus,Cormorant
25,Blithe,Prevett,Kafue flats lechwe
26,Abbie,Ferber,Common wombat
```

(continues on next page)

(continued from previous page)

```
27,Antonin,Sayes,Small-clawed otter
28,Vaughan,Barzen,Hawk-eagle
29,Elwira,Braemer,Monkey
30,Sibelle,Vennings,Javanese cormorant
31,Mavra,Bulter,Lemur
32,Darrelle,Sanford,Beisa oryx
33,Rodney,Whapples,Gull
34,Horace,Gerwood,Tropical buckeye butterfly
35,Garvin,Pestell,Bleu
36,Lauralee,Crowdace,Dama wallaby
37,Farica,Juara,Penguin
38,Bucky,Taylo,Crab
39,Carlynne,Pleasaunce,Common genet
40,Dana,Percy,White-throated kingfisher
41,Esma,McKerley,Southern ground hornbill
42,Hube,Grills,Flamingo
43,Hersch,Schneidau,Kangaroo
44,Hildy,Matfin,Gull
45,Lelia,Donaghy,American marten
46,Eric,Tydd,Bahama pintail
47,Clim,Tetsall,Spur-winged goose
48,Berke,Brotherwood,Little cormorant
49,Steve,Bride,Turtle
50,Christiane,Stoppe,Fox
51,Riley,Badgers,Swallow
52,Leonidas,Pughsley,Roseate cockatoo
53,Richard,Baudon,White-browed owl
54,Marline,Tousey,Indian mynah
55,Margarita,Breche,Phalarope
56,Gerek,Aspinwall,Great horned owl
57,Mabelle,Aronin,Grouse
58,Curtice,Provost,Indian mynah
59,Nikolaos,Cass,Desert tortoise
60,Edmund,Pogosian,Grenadier
61,Cindi,Vell,Catfish
62,Davis,Roberts,Three-banded plover
63,Clayborne,Jennrich,Roe deer
64,Malory,Iwanicki,Snowy owl
65,Toddy,Vannuchi,Arboral spiny rat
66,Terri,Dudson,Malachite kingfisher
67,Gabriel,Prine,Possum
68,Jeannine,Westwick,Gecko
69,Conney,Mattke,Stork
70,Eulalie,Wapplington,Flamingo
71,Darrick,Porcas,Rat
72,Matty,Marchment,Deer
73,Fedora,Semper,Nighthawk
74,Barnebas,Wychard,Flying fox
75,Edmund,Whitton,Squirrel
76,Oliver,Dragoe,Parrot
77,Carly,Royden,Lourie
78,Cairistiona,Brothwell,Chestnut weaver
79,Barnabas,Eastby,Lion
80,Clementina,McCoish,European stork
81,Rupert,Goosnell,Boa
82,Jerrylee,Keir,Lizard
```

(continues on next page)

(continued from previous page)

```
83,Cariotta,Strettell,Little blue penguin
84,Onida,Wysome,White-rumped vulture
85,Reggis,Thursby,Cape wild cat
86,Sharleen,Yele,Gazer
87,Obadias,Rosedale,Vine snake
88,Jodie,Harmond,Boat-billed heron
89,Jonie,Goodricke,Brush-tailed phascogale
90,Carissa,Clorley,Elk
91,Jacki,Belhome,Northern elephant seal
92,Virgina,Jarrette,House sparrow
93,Hasheem,Cordeiro,Oystercatcher
94,Roseann,Hussy,Fairy penguin
95,Emmeline,Saurat,Bee-eater
96,Ashlin,Ollerhead,Grey lourie
97,Frannie,Sailes,Jungle kangaroo
98,Shelden,Imason,Woodpecker
99,Nicole,Cattle,Starling
100,Bennie,Selliman,Ant
101,Monroe,Sturzaker,Wild turkey
102,Lou,Drew-Clifton,Ferruginous hawk
103,Gerladina,Broadbere,Yellow-billed stork
104,Laney,Scartifield,Common zebra
105,Rolfe,Dressel,Cape Barren goose
106,Clarita,Zylbermann,Red-legged pademelon
107,Ev,Buckston,Kite
108,Vidovic,Lawson,Buffalo
109,Daloris,Grzesiak,Southern brown bandicoot
110,Pier,Sproson,Brocket
111,Edwina,Barlace,Giant girdled lizard
112,Elvin,Birchwood,Deer
113,Jedediah,Lazonby,Wallaby
114,Ambur,Lochead,Gila monster
115,Mirabella,Ferron,Possum
116,Dougy,Gianinotti,Openbill
117,Cher,Sivill,Dog
118,Durante,Wissby,Genet
119,Rora,Shord,Dove
120,Dame,Jennison,Red-billed toucan
121,Ira,Karolowski,Eurasian red squirrel
122,Juliet,Hobson,Greylag goose
123,Natale,Tattersdill,Starling
124,Caitlin,Leggett,Ring-necked pheasant
125,Hamnet,Danelut,South African hedgehog
126,Christye,Stores,Porcupine
127,Vince,Paolo,Kangaroo
128,Cristy,Fesby,Starling
129,Britney,Standfield,Burrowing owl
130,Kaleena,Volkers,Langur
131,Averil,Kimbell,Rhea
132,Carmelita,Gehrels,Spotted-tailed quoll
133,Cory,Sreenan,Hyrax
134,Sybille,Filippone,Dove
135,Delia,Forkan,Agama lizard
136,Sigfried,Cattlemull,Eastern dwarf mongoose
137,Rowena,James,Fox
138,Shane,Naisey,Jaguarundi
```

(continues on next page)

(continued from previous page)

