Toby Gates – Computer Programming 1

Logbook for Computer Programming

Table of Contents

Term 1

1 – Introduction

1.1 – Quick points on Java

A Java program consists of a number of classes, which contain methods, which contain statements.

A Java application must have a main method.

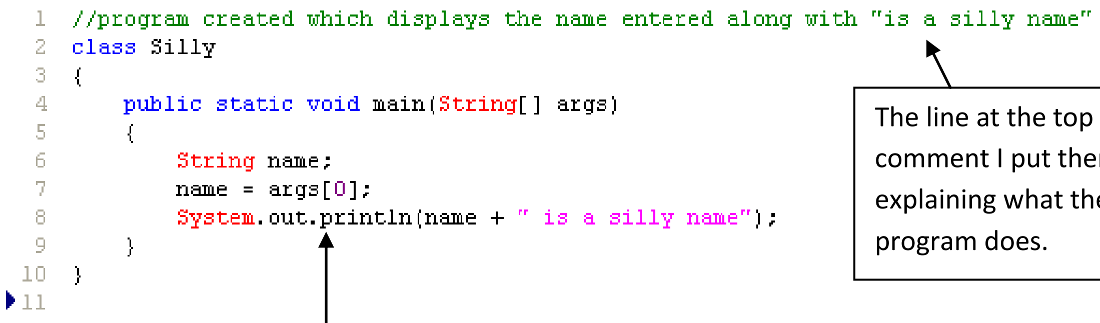
1.2 – Types of Brackets used in Java

{ } = Used to group sections of Java code

[ ] = Used for Arrays

( ) = Used for expressions or functions

1.3 "Silly.java" Program



```
//program created which displays the name entered along with "is a silly name"
class Silly
{
    public static void main(String[] args)
    {
        String name;
        name = args[0];
        System.out.println(name + " is a silly name");
    }
}
```

The line at the top is a comment I put there explaining what the java program does.

Above is a screenshot I have taken of the program Silly.java.

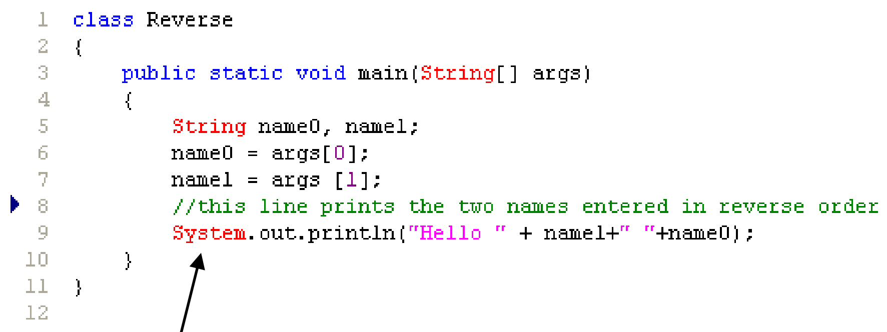
It shows how to output and display a name entered with another message after it, in this case the message is " is a silly name".

When the program is run and the name Bob is entered, the output displayed would be "Bob is a silly name."

1.4 “Reverse.java” Program V1 (Two Names)

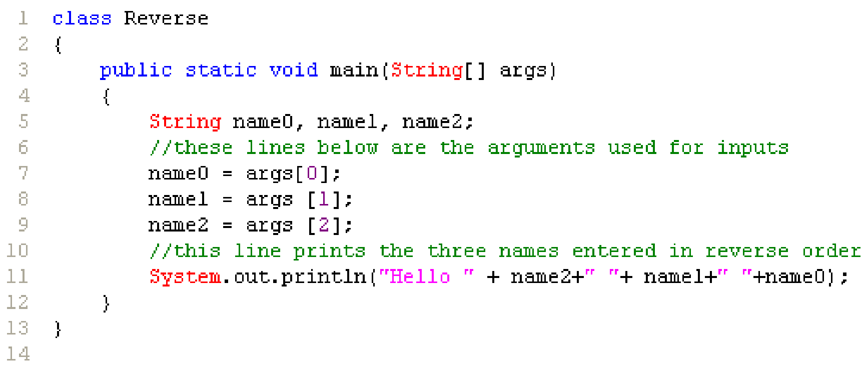
To the left is a screenshot of the java program “Reverse”.

The line at the bottom (the one that starts with **System.out.println**) outputs the 2 names entered in Reverse order.



```java
class Reverse
{
    public static void main(String[] args)
    {
        String name0, name1;
        name0 = args[0];
        name1 = args [1];
        //this line prints the two names entered in reverse order
        System.out.println("Hello " + name1+" "+name0);
    }
}
```

1.5 “Reverse.java” Program V2 (Three Names)



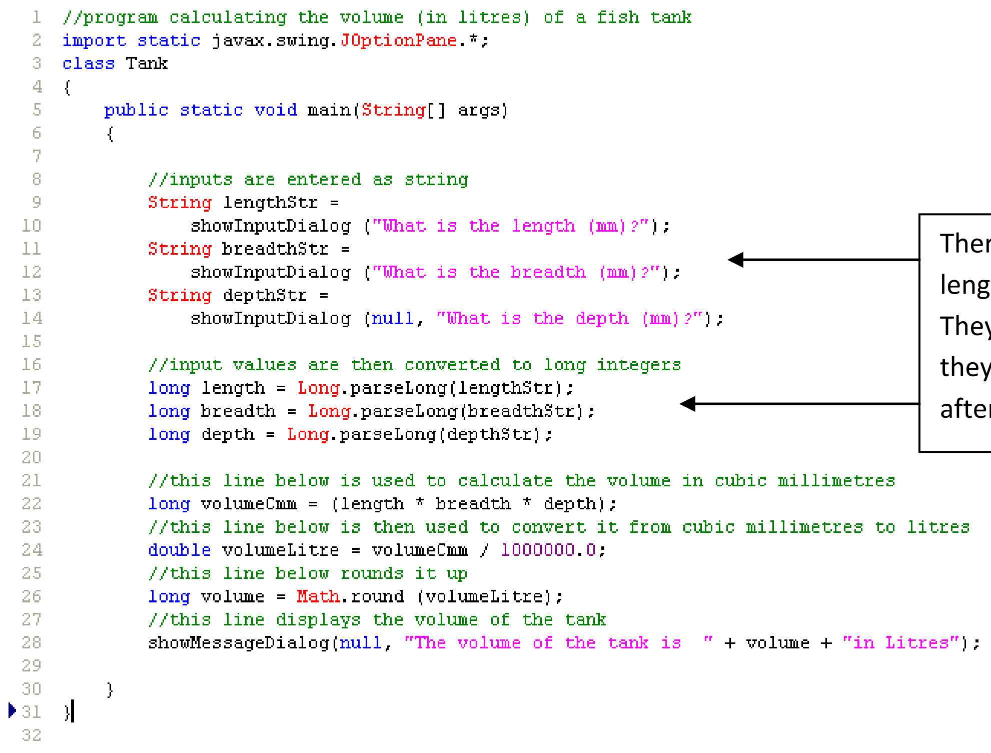
```java
class Reverse
{
    public static void main(String[] args)
    {
        String name0, name1, name2;
        //these lines below are the arguments used for inputs
        name0 = args[0];
        name1 = args [1];
        name2 = args [2];
        //this line prints the three names entered in reverse order
        System.out.println("Hello " + name2+" "+ name1+" "+name0);
    }
}
```

To the right is a quick screenshot of an updated version of the java program “Reverse”.

However, this version of Reverse outputs 3 names in reverse order, instead of just two.

2 – Variables and IO

2.1 "Tank.java" Program

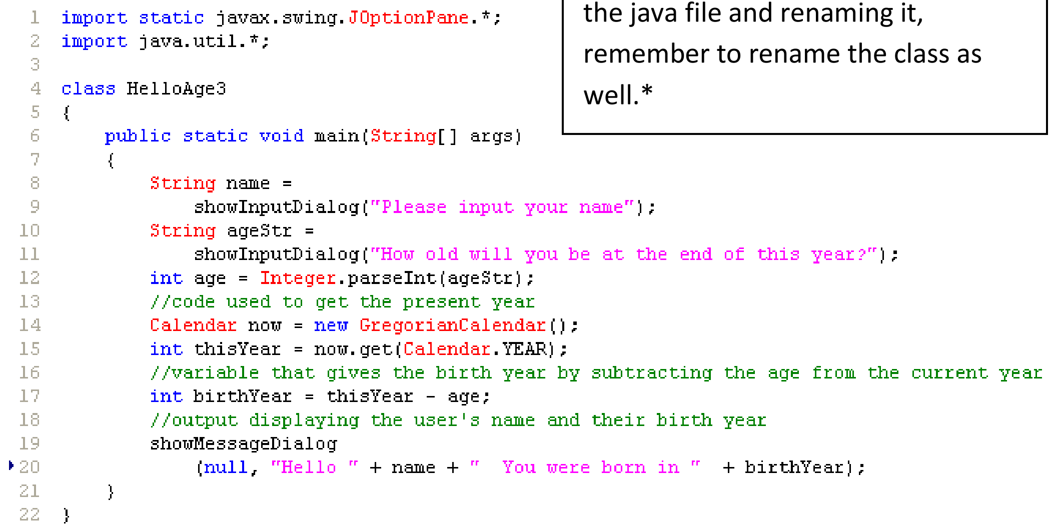


```java
//program calculating the volume (in litres) of a fish tank
import static javax.swing.JOptionPane.*;
class Tank
{
    public static void main(String[] args)
    {

        //inputs are entered as string
        String lengthStr =
            showInputDialog ("What is the length (mm)?");
        String breadthStr =
            showInputDialog ("What is the breadth (mm)?");
        String depthStr =
            showInputDialog (null, "What is the depth (mm)?");

        //input values are then converted to long integers
        long length = Long.parseLong(lengthStr);
        long breadth = Long.parseLong(breadthStr);
        long depth = Long.parseLong(depthStr);

        //this line below is used to calculate the volume in cubic millimetres
        long volumeCmm = (length * breadth * depth);
        //this line below is then used to convert it from cubic millimetres to litres
        double volumeLitre = volumeCmm / 1000000.0;
        //this line below rounds it up
        long volume = Math.round (volumeLitre);
        //this line displays the volume of the tank
        showMessageDialog(null, "The volume of the tank is  " + volume + "in Litres");

    }
}

```

There are the inputs for the length, the breadth and the depth. They are classed as string now but they are changed later to integer, after the inputs have been input.

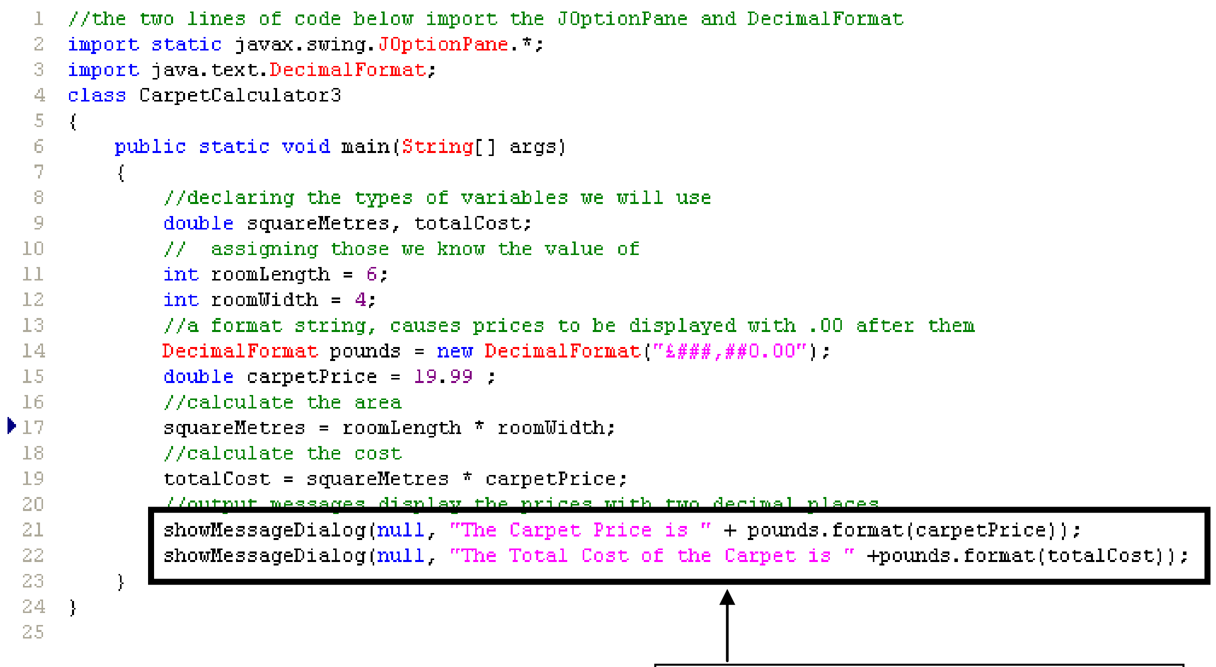
2.2 "HelloAge3.java" Program

When saving an updated version of the java file and renaming it, remember to rename the class as well.



```java
import static javax.swing.JOptionPane.*;
import java.util.*;

class HelloAge3
{
    public static void main(String[] args)
    {
        String name =
            showInputDialog("Please input your name");
        String ageStr =
            showInputDialog("How old will you be at the end of this year?");
        int age = Integer.parseInt(ageStr);
        //code used to get the present year
        Calendar now = new GregorianCalendar();
        int thisYear = now.get(Calendar.YEAR);
        //variable that gives the birth year by subtracting the age from the current year
        int birthYear = thisYear - age;
        //output displaying the user's name and their birth year
        showMessageDialog
            (null, "Hello " + name + "  You were born in "  + birthYear);
    }
}
```

2.3 “CarpetCalculator3.java” Program



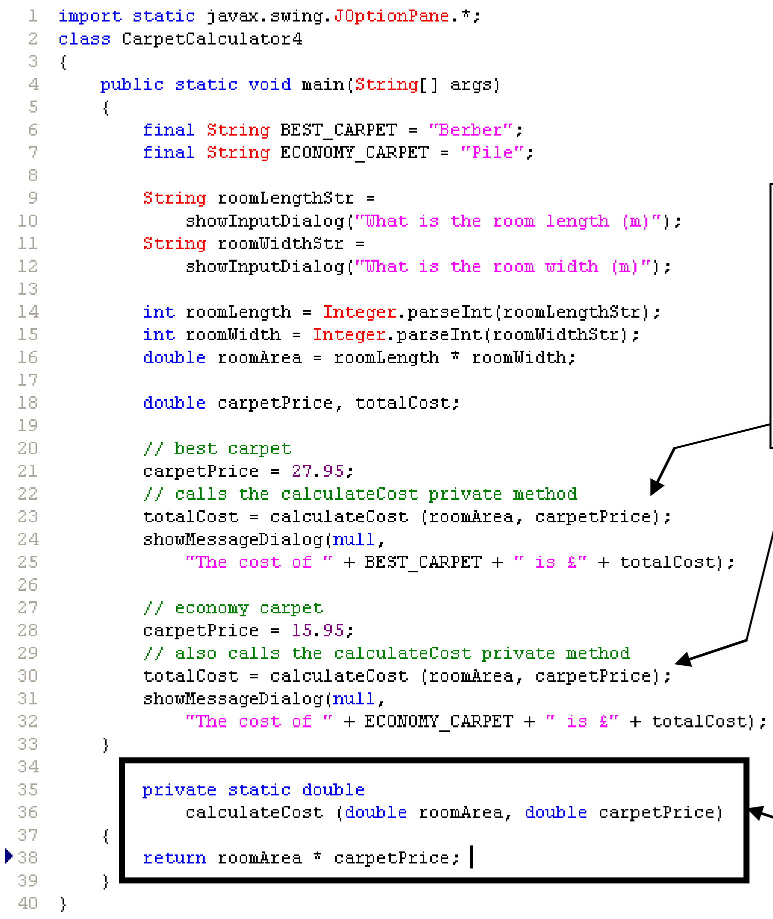
```java
//the two lines of code below import the JOptionPane and DecimalFormat
import static javax.swing.JOptionPane.*;
import java.text.DecimalFormat;
class CarpetCalculator3
{
    public static void main(String[] args)
    {
        //declaring the types of variables we will use
        double squareMetres, totalCost;
        //  assigning those we know the value of
        int roomLength = 6;
        int roomWidth = 4;
        //a format string, causes prices to be displayed with .00 after them
        DecimalFormat pounds = new DecimalFormat("£###,##0.00");
        double carpetPrice = 19.99 ;
        //calculate the area
        squareMetres = roomLength * roomWidth;
        //calculate the cost
        totalCost = squareMetres * carpetPrice;
        //output messages display the prices with two decimal places
        showMessageDialog(null, "The Carpet Price is " + pounds.format(carpetPrice));
        showMessageDialog(null, "The Total Cost of the Carpet is " +pounds.format(totalCost));
    }
}
```

I added the line for DecimalFormat just above the carpet price in the program, the comments there explain what it does.