```
139,Janine,Ielden,Eastern fox squirrel
140,Beverlie,Biggerstaff,Fox
141,Menard,Archbould,Roe deer
142,Darline,Keating,Caribou
143,Heall,Chritchley,Beaver
144,Corenda,Bunnell,Glossy starling
145,Town,Mandal,White-winged tern
146,Pietrek,Primmer,Superb starling
147,Guillemette,Jasper,Gemsbok
148,Diane,Mewes,Civet
149,Hatty,Liddle,Grouse
150,Ange,Beardwell,Deer

 */
```

End Of Topic

CHAPTER

THIRTYFOUR

STACKS AND QUEUES

34.1 Key Ideas

- Stacks
- Queues

34.2 Discussion

Stacks and queues are a way of organizing data and consuming data. The data can be stored in an array or linked list.

34.2.1 Stack

Definition

A **Stack** is a way of organizing and consuming data where the Last In is the First Out (LIFO). Or it can be thought of as First In is the Last Out (FILO).

Most people think of a stack as a stack of plates, where each plate represents data. Data is ***pushed*** onto the stack. Meaning. it is added to the array or linked list. Data is consumed by ***popping*** it off of the stack. Meaning, that the last piece of data added to the stack is removed from the array or linked list. This data is usually used by the part of the program that is popping/removing it from the stack. ***Peeking*** is looking at the last/next piece of data, but *not* removing it from the stack. An example is *Stack Array*

Methods usually associated with a **stack** are as follows:

- **Push** - adds data to the stack
- **Pop** - removes data from the stack
- **Peek** - retrieves the next piece/top data from the stack, but does not remove it.
- **isFull** - this is used when making a stack with an array since an array has a limited number of elements.
- **isEmpty** - this is used to determine if the stack is empty.

34.2.2 Queue

Definition

A **Queue** is a way of organizing and consuming data where the First In is the First Out (FIFO).

Most people think of a queue as people standing in a line, where people represent data. Data is consumed in the order in which it was added.

Methods usually associated with a **queue** are as follows:

- **Enqueue** - add data to the queue.
- **Dequeue** - remove data from the queue
- **Peek** - retrieves the next piece/"head" data from the queue, but does not remove it.
- **isFull** - this is used when making a queue with an array since an array has a limited number of elements.
- **isEmpty** - this is used to determine if the queue is empty.

34.2.3 Priority Queue

Definition

A **Priority Queue** is a queue where there is a mechanism to place data at the start or infront of other data.

End Of Topic

34.3 Stack - Using Array of Objects

34.3.1 Stack Methods

Methods usually associated with a **stack** are as follows:

- **Push** - adds data to the stack
- **Pop** - removes data from the stack
- **Peek** - retrieves the next piece/top data from the stack, but does not remove it.
- **isFull** - this is used when making a stack with an array since an array has a limited number of elements.
- **isEmpty** - this is used to determine if the stack is empty.

Tip: isFull is not needed if a Stack is created using a Linked List

Details discussed *here*

34.3.2 Lecture Code

```java
/*
 * Project: Stack Using Array of Objects
 * Programmer: James Goudy
 * DS132SU_StackArray
 *
 *
 * Stack
 * push
 * pop
 * peek
 * isEmpty
 * isFull
 *
 *
 */

class Town {

    public String city;
    public int population;

    // constructor
    public Town(String city, int population) {
        this.city = city;
        this.population = population;
    }

    public void displayCity() {
        System.out.print("{" + city + ", " + population + "} ");
    }

}

class Stack {

    private int maxSize;
    private Town[] stackArray;
    private int top = -1;

    //constructor
    public Stack(int maxSize) {
        this.maxSize = maxSize;
        stackArray = new Town[maxSize];
        top = -1;
    }

    public boolean push(String city, int population) {

        // add data to the stack
        if (isFull()) {
            return false;
        } else {
            Town theTown = new Town(city, population);
            stackArray[++top] = theTown;
            return true;
```

(continues on next page)

(continued from previous page)

```
        }

    }

    public Town pop() {
        // remove data from the stack
        return stackArray[top--];
    }

    public Town peek() {
        // look/peek at the top of the stack
        return stackArray[top];
    }

    public boolean isEmpty() {
        //check if the array is empty
        return (top == -1);
    }

    public boolean isFull() {
        // check if the array is full
        return (top == maxSize - 1);
    }

}

public class DS132SU_StackArray {

    public static void main(String[] args) {

        // create a stack
        Stack myStack = new Stack(10);
        Town tempTown = null;

        // add data to the stack
        myStack.push("Kali", 300000);
        myStack.push("Bozeman", 100000);
        myStack.push("Whitefish", 40000);
        myStack.push("Columbia Falls", 30000);

        // peek at the top data
        System.out.print("Peek - ");
        myStack.peek().displayCity();
        System.out.println("");

        // pop one data object from the stack and store it in an object
        tempTown = myStack.pop();
        tempTown.displayCity();

        // empty the list
        while (!myStack.isEmpty()) {
            myStack.pop().displayCity();
        }

        System.out.println("");
```

(continues on next page)

(continued from previous page)

```
        //ternary operator
        boolean flag;
        flag = myStack.push("Plains", 15000) ? true : false;

        if (flag) {
            System.out.println("Item added");
        } else {
            System.out.println("Item NOT added");
        }

        System.out.print("\nbye");
    }
}
```

End of Topic

34.4 Queue - Using Array Of Objects

34.4.1 Queue Methods

Methods usually associated with a **queue** are as follows:

- **Enqueue** - add data to the queue.
- **Dequeue** - remove data from the queue
- **Peek** - retrieves the next piece/"head" data from the queue, but does not remove it.
- **isFull** - this is used when making a queue with an array since an array has a limited number of elements.
- **isEmpty** - this is used to determine if the queue is empty.

Tip: isFull is not needed if a Stack is created using a Linked List

Check This Out!

// make a new queue object

Queue theQue = new Queue(5);

/* NOTICE THE CODE BELOW - make an object from the sub class */

Queue.Town theTown = theQue.new Town("", 0);

Details discussed *here*

34.4.2 Lecture Code

```java
/*
 *Programmer: James Goudy
 *Project: Queue of Objects
 *
 */
package com.mycompany.ds132su_queue;

/**
 *
 * @author jgoudy
 */
class Queue {

    class Town {
        // subclass
        public String city;
        public int population;

        public Town(String city, int population) {
            this.city = city;
            this.population = population;
        }

        public void displayTown() {
            System.out.print("{" + city + " - " + population + "} ");
        }

    }// end of town

    private int maxSize;
    private Town[] queArray;
    private int numItems;
    private int head;
    private int tail;

    // Queue Constructor
    public Queue(int maxSize) {
        this.maxSize = maxSize;
        queArray = new Town[maxSize];
        head = 0;
        tail = -1;
        numItems = 0;
    }

    public boolean isFull() {
        return (numItems == maxSize);
    }

    public boolean isEmpty() {
        return (numItems == 0);
    }

    public boolean enqueue(String city, int population) {

        // insert at tail
```

(continues on next page)

(continued from previous page)

```
        // "enqueue"
        if (isFull()) {
            return false;
        }

        // create town
        Town newTown = new Town(city, population);

        // wrap to front if neccessary
        if (tail == maxSize - 1) {
            tail = -1;
        }

        // add object to array
        queArray[++tail] = newTown;

        // increment the count of objects in the array
        numItems++;

        return true;
    }

    public Town dequeue() {
        // remove from the front - the head
        // "dequeue"

        if (isEmpty()) {
            System.out.print("Queue is empty");
            return null;
        }