The outputs use JOptionPanes and display the outputs by saying “The Carpet Price is” or “The Total Cost of the Carpet is” and then +pounds.format(carpetPrice)); or +pounds.format(totalCost));

3 – Methods

3.1 “CarpetCalculator4.java” Program

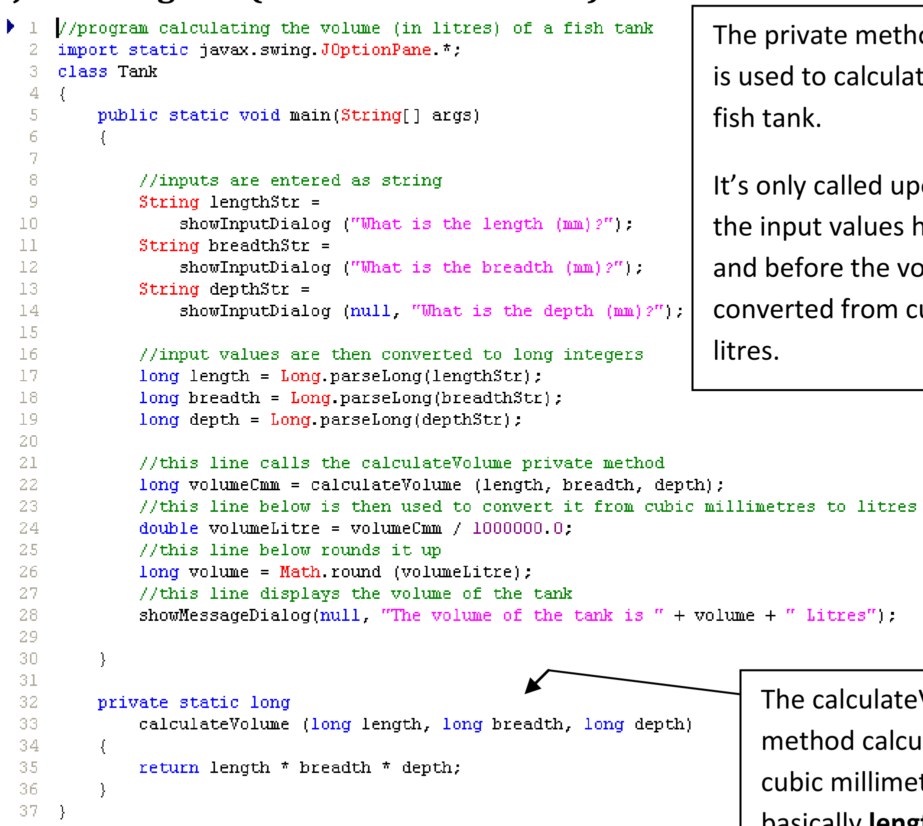


```java
import static javax.swing.JOptionPane.*;
class CarpetCalculator4
{
    public static void main(String[] args)
    {
        final String BEST_CARPET = "Berber";
        final String ECONOMY_CARPET = "Pile";

        String roomLengthStr =
            showInputDialog("What is the room length (m)");
        String roomWidthStr =
            showInputDialog("What is the room width (m)");

        int roomLength = Integer.parseInt(roomLengthStr);
        int roomWidth = Integer.parseInt(roomWidthStr);
        double roomArea = roomLength * roomWidth;

        double carpetPrice, totalCost;

        // best carpet
        carpetPrice = 27.95;
        // calls the calculateCost private method
        totalCost = calculateCost (roomArea, carpetPrice);
        showMessageDialog(null,
            "The cost of " + BEST_CARPET + " is £" + totalCost);

        // economy carpet
        carpetPrice = 15.95;
        // also calls the calculateCost private method
        totalCost = calculateCost (roomArea, carpetPrice);
        showMessageDialog(null,
            "The cost of " + ECONOMY_CARPET + " is £" + totalCost);
    }

        private static double
            calculateCost (double roomArea, double carpetPrice)
    {
        return roomArea * carpetPrice;
    }
}
```

CalculateCost is called upon twice in the program, for the total cost of both carpets. This is because both carpets use the same equation of: **roomArea * carpetPrice**.

This section of the code is the CalculateCost private method that’s called upon twice in the program.

3.2 "Tank.java" Program (with Private Method)



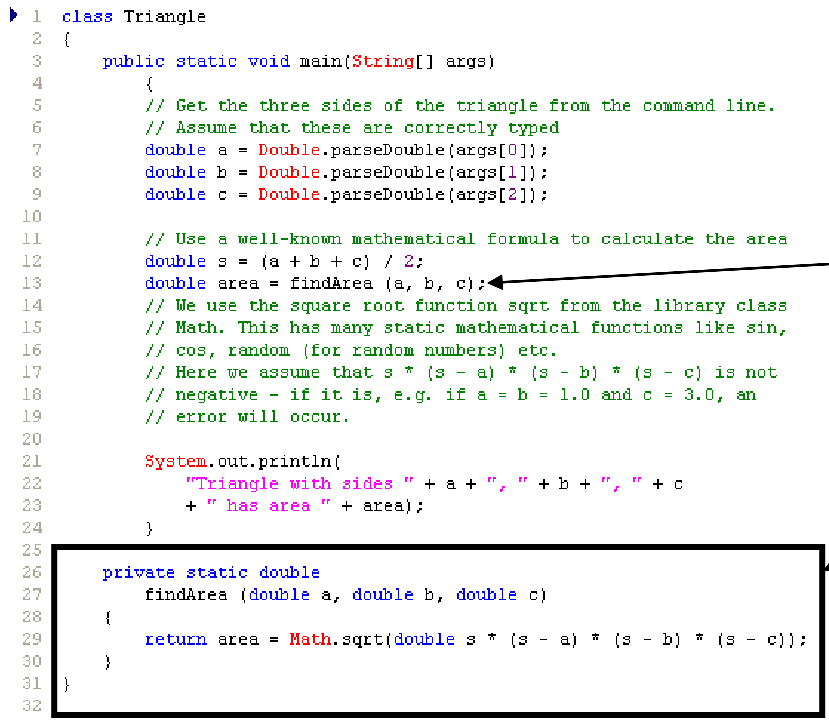
```java
//program calculating the volume (in litres) of a fish tank
import static javax.swing.JOptionPane.*;
class Tank
{
    public static void main(String[] args)
    {

        //inputs are entered as string
        String lengthStr =
            showInputDialog ("What is the length (mm)?");
        String breadthStr =
            showInputDialog ("What is the breadth (mm)?");
        String depthStr =
            showInputDialog (null, "What is the depth (mm)?");

        //input values are then converted to long integers
        long length = Long.parseLong(lengthStr);
        long breadth = Long.parseLong(breadthStr);
        long depth = Long.parseLong(depthStr);

        //this line calls the calculateVolume private method
        long volumeCmm = calculateVolume (length, breadth, depth);
        //this line below is then used to convert it from cubic millimetres to litres
        double volumeLitre = volumeCmm / 1000000.0;
        //this line below rounds it up
        long volume = Math.round (volumeLitre);
        //this line displays the volume of the tank
        showMessageDialog(null, "The volume of the tank is " + volume + " Litres");

    }

    private static long
        calculateVolume (long length, long breadth, long depth)
    {
        return length * breadth * depth;
    }
}
```

The private method used in this program is used to calculate the volume of the fish tank.

It's only called upon once, which is after the input values have been converted and before the volume calculated is converted from cubic millimetres to litres.

The calculateVolume private method calculates the volume in cubic millimetres, which is basically **length * breadth * depth**.

3.3 "Triangle.java" Program (with added Private Method)



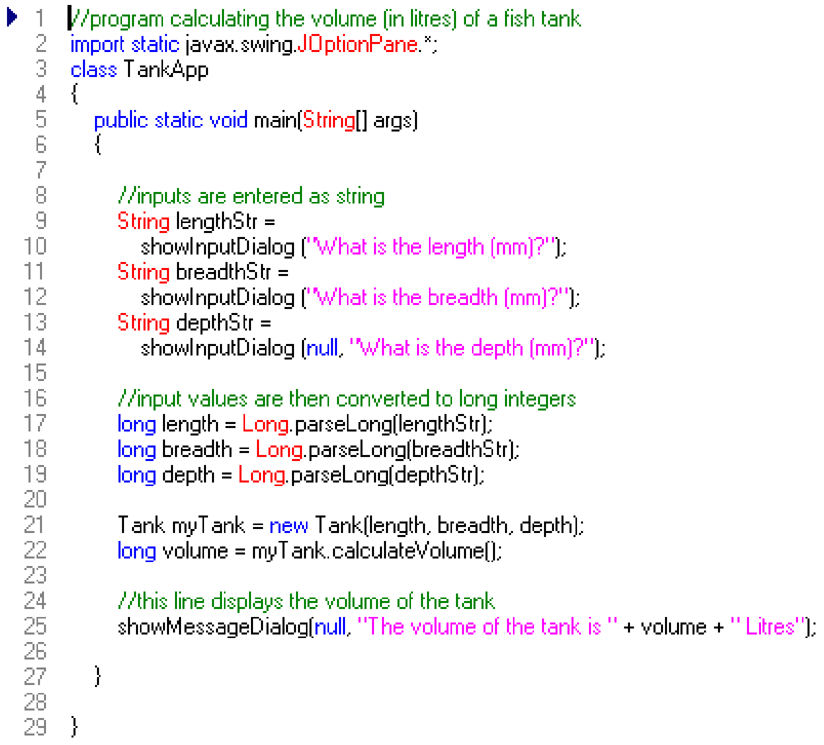
```
class Triangle
{
    public static void main(String[] args)
        {
        // Get the three sides of the triangle from the command line.
        // Assume that these are correctly typed
        double a = Double.parseDouble(args[0]);
        double b = Double.parseDouble(args[1]);
        double c = Double.parseDouble(args[2]);

        // Use a well-known mathematical formula to calculate the area
        double s = (a + b + c) / 2;
        double area = findArea (a, b, c);
        // We use the square root function sqrt from the library class
        // Math. This has many static mathematical functions like sin,
        // cos, random (for random numbers) etc.
        // Here we assume that s * (s - a) * (s - b) * (s - c) is not
        // negative - if it is, e.g. if a = b = 1.0 and c = 3.0, an
        // error will occur.

        System.out.println(
            "Triangle with sides " + a + ", " + b + ", " + c
            + " has area " + area);
        }

    private static double
        findArea (double a, double b, double c)
    {
        return area = Math.sqrt(double s * (s - a) * (s - b) * (s - c));
    }
}
```

This program's purpose is to work out the area of a triangle. The private method is called upon here in the code.

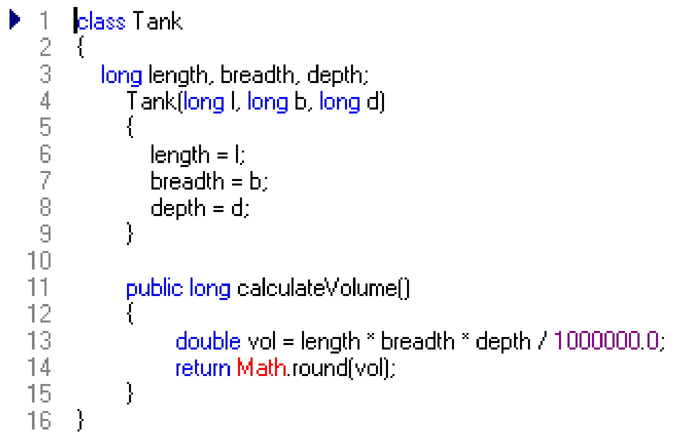
As you can see the private method performs one of the calculations of finding the area below.

4 – *Objects and Classes*

4.1 "TankApp.java / Tank.java" Program

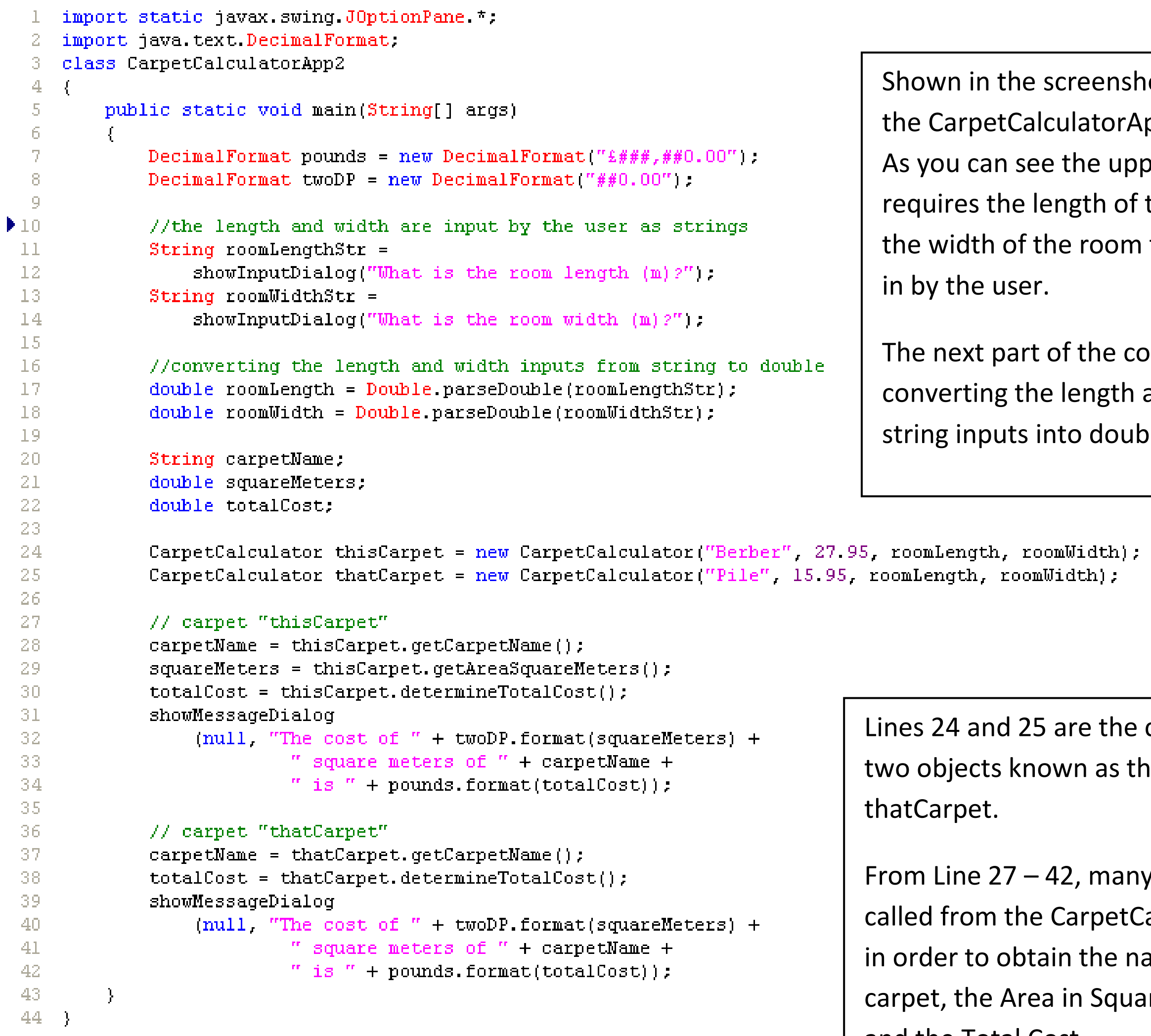


```
//program calculating the volume (in litres) of a fish tank
import static javax.swing.JOptionPane.*;
class TankApp
{
   public static void main(String[] args)
   {

      //inputs are entered as string
      String lengthStr =
         showInputDialog ("What is the length (mm)?");
      String breadthStr =
         showInputDialog ("What is the breadth (mm)?");
      String depthStr =
         showInputDialog (null, "What is the depth (mm)?");

      //input values are then converted to long integers
      long length = Long.parseLong(lengthStr);
      long breadth = Long.parseLong(breadthStr);
      long depth = Long.parseLong(depthStr);

      Tank myTank = new Tank(length, breadth, depth);
      long volume = myTank.calculateVolume();

      //this line displays the volume of the tank
      showMessageDialog(null, "The volume of the tank is " + volume + " Litres");

   }

}
```



```java
class Tank
{
    long length, breadth, depth;
        Tank(long l, long b, long d)
        {
            length = l;
            breadth = b;
            depth = d;
        }

        public long calculateVolume()
        {
            double vol = length * breadth * depth / 1000000.0;
            return Math.round(vol);
        }
}
```

4.2 "CarpetCalculatorApp2.java / CarpetCalculator.java"



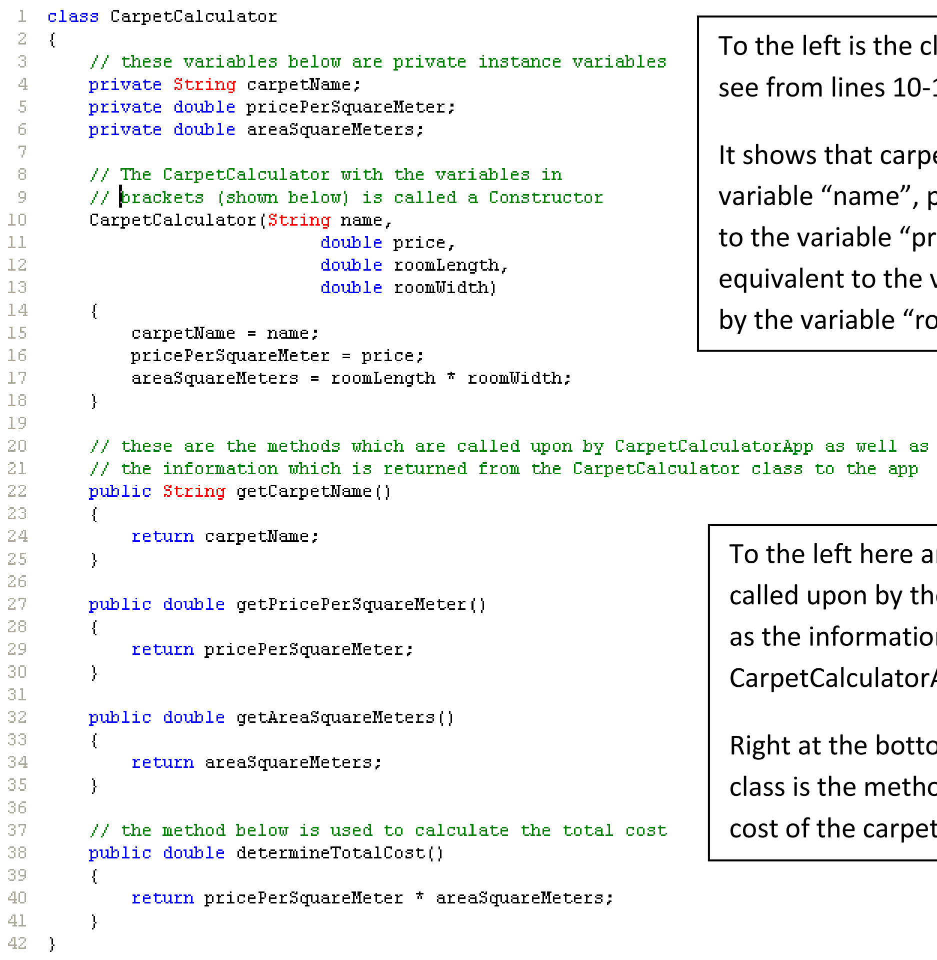
```java
 1  import static javax.swing.JOptionPane.*;
 2  import java.text.DecimalFormat;
 3  class CarpetCalculatorApp2
 4  {
 5      public static void main(String[] args)
 6      {
 7          DecimalFormat pounds = new DecimalFormat("£###,##0.00");
 8          DecimalFormat twoDP = new DecimalFormat("##0.00");
 9
10          //the length and width are input by the user as strings
11          String roomLengthStr =
12              showInputDialog("What is the room length (m)?");
13          String roomWidthStr =
14              showInputDialog("What is the room width (m)?");
15
16          //converting the length and width inputs from string to double
17          double roomLength = Double.parseDouble(roomLengthStr);
18          double roomWidth = Double.parseDouble(roomWidthStr);
19
20          String carpetName;
21          double squareMeters;
22          double totalCost;
23
24          CarpetCalculator thisCarpet = new CarpetCalculator("Berber", 27.95, roomLength, roomWidth);
25          CarpetCalculator thatCarpet = new CarpetCalculator("Pile", 15.95, roomLength, roomWidth);
26
27          // carpet "thisCarpet"
28          carpetName = thisCarpet.getCarpetName();
29          squareMeters = thisCarpet.getAreaSquareMeters();
30          totalCost = thisCarpet.determineTotalCost();
31          showMessageDialog
32              (null, "The cost of " + twoDP.format(squareMeters) +
33                      " square meters of " + carpetName +
34                      " is " + pounds.format(totalCost));
35
36          // carpet "thatCarpet"
37          carpetName = thatCarpet.getCarpetName();
38          totalCost = thatCarpet.determineTotalCost();
39          showMessageDialog
40              (null, "The cost of " + twoDP.format(squareMeters) +
41                      " square meters of " + carpetName +
42                      " is " + pounds.format(totalCost));
43      }
44  }
```

Shown in the screenshot to the left is the CarpetCalculatorApp2 in Edit Plus. As you can see the upper section requires the length of the room and the width of the room to be entered in by the user.

The next part of the code is converting the length and width string inputs into doubles

Lines 24 and 25 are the creation of two objects known as thisCarpet and thatCarpet.

From Line 27 – 42, many methods are called from the CarpetCalculator class in order to obtain the name of the carpet, the Area in Square Meters and the Total Cost.



```
1  class CarpetCalculator
2  {
3      // these variables below are private instance variables
4      private String carpetName;
5      private double pricePerSquareMeter;
6      private double areaSquareMeters;
7
8      // The CarpetCalculator with the variables in
9      // brackets (shown below) is called a Constructor
10     CarpetCalculator(String name,
11                      double price,
12                      double roomLength,
13                      double roomWidth)
14     {
15         carpetName = name;
16         pricePerSquareMeter = price;
17         areaSquareMeters = roomLength * roomWidth;
18     }
19
20     // these are the methods which are called upon by CarpetCalculatorApp as well as
21     // the information which is returned from the CarpetCalculator class to the app
22     public String getCarpetName()
23     {
24         return carpetName;
25     }
26
27     public double getPricePerSquareMeter()
28     {
29         return pricePerSquareMeter;
30     }
31
32     public double getAreaSquareMeters()
33     {
34         return areaSquareMeters;
35     }
36
37     // the method below is used to calculate the total cost
38     public double determineTotalCost()
39     {
40         return pricePerSquareMeter * areaSquareMeters;
41     }
42 }
```

To the left is the class CarpetCalculator, as you can see from lines 10-18 in the code is the Constructor.

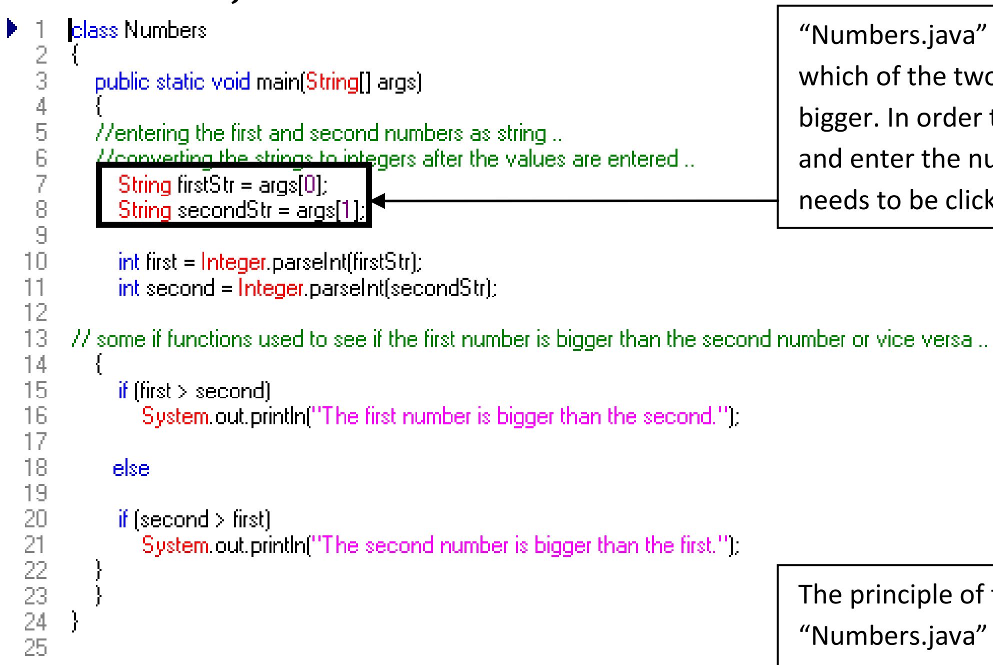
It shows that carpetName is equivalent to the variable “name”, pricePerSquareMeter is equivalent to the variable “price” and areaSquareMeters is equivalent to the variable “roomLength” multiplied by the variable “roomWidth”.

To the left here are all the methods that will be called upon by the CarpetCalculatorApp as well as the information that will be returned to the CarpetCalculatorApp.

Right at the bottom of the CarpetCalculator class is the method used to work out the total cost of the carpet.

5 – Decisions and Selections

5.1 “Numbers.java”

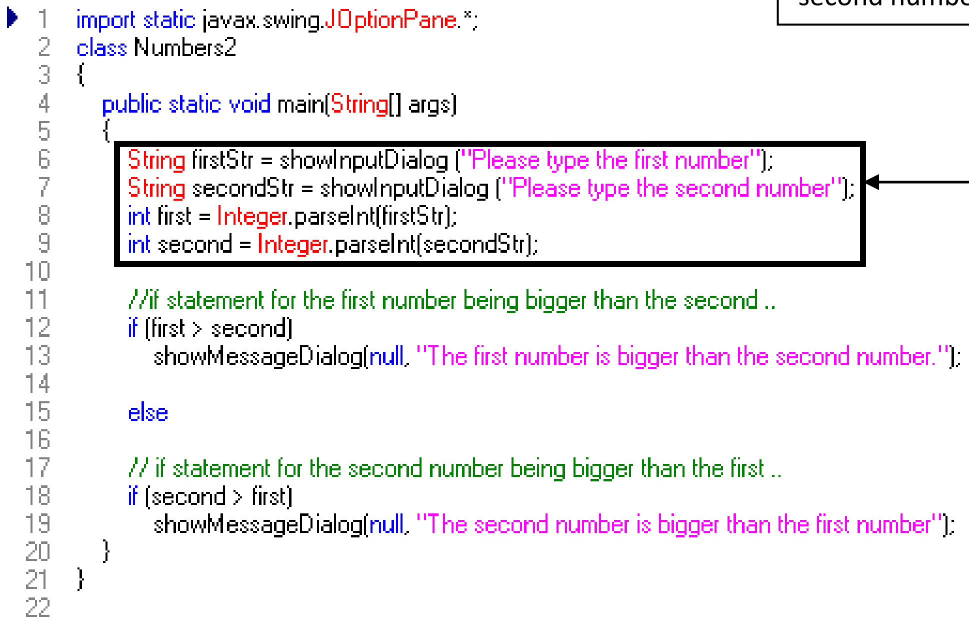


```java
class Numbers
{
   public static void main(String[] args)
   {
   //entering the first and second numbers as string ..
   //converting the strings to integers after the values are entered ..
      String firstStr = args[0];
      String secondStr = args[1];

      int first = Integer.parseInt(firstStr);
      int second = Integer.parseInt(secondStr);

// some if functions used to see if the first number is bigger than the second number or vice versa ..
   {
      if (first > second)
         System.out.println("The first number is bigger than the second.");

     else

      if (second > first)
         System.out.println("The second number is bigger than the first.");
   }
   }
}
```

“Numbers.java” uses arguments to see which of the two numbers entered is bigger. In order to run the program and enter the numbers, Java + Args needs to be clicked on NOT Java.

The principle of the programs “Numbers.java” and “Numbers2.java” is to display whether or not the first number entered is bigger than the second number and vice versa.