        // retreive the next in queue and move the head to
        // the next data item.
        Town temp = queArray[head++];

        if (head == maxSize) {
            head = 0;
        }

        // decrease the count of objects in the array
        numItems--;

        return temp;
    }

    public Town peek() {
        // retreive the head information but do not remove
        return queArray[head];
    }

    public void displayQueue() {
        int cntr = 0;
        int pos = head;
        while (cntr < numItems) {
            queArray[pos].displayTown();
            cntr++;
```

(continues on next page)

(continued from previous page)

```
            pos++;
            // loop to the start of the array if necessary
            if (pos == maxSize) {
                pos = 0;
            }

        }
        System.out.println("\n");
    }

} // end of queue

public class DS132SU_Queue {

    public static void main(String[] args) {

        // make a new queue object
        Queue theQue = new Queue(5);

        // make an object from the sub class
        Queue.Town theTown = theQue.new Town("", 0);

        String acity;

        theQue.enqueue("Bozeman", 100000);
        theQue.enqueue("Culver", 100001);
        theQue.enqueue("Dover", 100002);
        theQue.enqueue("Edger", 100003);

        System.out.println("\n------ Queue--------\n");

        theQue.displayQueue();

        System.out.println("\nDequeue the first two items in the queue");

        // return next item from the queue
        theTown = theQue.dequeue();
        System.out.println("\n" + theTown.city + " - " + theTown.population);

        theTown = theQue.dequeue();
        System.out.println("\n" + theTown.city + " - " + theTown.population);

        System.out.println("\n--------------\n");

        System.out.println("\nDequeue and retreive only the city");

        // another option to dequeue and retrieve city only
        acity = theQue.dequeue().city;
        System.out.println(acity);

        System.out.println("\n------ Queue--------\n");

        theQue.displayQueue();

        theQue.enqueue("Franklin", 104);
```

(continues on next page)

(continued from previous page)

```
        theQue.enqueue("Georgetown", 105);
        theQue.enqueue("Highland", 106);

        System.out.println("\n------- Queue -------\n");
        theQue.displayQueue();

        System.out.println("\n----- Peek just city ---------\n");

        acity = theQue.peek().city;
        System.out.println(acity);

        System.out.println("\n----- Peek city population ---------\n");

        theTown = theQue.peek();
        System.out.println("\n" + theTown.city + " - " + theTown.population);

        System.out.println("\n------ Queue --------\n");
        // notice that peek did not remove any towns
        theQue.displayQueue();

    }
}

/*
 Output

------ Queue--------

{Bozeman - 100000} {Culver - 100001} {Dover - 100002} {Edger - 100003}


Dequeue the first two items in the queue

Bozeman - 100000

Culver - 100001

--------------


Dequeue and retreive only the city
Dover

------ Queue--------

{Edger - 100003}


------- Queue -------

{Edger - 100003} {Franklin - 104} {Georgetown - 105} {Highland - 106}


----- Peek just city ---------

Edger
```

(continues on next page)

(continued from previous page)

```
----- Peek city population ---------


Edger - 100003

------ Queue --------

{Edger - 100003} {Franklin - 104} {Georgetown - 105} {Highland - 106}



 */
```

End Of Topic

34.5 Priority Queue

34.5.1 Key Ideas

- Priority Queue

Definition

A **Priority Queue** is a queue where there is a mechanism to place data at the start or in front of other data.

34.5.2 Lecture Code

```
// priorityQ.java
// demonstrates priority queue

/**
 *
 * @author jgoudy
 */
class PriorityQ {

    private int maxSize;
    private int[] queArr;
    private int nItems;

    // constructor
    public PriorityQ(int maxSize) {
        this.maxSize = maxSize;
        queArr = new int[maxSize];
        nItems = 0;
    }
```

(continues on next page)

(continued from previous page)

```
    public boolean isFull() {
        return (nItems == maxSize);
    }

    public boolean isEmpty() {
        return (nItems == 0);
    }

    // enqueue - add to queue
    // lower numbers take a higher place in the queue

    public void enqueue(int key) {
        int c;

        if (isEmpty()) {
            queArr[nItems++] = key;
        } else {
            for (c = nItems - 1; c >= 0; c--) {
                if (key > queArr[c]) {
                    queArr[c + 1] = queArr[c];
                } else {
                    break;
                }
            }// end of for

            queArr[c + 1] = key;

            nItems++;
        }
    }

    // retreive the next item from the queue
    // and remove it from the queue
    public int dequeue() {
        return queArr[--nItems];
    }

    // reteive the data from the next item in the queue
    // but does NOT remove it
    public int peek() {
        return queArr[nItems - 1];
    }