5.2 “Numbers2.java”

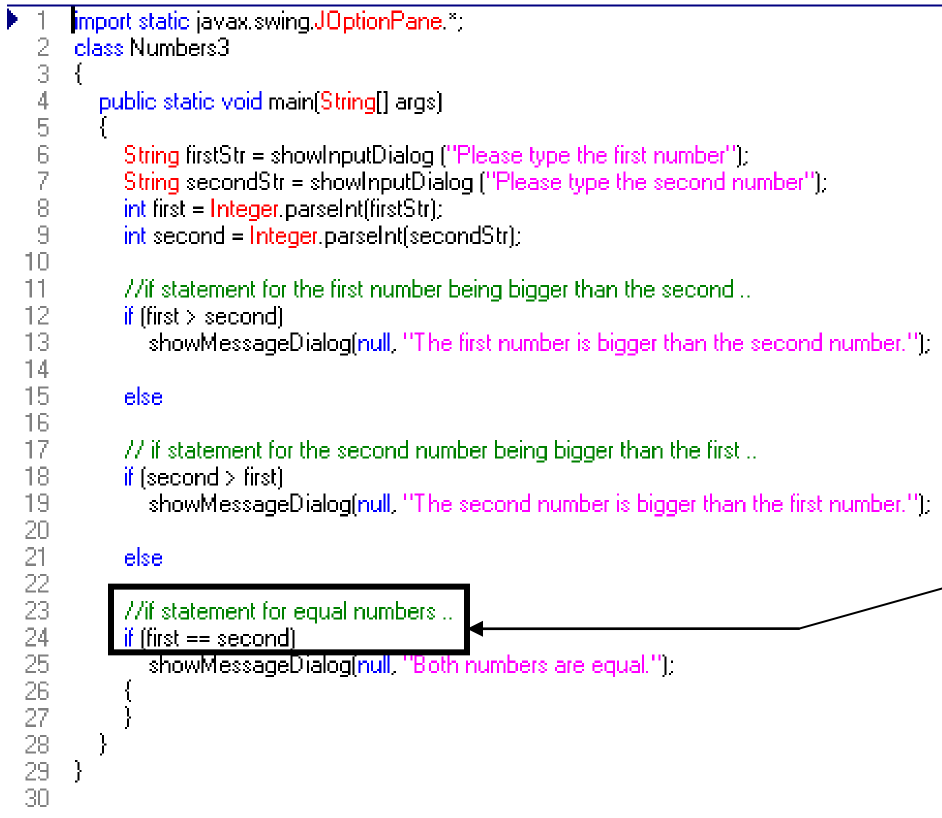


```java
import static javax.swing.JOptionPane.*;
class Numbers2
{
   public static void main(String[] args)
   {
      String firstStr = showInputDialog ("Please type the first number");
      String secondStr = showInputDialog ("Please type the second number");
      int first = Integer.parseInt(firstStr);
      int second = Integer.parseInt(secondStr);

      //if statement for the first number being bigger than the second ..
      if (first > second)
         showMessageDialog(null, "The first number is bigger than the second number.");

      else

      // if statement for the second number being bigger than the first ..
      if (second > first)
         showMessageDialog(null, "The second number is bigger than the first number");
   }
}
```

Unlike “Numbers.java”, this program (“Numbers2.java”) uses JOptionPanes to enter the two numbers allowing the program to work out which one is bigger.

To run this program, Java needs to be clicked NOT Java + Args.

5.3 "Numbers3.java"

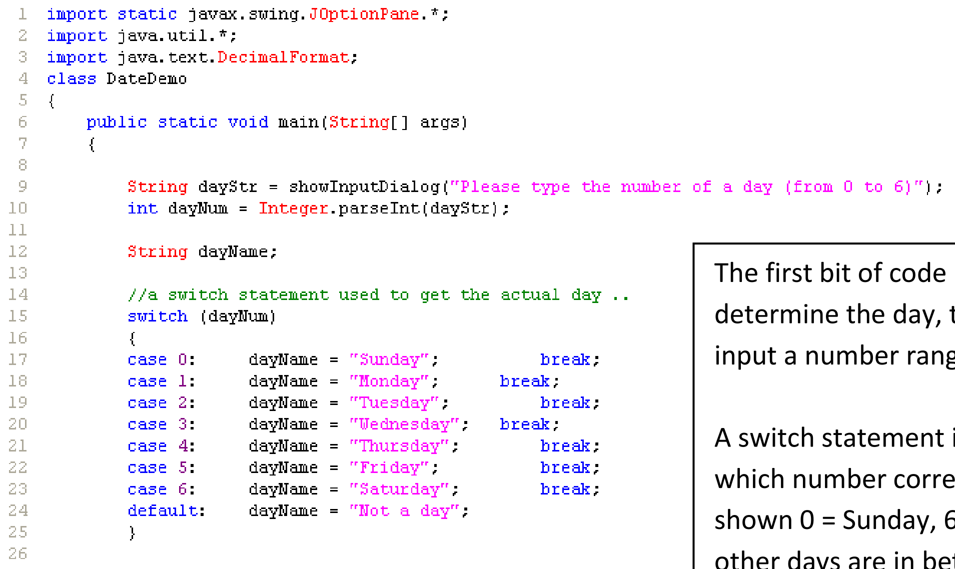


```java
import static javax.swing.JOptionPane.*;
class Numbers3
{
   public static void main(String[] args)
   {
      String firstStr = showInputDialog ("Please type the first number");
      String secondStr = showInputDialog ("Please type the second number");
      int first = Integer.parseInt(firstStr);
      int second = Integer.parseInt(secondStr);

      //if statement for the first number being bigger than the second ..
      if (first > second)
         showMessageDialog(null, "The first number is bigger than the second number.");

      else

      // if statement for the second number being bigger than the first ..
      if (second > first)
         showMessageDialog(null, "The second number is bigger than the first number.");

      else

      //if statement for equal numbers ..
      if (first == second)
         showMessageDialog(null, "Both numbers are equal.");
      {
      }
   }
}
```

The layout of this program is the same as Numbers2, except for the third IF statement. This IF statement has "first == second" instead of the usual ">" or "<".

The "==" in this IF statement means equal to. It uses two equal signs because one equal sign already stands for something else in Java.

5.4 "Date Demo.java"



```java
import static javax.swing.JOptionPane.*;
import java.util.*;
import java.text.DecimalFormat;
class DateDemo
{
    public static void main(String[] args)
    {

        String dayStr = showInputDialog("Please type the number of a day (from 0 to 6)");
        int dayNum = Integer.parseInt(dayStr);

        String dayName;

        //a switch statement used to get the actual day ..
        switch (dayNum)
        {
        case 0:     dayName = "Sunday";         break;
        case 1:     dayName = "Monday";     break;
        case 2:     dayName = "Tuesday";        break;
        case 3:     dayName = "Wednesday";  break;
        case 4:     dayName = "Thursday";       break;
        case 5:     dayName = "Friday";         break;
        case 6:     dayName = "Saturday";       break;
        default:    dayName = "Not a day";
        }

```

The first bit of code is the code used to determine the day, the user is asked to input a number ranging from 0 to 6.

A switch statement is then used to see which number corresponds to which day. As shown 0 = Sunday, 6 = Saturday and all the other days are in between.

```

        String dayofmonthStr = showInputDialog("Please type the number of the month's day (from 0 to 30");
        int dayofmonthNum = Integer.parseInt(dayofmonthStr);

        String dayofmonthName;

        //a switch statement used to get the day of the month ..
        switch (dayofmonthNum)
        {
        case 0:         dayofmonthName = "1st";     break;
        case 1:         dayofmonthName = "2nd";     break;
        case 2:         dayofmonthName = "3rd";     break;
        case 3:         dayofmonthName = "4th";     break;
        case 4:         dayofmonthName = "5th";     break;
        case 5:         dayofmonthName = "6th";     break;
        case 6:         dayofmonthName = "7th";     break;
        case 7:         dayofmonthName = "8th";     break;
        case 8:         dayofmonthName = "9th";     break;
        case 9:         dayofmonthName = "10th";    break;
        case 10:        dayofmonthName = "11th";    break;
        case 11:        dayofmonthName = "12th";    break;
        case 12:        dayofmonthName = "13th";    break;
        case 13:        dayofmonthName = "14th";    break;
        case 14:        dayofmonthName = "15th";    break;
        case 15:        dayofmonthName = "16th";    break;
        case 16:        dayofmonthName = "17th";    break;
        case 17:        dayofmonthName = "18th";    break;
        case 18:        dayofmonthName = "19th";    break;
        case 19:        dayofmonthName = "20th";    break;
        case 20:        dayofmonthName = "21st";    break;
        case 21:        dayofmonthName = "22nd";    break;
        case 22:        dayofmonthName = "23rd";    break;
        case 23:        dayofmonthName = "24th";    break;
        case 24:        dayofmonthName = "25th";    break;
        case 25:        dayofmonthName = "26th";    break;
        case 26:        dayofmonthName = "27th";    break;
        case 27:        dayofmonthName = "28th";    break;
        case 28:        dayofmonthName = "29th";    break;
        case 29:        dayofmonthName = "30th";    break;
        case 30:        dayofmonthName = "31st";    break;
        default:        dayofmonthName = "Not a day of the month";
        }
```

The next bit of code is quite similar to the first bit, except this time the program asks for the number representing the day of the month.

As before a switch statement is used for all 31 (possible) days in a month. 0 = 1st, 30 = 31st and the other days of the month are in between.

```

        String monthStr = showInputDialog("Please type the number of a month (from 0 to 11)");
        int monthNum = Integer.parseInt(monthStr);

        String monthName;

        //a switch statement used to get the month of the year ..
        switch (monthNum)
        {
        case 0:     monthName = "January";          break;
        case 1:     monthName = "February";     break;
        case 2:     monthName = "March";            break;
        case 3:     monthName = "April";                break;
        case 4:     monthName = "May";              break;
        case 5:     monthName = "June";             break;
        case 6:     monthName = "July";             break;
        case 7:     monthName = "August";           break;
        case 8:     monthName = "September";    break;
        case 9:     monthName = "October";      break;
        case 10:    monthName = "November";     break;
        case 11:    monthName = "December";     break;
        default:    monthName = "Not a month";
        }


        Calendar today = new GregorianCalendar();
        int dayOfWeek = today.get(Calendar.DAY_OF_WEEK);
        int dayOfMonth = today.get(Calendar.DATE);
        int month = today.get(Calendar.MONTH);
        int year = today.get(Calendar.YEAR);

        //the output, which display the numbers entered into all 3 switches as text and the current year ..
        showMessageDialog(null, "Today is " + dayName + " " + dayofmonthName + " " + monthName + " " + today.get(Calendar.YEAR));

    }
}
```

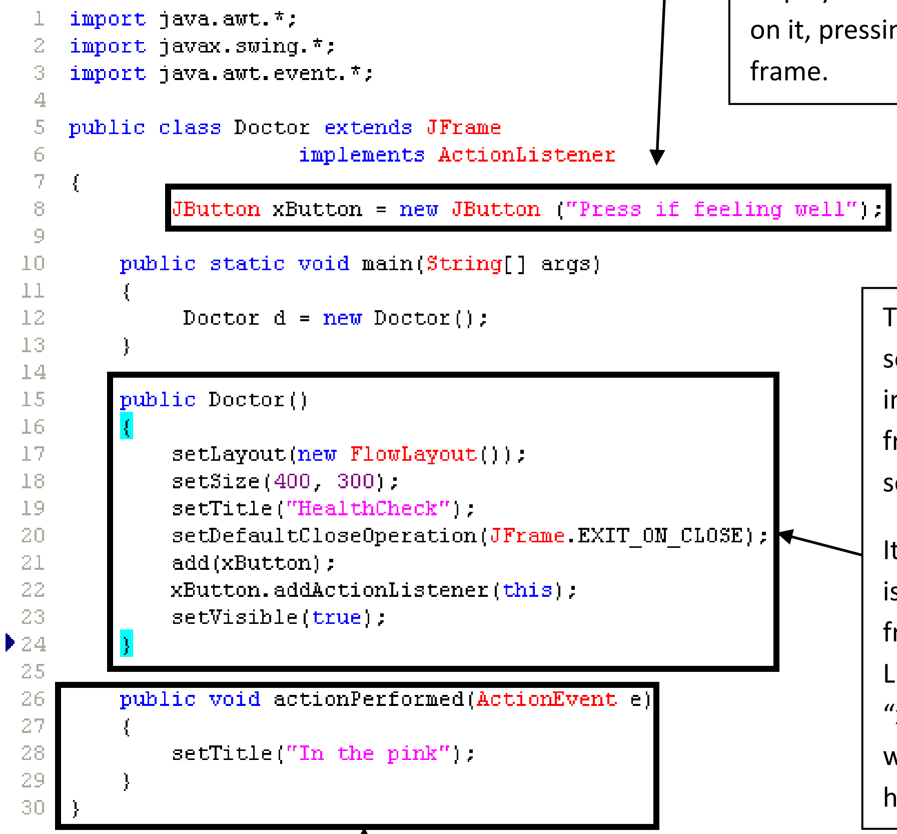
The next bit of code shows a third message box asking the user to enter the number of a month.

As usual there is a switch statement that I wrote to deal with this with 0 = January, 11 = December and the other months in between those two.

This line right at the end is the output which displays the day name, day of the month, the month name and the current year. The numbers that are input by the user for the day name, day of the month and the month name are output here as text, this is thanks to the switch statements and certain variables.

6 – GUI Programming (Part 1)

6.1 “Doctor.java”



```
 1  import java.awt.*;
 2  import javax.swing.*;
 3  import java.awt.event.*;
 4
 5  public class Doctor extends JFrame
 6                        implements ActionListener
 7  {
 8          JButton xButton = new JButton ("Press if feeling well");
 9
10      public static void main(String[] args)
11      {
12           Doctor d = new Doctor();
13      }
14
15      public Doctor()
16      {
17          setLayout(new FlowLayout());
18          setSize(400, 300);
19          setTitle("HealthCheck");
20          setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
21          add(xButton);
22          xButton.addActionListener(this);
23          setVisible(true);
24      }
25
26      public void actionPerformed(ActionEvent e)
27      {
28          setTitle("In the pink");
29      }
30  }
```

Line 8 of the code is the creation of a new button for the java frame. It displays the text “Press if feeling well” on it, pressing it changes the title of the frame.

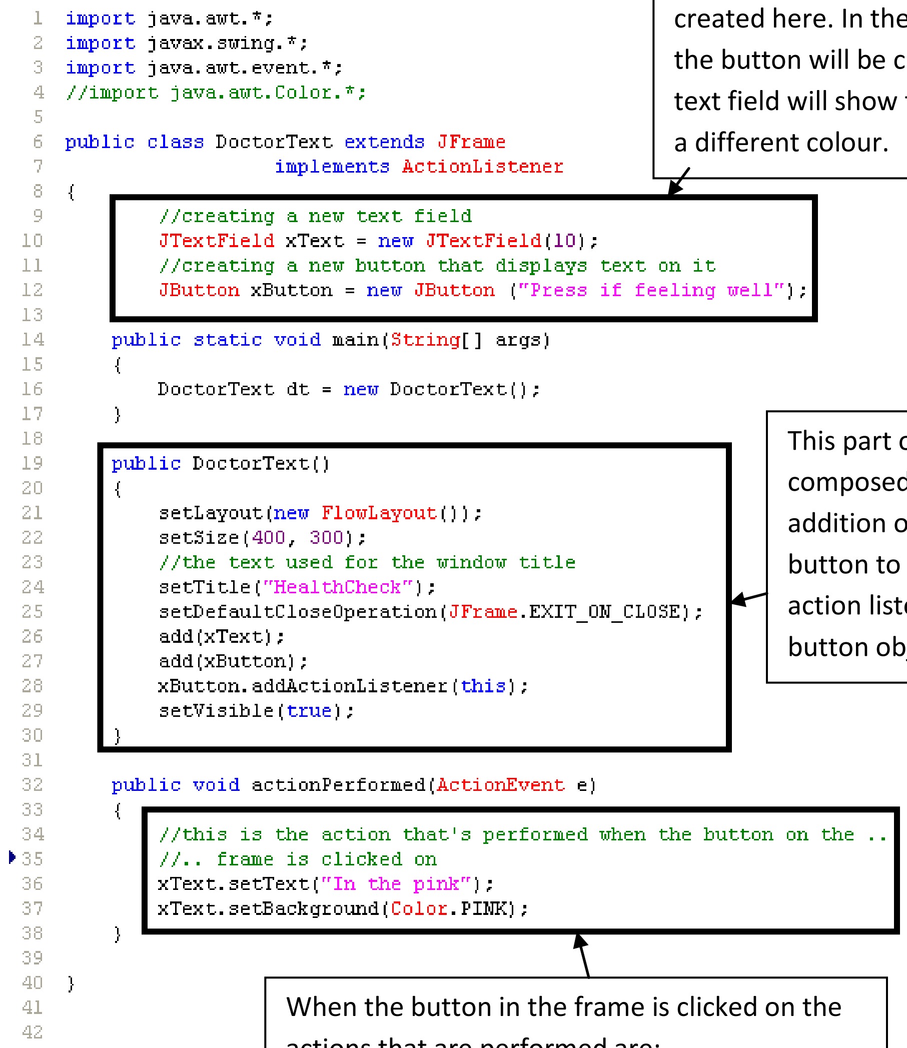
This section of the code contains some automatically generated information when creating a new frame, such as setLayout, setSize, setVisible etc.

It also contains “add(xButton)” which is basically adding a button to the frame. I have also added an Action Listener to the button called “xButton” so the computer will know when an action involving that button has taken place.

Line 28 of the code is part of the Actions performed when the button is clicked on the frame when the program is running. What it does is when the button is clicked; it changes the title of the actual frame to “In the pink”.

6.2 "DoctorText.java"

A new text field called "xText" and a new button called "xButton" are created here. In the actual program the button will be clicked on and the text field will show text and change to a different colour.



```java
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
//import java.awt.Color.*;

public class DoctorText extends JFrame
                  implements ActionListener
{
        //creating a new text field
        JTextField xText = new JTextField(10);
        //creating a new button that displays text on it
        JButton xButton = new JButton ("Press if feeling well");

    public static void main(String[] args)
    {
        DoctorText dt = new DoctorText();
    }

    public DoctorText()
    {
        setLayout(new FlowLayout());
        setSize(400, 300);
        //the text used for the window title
        setTitle("HealthCheck");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        add(xText);
        add(xButton);
        xButton.addActionListener(this);
        setVisible(true);
    }

    public void actionPerformed(ActionEvent e)
    {
        //this is the action that's performed when the button on the ..
        //.. frame is clicked on
        xText.setText("In the pink");
        xText.setBackground(Color.PINK);
    }

}
```

This part of the code is composed of frame details, the addition of the text field and button to the frame and the action listeners set to the button object.

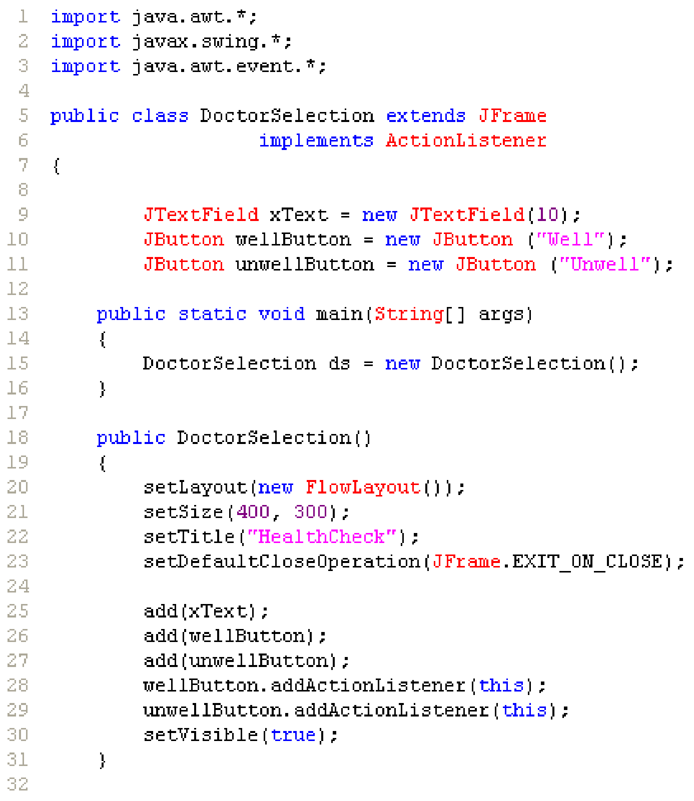
When the button in the frame is clicked on the actions that are performed are;

- The words "In the pink" are displayed in the text field that was once empty.
- The background colour of the text field then changes to a pink colour.

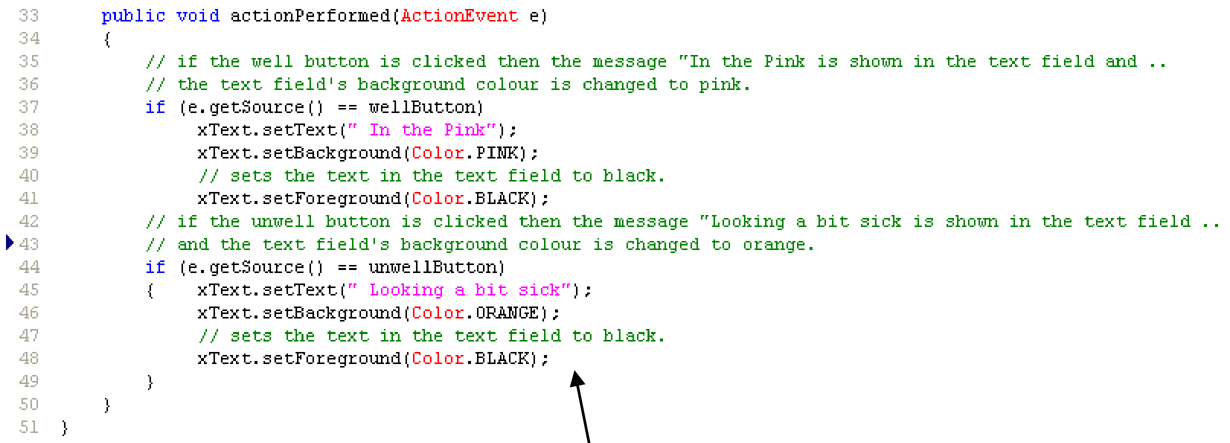
6.3 “DoctorSelection.java”

In the screenshot to left, the top half of the code for DoctorSelection.java is displayed. It’s basically all setting the size and properties of different objects in the program such as text fields and buttons.

It also shows the objects being added to the J Frame as well as the Action Listeners that are added to listen out for when the buttons are pressed.



```java
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class DoctorSelection extends JFrame
                  implements ActionListener
{

        JTextField xText = new JTextField(10);
        JButton wellButton = new JButton ("Well");
        JButton unwellButton = new JButton ("Unwell");

    public static void main(String[] args)
    {
        DoctorSelection ds = new DoctorSelection();
    }

    public DoctorSelection()
    {
        setLayout(new FlowLayout());
        setSize(400, 300);
        setTitle("HealthCheck");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        add(xText);
        add(wellButton);
        add(unwellButton);
        wellButton.addActionListener(this);
        unwellButton.addActionListener(this);
        setVisible(true);
    }

```



```java
    public void actionPerformed(ActionEvent e)
    {
        // if the well button is clicked then the message "In the Pink is shown in the text field and ..
        // the text field's background colour is changed to pink.
        if (e.getSource() == wellButton)
            xText.setText(" In the Pink");
            xText.setBackground(Color.PINK);
            // sets the text in the text field to black.
            xText.setForeground(Color.BLACK);
        // if the unwell button is clicked then the message "Looking a bit sick is shown in the text field ..
        // and the text field's background colour is changed to orange.
        if (e.getSource() == unwellButton)
        {   xText.setText(" Looking a bit sick");
            xText.setBackground(Color.ORANGE);
            // sets the text in the text field to black.
            xText.setForeground(Color.BLACK);
        }
    }
}
```

In the Action Performed part of the code, the code is displayed for when the “well” button is clicked and for also when the “un-well” button is clicked.

As you can see the text field turns a different colour depending on which button was clicked on as well as a different piece of text being displayed in there too.

6.4 "DoctorOption.java"

This section of the code in DoctorOption.java shows the creation of two radio buttons (named wellButton and unwellButton), the creation of a new button group (named userCondition), the creation of a Text Field and the creation of a button.

```java
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class DoctorOption extends JFrame
                  implements ActionListener
{
        //creation of two new radio buttons
        JRadioButton wellButton = new JRadioButton ("Well", true);
        JRadioButton unwellButton = new JRadioButton ("Unwell", false);
        //creation of a new button group
        ButtonGroup userCondition = new ButtonGroup();
        JTextField xText = new JTextField(20);
        JButton doctorButton = new JButton ("Tell the doctor how you feel");

    public static void main(String[] args)
    {
        DoctorOption doo = new DoctorOption();
    }


        public DoctorOption()
    {
        setLayout(new FlowLayout());
        setSize(400, 300);
        setTitle("HealthCheck");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        add(xText);
        //adding the well and unwell radio buttons to the button group
        userCondition.add(wellButton);
        userCondition.add(unwellButton);
        //adding the well and unwell radio buttons and the doctor button to
        //the frame
        add(wellButton);
        add(unwellButton);
        add(doctorButton);
        doctorButton.addActionListener(this);
        setVisible(true);
    }
```

The adding of the well and unwell radio buttons to the button group takes places in this section of the code.

Underneath the adding of radio buttons to the radio group, there is some code that adds all 3 buttons to the frame in java. If they're not added to the frame then they won't show up.

```java

    public void actionPerformed(ActionEvent e)
    {
        String health;
        if (wellButton.isSelected())
         {health = " In the Pink";
         xText.setText(health);
         //changes the background color of the text field to ..
         // .. pink when the Well button is selected
         xText.setBackground(Color.PINK);}
        else
        {health = " Looking a bit sick";
         xText.setText(health);
         xText.setBackground(Color.ORANGE);}
    }
}
```

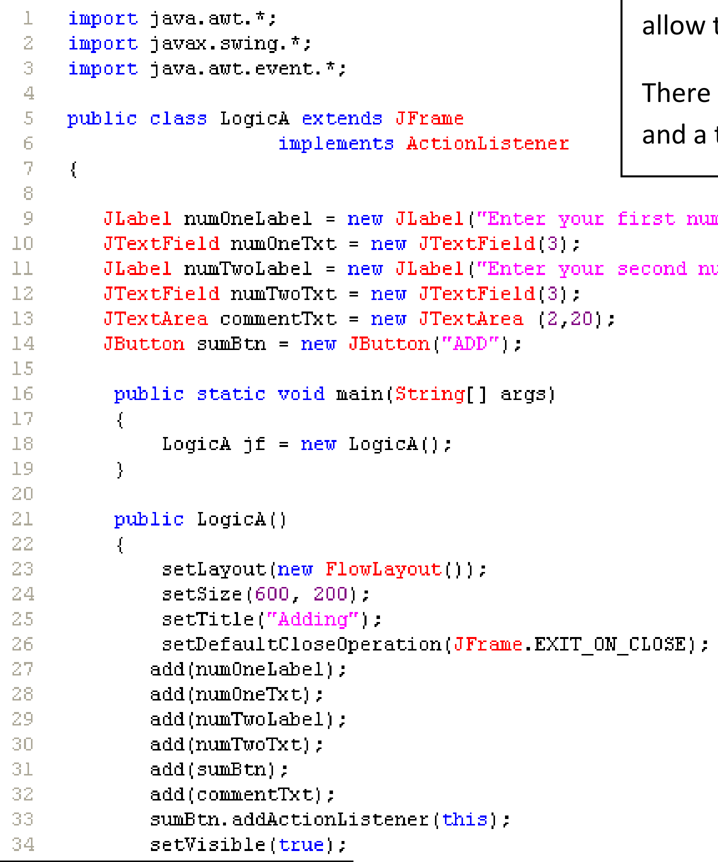
This bottom section of the code is where all the actions that are performed are listed, such as the text field displaying one message or another, the text field changing to either a pink colour or an orange colour and so forth.

7 – GUI Programming (Part 2)

In the screenshot to the left, the code for the program Logic A.java is shown. As you can see it contains two text fields (along with two labels) which allow the user to enter two numbers.

There is also a button to add both numbers together and a text area for displaying the answer.

7.1 "Logic A.java"



```java
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class LogicA extends JFrame
                  implements ActionListener
{

   JLabel numOneLabel = new JLabel("Enter your first number:  ");
   JTextField numOneTxt = new JTextField(3);
   JLabel numTwoLabel = new JLabel("Enter your second number:  ");
   JTextField numTwoTxt = new JTextField(3);
   JTextArea commentTxt = new JTextArea (2,20);
   JButton sumBtn = new JButton("ADD");

    public static void main(String[] args)
    {
        LogicA jf = new LogicA();
    }

    public LogicA()
    {
        setLayout(new FlowLayout());
        setSize(600, 200);
        setTitle("Adding");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
       add(numOneLabel);
       add(numOneTxt);
       add(numTwoLabel);
       add(numTwoTxt);
       add(sumBtn);
       add(commentTxt);
       sumBtn.addActionListener(this);
       setVisible(true);
```

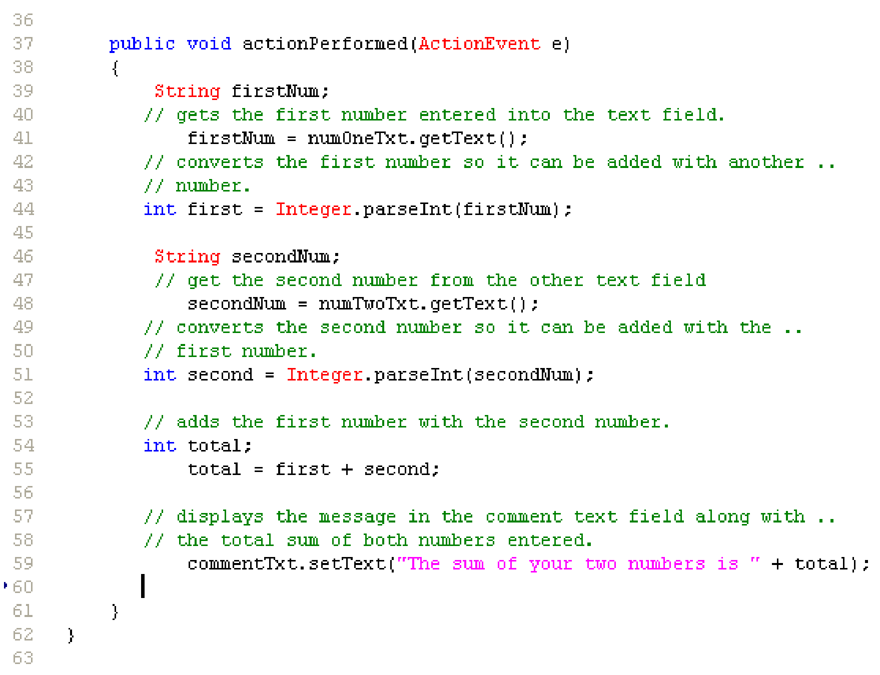
To the left are the details of the JFrame and the adding of the labels, text fields, text area and the sum button.