    // print the queue
    public void printPriorityQ() {
        System.out.println();
        for (int c = 0; c < nItems; c++) {
            System.out.print(queArr[c] + " ");
        }
        System.out.println();

    }

} // end of class

public class Ds_priorityQueue {
```

(continues on next page)

(continued from previous page)

```
    public static void main(String[] args) {

// assumption lower the number - higher the priority
        PriorityQ thePQ = new PriorityQ(10);

        // add data items to the queue
        thePQ.enqueue(30);
        thePQ.enqueue(50);
        thePQ.enqueue(10);
        thePQ.enqueue(40);
        thePQ.enqueue(20);
        thePQ.enqueue(60);
        thePQ.enqueue(5);

        // print the queue
        thePQ.printPriorityQ();

        System.out.println("\n-----------\n");

        // peek at the next data item
        System.out.println("Peek " + thePQ.peek());

        System.out.println("\n-----------\n");

        // remove all the items from the queue
        while (!thePQ.isEmpty()) {
            int x = thePQ.dequeue();
            System.out.print(x + " ");
        }

        System.out.println("\n-----------\n");

    }
}

/* Output

 60 50 40 30 20 10 5

-----------

Peek 5

-----------

5 10 20 30 40 50 60
-----------

 */
```

End Of Topic

CHAPTER THIRTYFIVE

SORTING ALGORITHMS

35.1 Visualizations

Visual Aglo

Toptal

Comparison Sorting Algorithms

Sort Visualizer

End Of Topic

35.2 Bubble Sort

35.2.1 Key Ideas

- Bubble Sort

Definition

Bubble Sort is a simple sorting algorithm that repeatedly iterates through the list or array. It compares adjacent elements and swaps them if they are in the wrong order. The algorithm repeatedly passes through the list until the list is sorted. It is a comparison sort and is named for the way smaller or larger elements “bubble” to the top of the list.

35.3 Performance

Bubble sort has a worst-case and average complexity of On^2^.

35.3.1 Visualizations

Visual Aglo

Toptal

Comparison Sorting Algorithms

Sort Visualizer

35.3.2 Videos

35.3.3 Lecture Code

```java
/*
 * Programmer: James Goudy
 * Project: Bubble Sort
 */
package com.mycompany.bubblesort_lecturecode;

import java.util.Random;

class BubbleSort {

    // arrInt is an array of integers
    // numDataElements is the actual count
    // of elements of data in the array
    // algorithm assumes the data is contiguous
    int arrInt[];
    int numDataElments;

    public BubbleSort(int[] arrInt, int numDataElments) {
        this.arrInt = arrInt;
        this.numDataElments = numDataElments;
    }

    public void Sort() {
        //
        int n = numDataElments;

        for (int c = 0; c < n; c++) {
            for (int j = 1; j < (n); j++) {

                // check if the left element is
                // greater to the one on the right
                // "Bubble" the lowest to the left

                if (arrInt[j - 1] > arrInt[j]) {
                    // swap left element arr[j-1]
                    // with the one on the right and arr[j]

                    // store left one in temp
                    int temp = arrInt[j - 1];
                    //copy the right into the left
                    arrInt[j - 1] = arrInt[j];
                    //copy the left into the right
```

(continues on next page)

(continued from previous page)

```
                    arrInt[j] = temp;
                }
            }
        }
    }

}

public class BubbleSort_LectureCode {

    static int arrSize = 8;
    //static int theArray[] = new int[arrSize];
    static int theArray[] ={3,60,35,2,45,320,5,1};

    static void fillTheArray()
    {
        Random RNG  = new Random();
        for(int c = 0; c < arrSize; c++)
        {
            theArray[c] = RNG.nextInt(0,(arrSize*10));
        }
    }

    static void printArray(int anArray[], int numOfDataElements)
    {
        System.out.println("");
        for (int i = 0; i < numOfDataElements; i++) {
            System.out.print(anArray[i] + " ");
        }
        System.out.println("\n--------------\n");

    }

    public static void main(String[] args) {

        // option to randomly fill the array
        //fillTheArray();

        printArray(theArray, arrSize);

        BubbleSort bs = new BubbleSort(theArray, arrSize);
        bs.Sort();

        printArray(theArray, arrSize);


    }
}

/*
3 60 35 2 45 320 5 1
--------------



1 2 3 5 35 45 60 320
--------------
*/
```

End Of Topic

35.4 End Of Section

Algorithms and Data Structures

End Of Section