It also adds an Action Listener to the sum button and allows the setting of the JFrame's title, size and layout.

To the right is the Action Performed section of the Logic A code. The String variables firstNum and secondNum get the numbers entered by the user in the first and second text fields with the use of .getText().

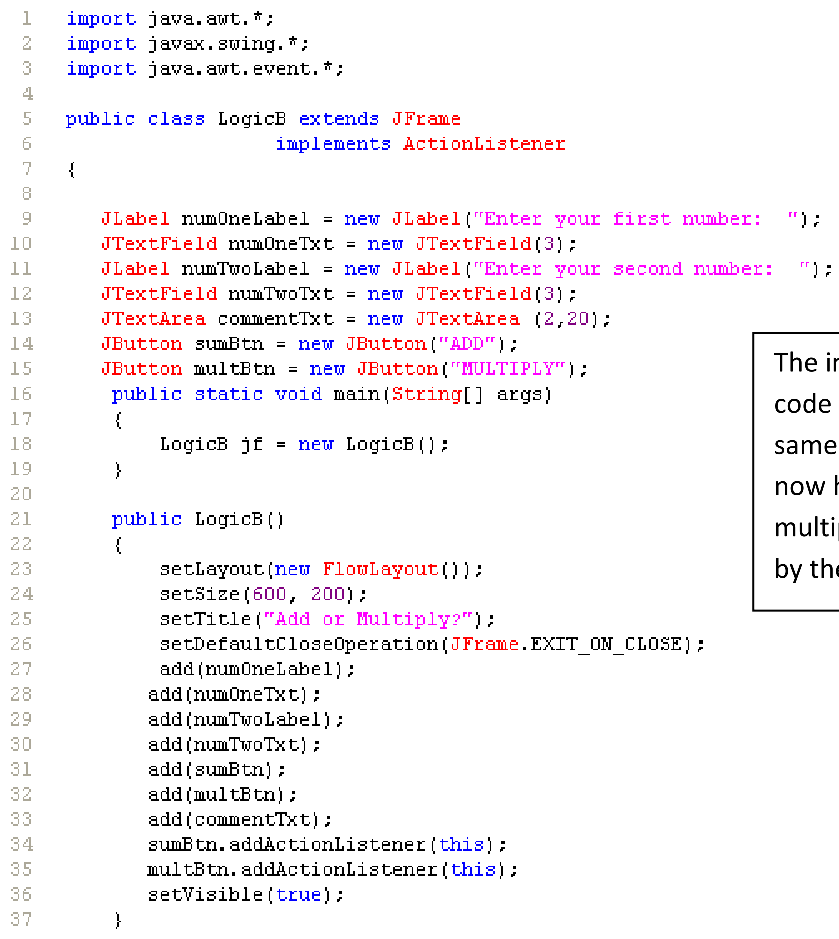
Both numbers are then converted to integers so they can be added together with the use of Integer.parseInt().

The variable Total is then declared as an integer as is the value of the first number + the second number. In order to display an integer value (the total) in the text area, a string needs to be put in first with a "+ total" (or whatever the variable name is) after it.



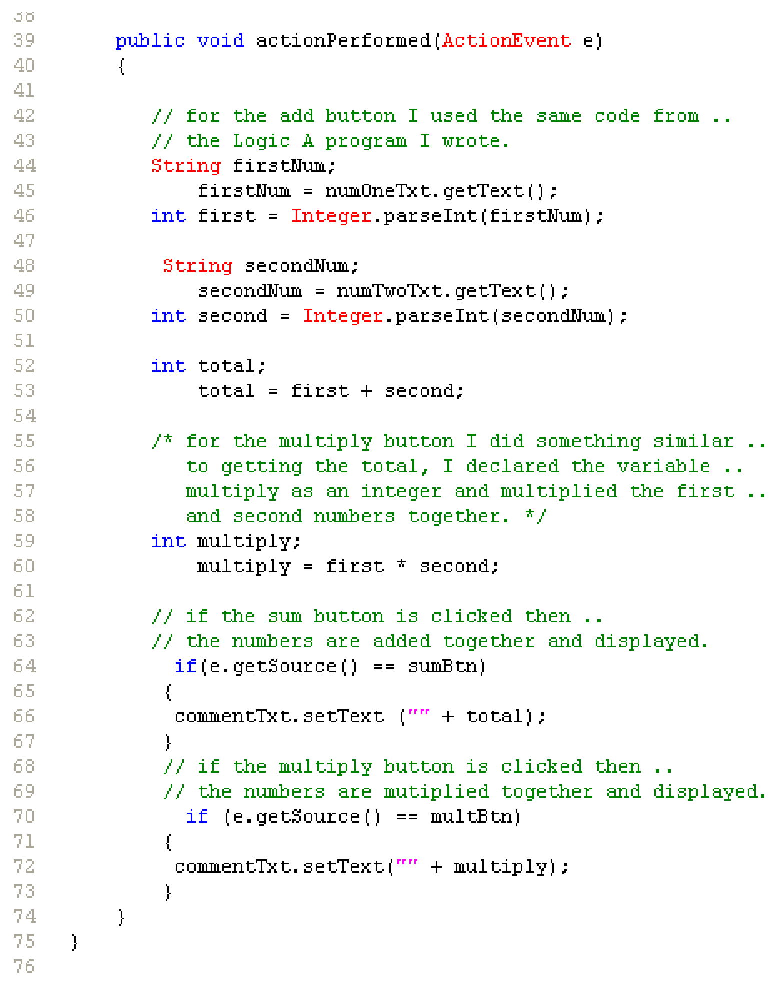
```java

      public void actionPerformed(ActionEvent e)
      {
          String firstNum;
         // gets the first number entered into the text field.
             firstNum = numOneTxt.getText();
         // converts the first number so it can be added with another ..
         // number.
         int first = Integer.parseInt(firstNum);

          String secondNum;
          // get the second number from the other text field
             secondNum = numTwoTxt.getText();
         // converts the second number so it can be added with the ..
         // first number.
         int second = Integer.parseInt(secondNum);

         // adds the first number with the second number.
         int total;
             total = first + second;

         // displays the message in the comment text field along with ..
         // the total sum of both numbers entered.
             commentTxt.setText("The sum of your two numbers is " + total);

      }
  }

```

7.2 “Logic B.java”



```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class LogicB extends JFrame
                  implements ActionListener
{

   JLabel numOneLabel = new JLabel("Enter your first number:  ");
   JTextField numOneTxt = new JTextField(3);
   JLabel numTwoLabel = new JLabel("Enter your second number:  ");
   JTextField numTwoTxt = new JTextField(3);
   JTextArea commentTxt = new JTextArea (2,20);
   JButton sumBtn = new JButton("ADD");
   JButton multBtn = new JButton("MULTIPLY");
    public static void main(String[] args)
    {
        LogicB jf = new LogicB();
    }

    public LogicB()
    {
        setLayout(new FlowLayout());
        setSize(600, 200);
        setTitle("Add or Multiply?");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        add(numOneLabel);
       add(numOneTxt);
       add(numTwoLabel);
       add(numTwoTxt);
       add(sumBtn);
       add(multBtn);
       add(commentTxt);
       sumBtn.addActionListener(this);
       multBtn.addActionListener(this);
       setVisible(true);
    }
```

The image to the left displays the code for “Logic B.java”. It has the same features as Logic A except it now has an additional button for multiplying the two numbers entered by the user.

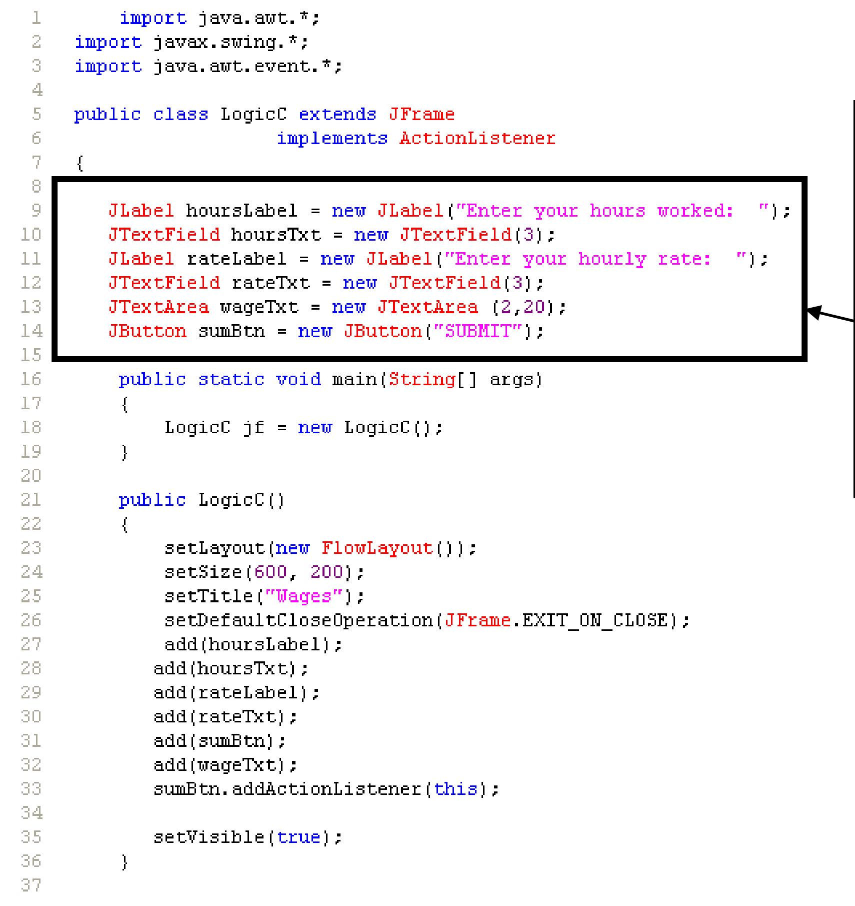


```
    public void actionPerformed(ActionEvent e)
    {

       // for the add button I used the same code from ..
       // the Logic A program I wrote.
       String firstNum;
           firstNum = numOneTxt.getText();
       int first = Integer.parseInt(firstNum);

        String secondNum;
           secondNum = numTwoTxt.getText();
       int second = Integer.parseInt(secondNum);

       int total;
           total = first + second;

       /* for the multiply button I did something similar ..
          to getting the total, I declared the variable ..
          multiply as an integer and multiplied the first ..
          and second numbers together. */
       int multiply;
           multiply = first * second;

       // if the sum button is clicked then ..
       // the numbers are added together and displayed.
         if(e.getSource() == sumBtn)
        {
         commentTxt.setText ("" + total);
        }
        // if the multiply button is clicked then ..
        // the numbers are mutiplied together and displayed.
          if (e.getSource() == multBtn)
        {
         commentTxt.setText("" + multiply);
        }
    }
}
```

The Action Performed section of Logic B's code also has the same features as Logic A's code, however it has a new integer variable called multiply which multiples the first number with the second number.

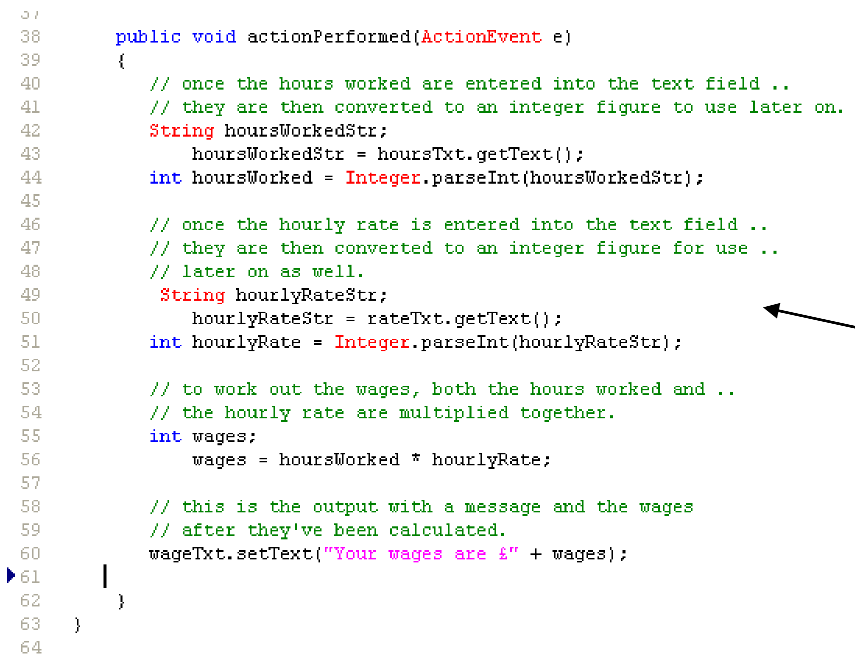
The "e.getSource () == sumBtn" is used to display the total of both numbers added together in the text area. While the "e.getSource () == multBtn" is used to display the total of both numbers multiplied together in the text area.

7.3 “Logic C.java”



```java
    import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class LogicC extends JFrame
                  implements ActionListener
{

   JLabel hoursLabel = new JLabel("Enter your hours worked:  ");
   JTextField hoursTxt = new JTextField(3);
   JLabel rateLabel = new JLabel("Enter your hourly rate:  ");
   JTextField rateTxt = new JTextField(3);
   JTextArea wageTxt = new JTextArea (2,20);
   JButton sumBtn = new JButton("SUBMIT");

    public static void main(String[] args)
    {
        LogicC jf = new LogicC();
    }

    public LogicC()
    {
        setLayout(new FlowLayout());
        setSize(600, 200);
        setTitle("Wages");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        add(hoursLabel);
       add(hoursTxt);
       add(rateLabel);
       add(rateTxt);
       add(sumBtn);
       add(wageTxt);
       sumBtn.addActionListener(this);

       setVisible(true);
    }

```

To the left is the code for “Logic C.java”. As you can see it’s slightly different as the labels now talk about hours worked and the hourly rate, however it’s still the same two number concept. The button this time also says “Submit” instead of “Add” or “Multiply”.



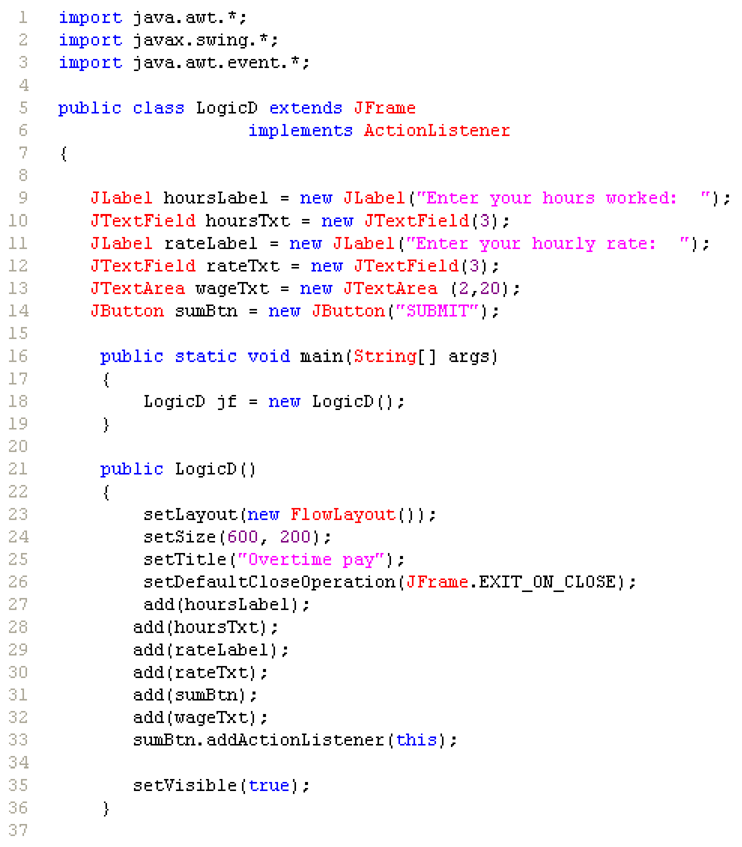
```java
    public void actionPerformed(ActionEvent e)
    {
       // once the hours worked are entered into the text field ..
       // they are then converted to an integer figure to use later on.
       String hoursWorkedStr;
           hoursWorkedStr = hoursTxt.getText();
       int hoursWorked = Integer.parseInt(hoursWorkedStr);

       // once the hourly rate is entered into the text field ..
       // they are then converted to an integer figure for use ..
       // later on as well.
        String hourlyRateStr;
           hourlyRateStr = rateTxt.getText();
       int hourlyRate = Integer.parseInt(hourlyRateStr);

       // to work out the wages, both the hours worked and ..
       // the hourly rate are multiplied together.
       int wages;
           wages = hoursWorked * hourlyRate;

       // this is the output with a message and the wages
       // after they've been calculated.
       wageTxt.setText("Your wages are £" + wages);

    }
}
```

Here is the Action Performed section of Logic C’s code. It’s more or less the same concept before, except the hours worked represents the first number that’s entered and the hourly rate represents the second number entered.

The wages that are output in the text area is basically the total of both numbers multiplied together. It also has a different message displayed before the wages which allows the integer to be shown.

7.4 "Logic D.java"

The Image to the right is a screenshot of the code from "Logic D.java". As you can see it's more or less the same thing as Logic C. The title of the JFrame has also changed to "Overtime pay".



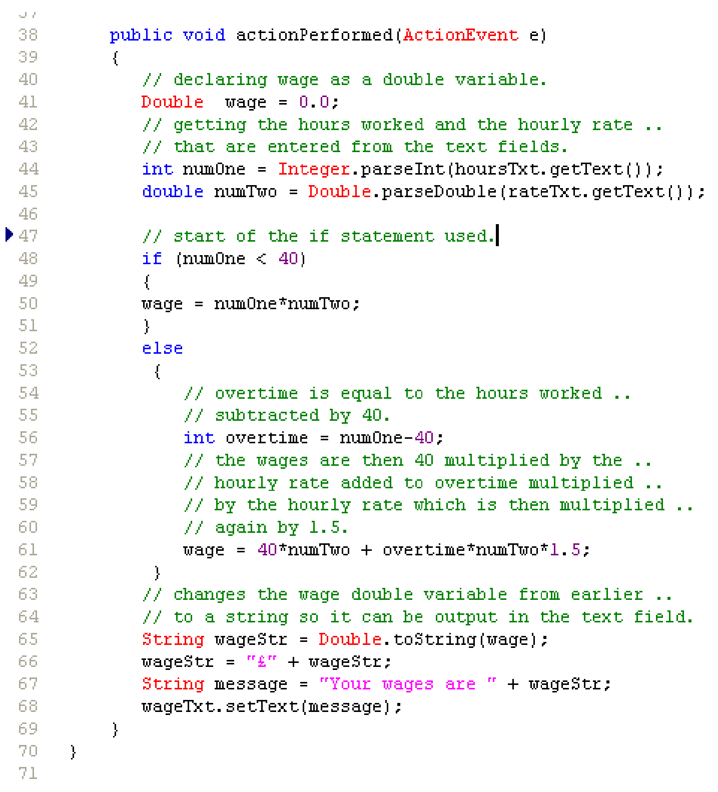
```
1  import java.awt.*;
2  import javax.swing.*;
3  import java.awt.event.*;
4
5  public class LogicD extends JFrame
6                    implements ActionListener
7  {
8
9     JLabel hoursLabel = new JLabel("Enter your hours worked:  ");
10    JTextField hoursTxt = new JTextField(3);
11    JLabel rateLabel = new JLabel("Enter your hourly rate:  ");
12    JTextField rateTxt = new JTextField(3);
13    JTextArea wageTxt = new JTextArea (2,20);
14    JButton sumBtn = new JButton("SUBMIT");
15
16     public static void main(String[] args)
17     {
18         LogicD jf = new LogicD();
19     }
20
21     public LogicD()
22     {
23         setLayout(new FlowLayout());
24         setSize(600, 200);
25         setTitle("Overtime pay");
26         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
27         add(hoursLabel);
28        add(hoursTxt);
29        add(rateLabel);
30        add(rateTxt);
31        add(sumBtn);
32        add(wageTxt);
33        sumBtn.addActionListener(this);
34
35        setVisible(true);
36     }
37
```

In the Action Performed section of Logic D's code, you can see that wage has been declared as a Double variable so it can be set as the value "0.0".

At lines 44 and 45, the integer variables of "numOne" and "numTwo" are declared as set as the entered values which were taken from the text fields and converted into Integer values.

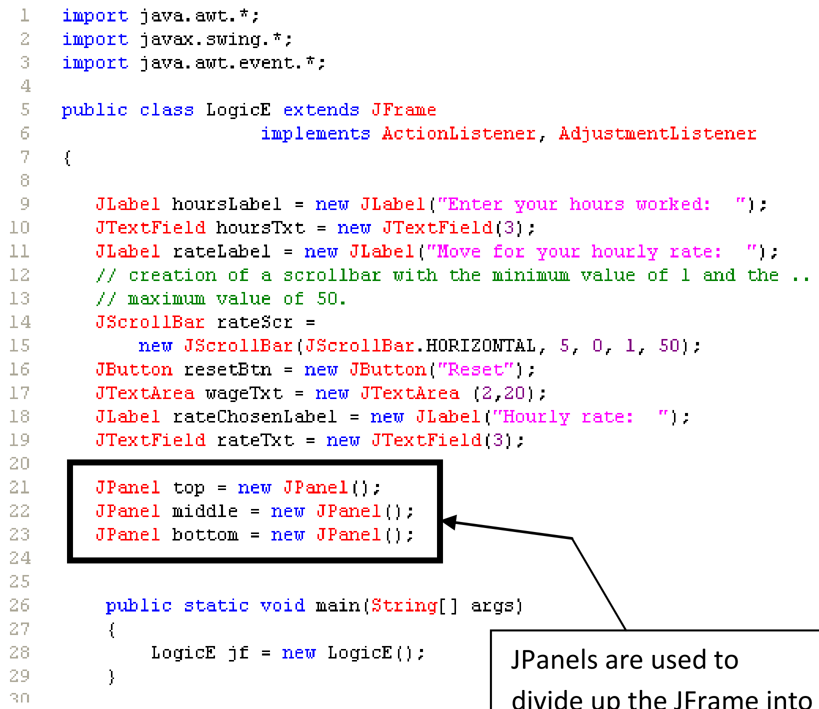
An IF statement is used to see if the hours worked is under 40 hours, if it is then it calculates normally, if it's not then additional overtime pay is added (see comments for more details).

The variable "wageStr" converts the wage double value to String. It's then given the "£" sign and then declared as a variable called message, it's then displayed in the wage text field.



```
38     public void actionPerformed(ActionEvent e)
39     {
40        // declaring wage as a double variable.
41        Double  wage = 0.0;
42        // getting the hours worked and the hourly rate ..
43        // that are entered from the text fields.
44        int numOne = Integer.parseInt(hoursTxt.getText());
45        double numTwo = Double.parseDouble(rateTxt.getText());
46
47        // start of the if statement used.
48        if (numOne < 40)
49        {
50        wage = numOne*numTwo;
51        }
52        else
53         {
54            // overtime is equal to the hours worked ..
55            // subtracted by 40.
56            int overtime = numOne-40;
57            // the wages are then 40 multiplied by the ..
58            // hourly rate added to overtime multiplied ..
59            // by the hourly rate which is then multiplied ..
60            // again by 1.5.
61            wage = 40*numTwo + overtime*numTwo*1.5;
62         }
63        // changes the wage double variable from earlier ..
64        // to a string so it can be output in the text field.
65        String wageStr = Double.toString(wage);
66        wageStr = "£" + wageStr;
67        String message = "Your wages are " + wageStr;
68        wageTxt.setText(message);
69     }
70  }
71
```

7.5 "Logic E.java"



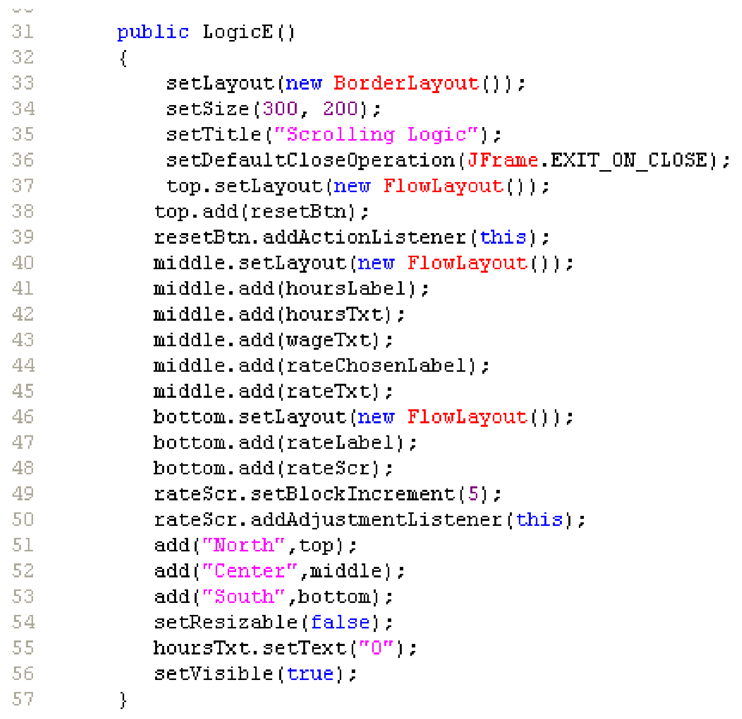
```
1   import java.awt.*;
2   import javax.swing.*;
3   import java.awt.event.*;
4
5   public class LogicE extends JFrame
6                   implements ActionListener, AdjustmentListener
7   {
8
9      JLabel hoursLabel = new JLabel("Enter your hours worked:  ");
10     JTextField hoursTxt = new JTextField(3);
11     JLabel rateLabel = new JLabel("Move for your hourly rate:  ");
12     // creation of a scrollbar with the minimum value of 1 and the ..
13     // maximum value of 50.
14     JScrollBar rateScr =
15         new JScrollBar(JScrollBar.HORIZONTAL, 5, 0, 1, 50);
16     JButton resetBtn = new JButton("Reset");
17     JTextArea wageTxt = new JTextArea (2,20);
18     JLabel rateChosenLabel = new JLabel("Hourly rate:  ");
19     JTextField rateTxt = new JTextField(3);
20
21     JPanel top = new JPanel();
22     JPanel middle = new JPanel();
23     JPanel bottom = new JPanel();
24
25
26      public static void main(String[] args)
27      {
28          LogicE jf = new LogicE();
29      }
30
```

JPanels are used to divide up the JFrame into top, middle and bottom sections.

The screenshot to the left shows the code for Logic E. As you can see there are a few significant changes, such as the addition of a scroll bar to enter the hourly rate and a reset button used to set the scroll bar back to its default value.

Lines 14 and 15 show the creation of the scroll bar and the setting of it's properties (as you can see in the code it's orientation is set to be Horizontal, it's Initial Value is 5, the Extent of Handle is 0, the Min Value is 1 and the Max Value is 50.

A text field is still used to show the hourly rate that's selected by the scroll bar.

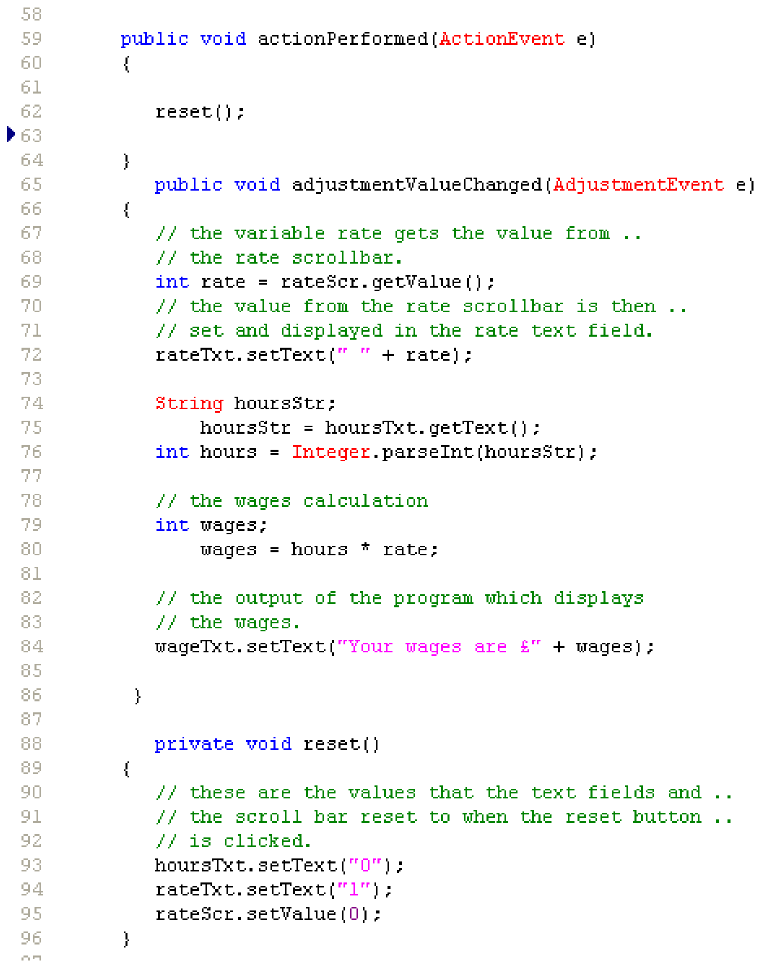


```
31      public LogicE()
32      {
33          setLayout(new BorderLayout());
34          setSize(300, 200);
35          setTitle("Scrolling Logic");
36          setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
37          top.setLayout(new FlowLayout());
38         top.add(resetBtn);
39         resetBtn.addActionListener(this);
40         middle.setLayout(new FlowLayout());
41         middle.add(hoursLabel);
42         middle.add(hoursTxt);
43         middle.add(wageTxt);
44         middle.add(rateChosenLabel);
45         middle.add(rateTxt);
46         bottom.setLayout(new FlowLayout());
47         bottom.add(rateLabel);
48         bottom.add(rateScr);
49         rateScr.setBlockIncrement(5);
50         rateScr.addAdjustmentListener(this);
51         add("North",top);
52         add("Center",middle);
53         add("South",bottom);
54         setResizable(false);
55         hoursTxt.setText("0");
56         setVisible(true);
57      }
```

In order to add certain objects to a specific JPanel on the JFrame you need to first type the name then put a dot and then type "add(the name of object);". An example would be "top.add(resetBtn);" as shown in the screenshot to the left. This reset button will be added to the top JPanel on the JFrame.

An Adjustment Listener is set to the rate scrollbar so the program listens out for when the user moves the scrollbar. Adding North, Center and South respectively add and set the JPanels to the appropriate sections of the JFrame. The value of hoursTxt is set to 0.

The JFrame isn't made resizable as it's value is set to false and Set Visible is set to true so the actual JFrame can be seen on the screen when it's run.



```
    public void actionPerformed(ActionEvent e)
    {

       reset();


    }
       public void adjustmentValueChanged(AdjustmentEvent e)
    {
       // the variable rate gets the value from ..
       // the rate scrollbar.
       int rate = rateScr.getValue();
       // the value from the rate scrollbar is then ..
       // set and displayed in the rate text field.
       rateTxt.setText(" " + rate);

       String hoursStr;
           hoursStr = hoursTxt.getText();
       int hours = Integer.parseInt(hoursStr);

       // the wages calculation
       int wages;
           wages = hours * rate;

       // the output of the program which displays
       // the wages.
       wageTxt.setText("Your wages are £" + wages);

     }

       private void reset()
    {
       // these are the values that the text fields and ..
       // the scroll bar reset to when the reset button ..
       // is clicked.
       hoursTxt.setText("0");
       rateTxt.setText("1");
       rateScr.setValue(0);
    }
```

The image to the left shows the Action Performed section of Logic E's code. As you can see there is a section called "Adjustment Value Changed" the code in there is mainly related to the scroll bar that was added.

The integer variable rate has been declared and gets its value from whatever number the rate scrollbar is set to, the rate text field then displays the value of the rate variable.

The hours worked are taken from the hours text field and converted into an integer. The wages calculation is used from Logic D to calculate the wages and it's displayed in the wages text area (positioned after a String).

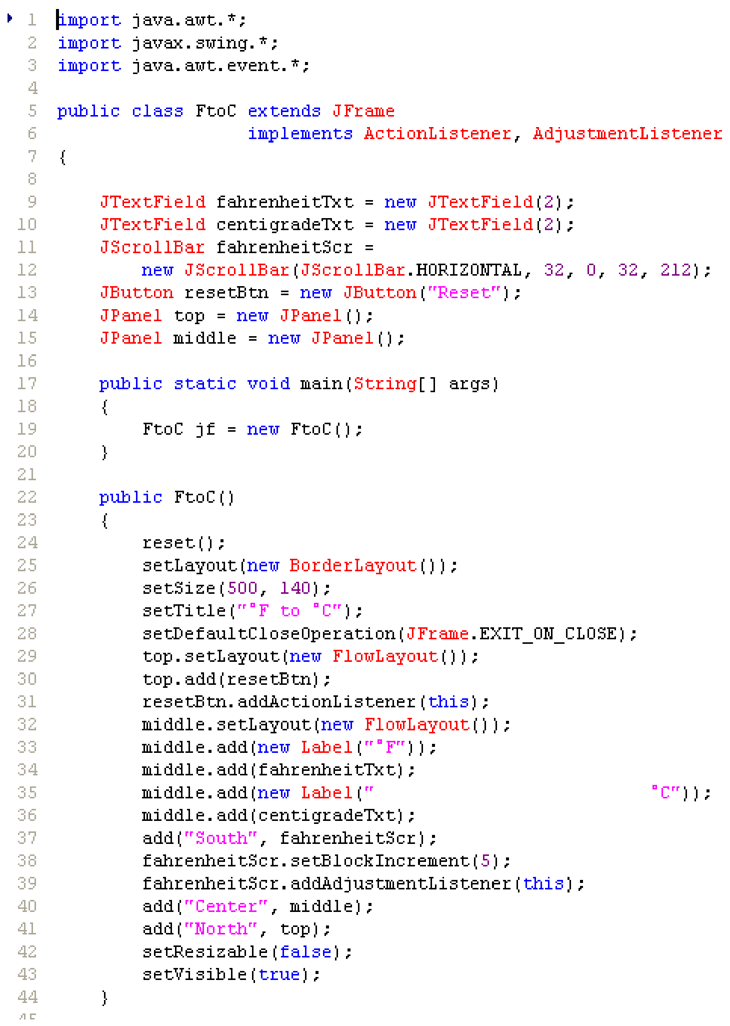
The section under "private void reset()" contains the values for which the text fields and the scroll bar reset to if the reset button is clicked. The "reset();" found just under the Action Performed calls upon this private reset function.

7.6 “FtoC.Java”

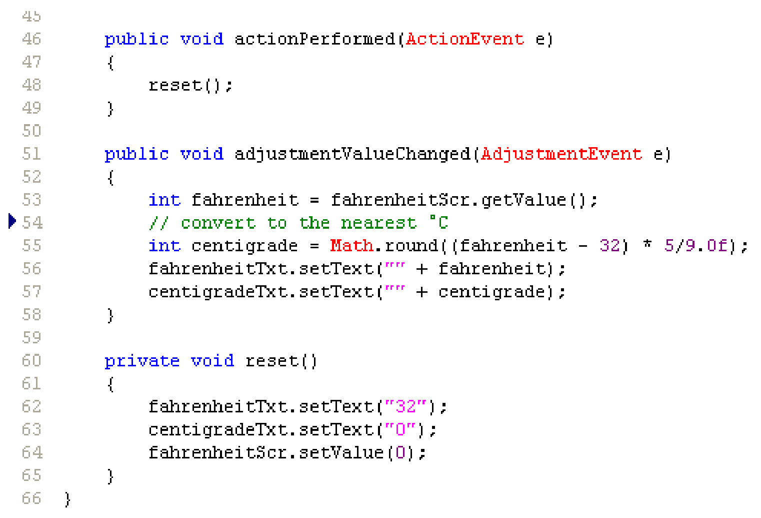
The code to “FtoC.java” is quite similar to the Logic E aside from the names. FtoC has a scroll bar which starts at a minimum of 32 and goes up to maximum of 212, this is represent Fahrenheit which is shown in the Fahrenheit text field.

Two text fields have also been created, one called “fahrenheitTxt” which will later display the Fahrenheit and the other called “centigradeTxt” which will later display the centigrade.

The lower part of the screenshot the right involves adding objects to the JFrame as well as setting a few of the JFrame properties.



```java
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class FtoC extends JFrame
                  implements ActionListener, AdjustmentListener
{

    JTextField fahrenheitTxt = new JTextField(2);
    JTextField centigradeTxt = new JTextField(2);
    JScrollBar fahrenheitScr =
        new JScrollBar(JScrollBar.HORIZONTAL, 32, 0, 32, 212);
    JButton resetBtn = new JButton("Reset");
    JPanel top = new JPanel();
    JPanel middle = new JPanel();

    public static void main(String[] args)
    {
        FtoC jf = new FtoC();
    }

    public FtoC()
    {
        reset();
        setLayout(new BorderLayout());
        setSize(500, 140);
        setTitle("°F to °C");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        top.setLayout(new FlowLayout());
        top.add(resetBtn);
        resetBtn.addActionListener(this);
        middle.setLayout(new FlowLayout());
        middle.add(new Label("°F"));
        middle.add(fahrenheitTxt);
        middle.add(new Label("                              °C"));
        middle.add(centigradeTxt);
        add("South", fahrenheitScr);
        fahrenheitScr.setBlockIncrement(5);
        fahrenheitScr.addAdjustmentListener(this);
        add("Center", middle);
        add("North", top);
        setResizable(false);
        setVisible(true);
    }
```



```java

    public void actionPerformed(ActionEvent e)
    {
        reset();
    }

    public void adjustmentValueChanged(AdjustmentEvent e)
    {
        int fahrenheit = fahrenheitScr.getValue();
        // convert to the nearest °C
        int centigrade = Math.round((fahrenheit - 32) * 5/9.0f);
        fahrenheitTxt.setText("" + fahrenheit);
        centigradeTxt.setText("" + centigrade);
    }

    private void reset()
    {
        fahrenheitTxt.setText("32");
        centigradeTxt.setText("0");
        fahrenheitScr.setValue(0);
    }
}
```

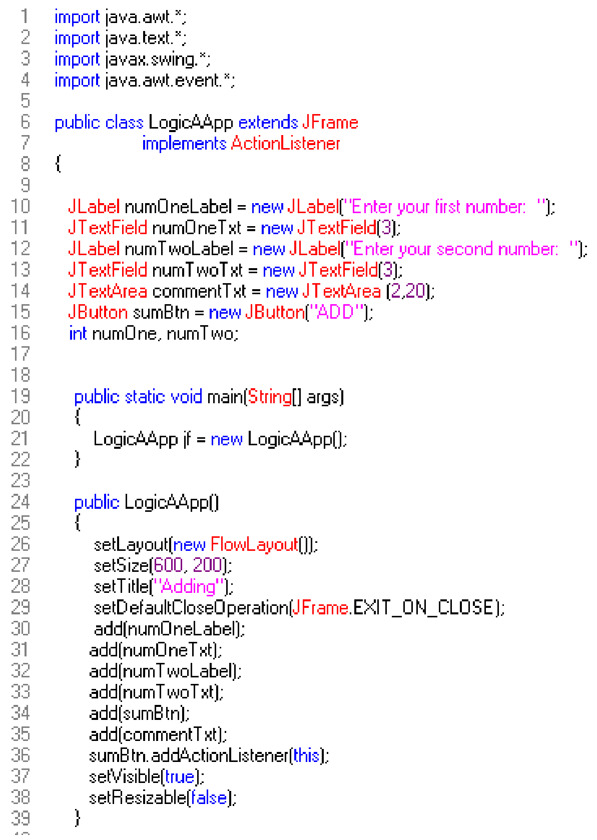
In the Action Performed section of the code, you can see that the integer variable Fahrenheit is basically getting the value from the Fahrenheit scroll bar, while the integer variable centigrade is converted from the Fahrenheit value given by the scroll bar.

Both the Fahrenheit and centigrade are displayed in the Fahrenheit and centigrade text fields respectively.

Similar to Logic E, the private reset function contains the default values of both the text fields and the scroll bar.

8 – Auxillary Classes

8.1 “LogicA Auxillary (a) .java”



```
import java.awt.*;
import java.text.*;
import javax.swing.*;
import java.awt.event.*;

public class LogicAApp extends JFrame
              implements ActionListener
{

  JLabel numOneLabel = new JLabel("Enter your first number:  ");
  JTextField numOneTxt = new JTextField(3);
  JLabel numTwoLabel = new JLabel("Enter your second number:  ");
  JTextField numTwoTxt = new JTextField(3);
  JTextArea commentTxt = new JTextArea (2,20);
  JButton sumBtn = new JButton("ADD");
  int numOne, numTwo;


   public static void main(String[] args)
   {
      LogicAApp jf = new LogicAApp();
   }

   public LogicAApp()
   {
      setLayout(new FlowLayout());
      setSize(600, 200);
      setTitle("Adding");
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      add(numOneLabel);
     add(numOneTxt);
     add(numTwoLabel);
     add(numTwoTxt);
     add(sumBtn);
     add(commentTxt);
     sumBtn.addActionListener(this);
     setVisible(true);
     setResizable(false);
   }
```

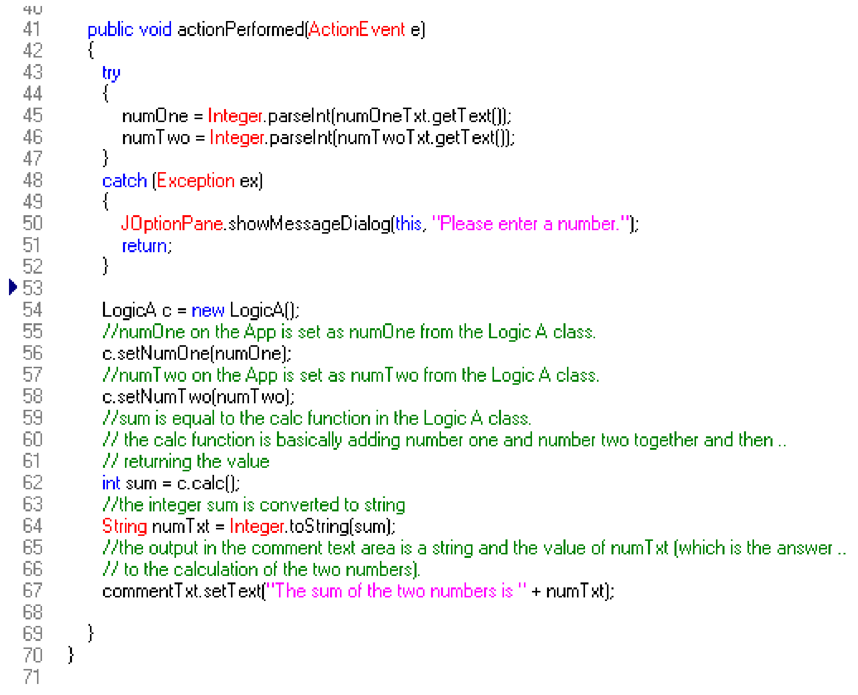
The image to the left is the code of the Logic A Application. It contains all the code for the GUI stuff that’s going to be used on the JFrame such as text fields, labels, text areas, buttons and other things.

It also contains the code for placing objects onto the JFrame, adding Action Listeners for the button and also whether the JFrame is visible and resizable.

The screenshot the right shows the code for the Action Performed section in the Logic A Application.

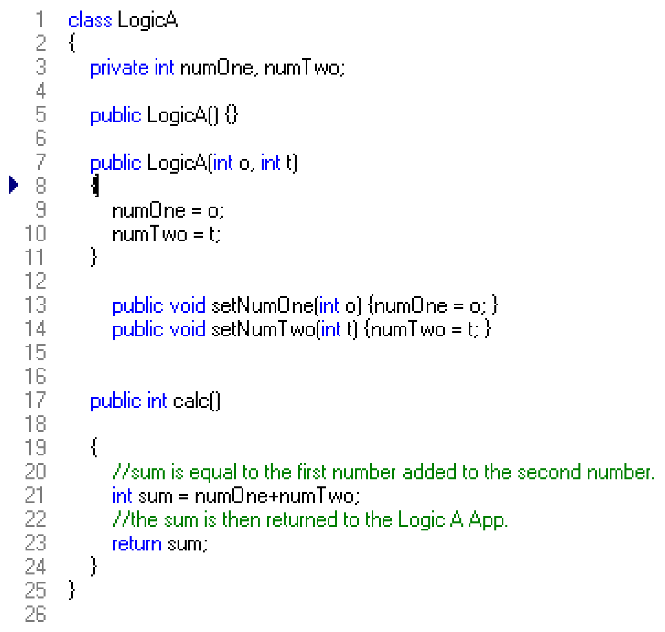
It basically sets the first number in the app as the first number in the auxiliary class and sets the second number in the app as the second number in the auxiliary class. It also sets the sum of both numbers as the calc function in the auxiliary class as well.

The sum is then converted to string and displayed in the text area.



```
   public void actionPerformed(ActionEvent e)
   {
     try
     {
        numOne = Integer.parseInt(numOneTxt.getText());
        numTwo = Integer.parseInt(numTwoTxt.getText());
     }
     catch (Exception ex)
     {
        JOptionPane.showMessageDialog(this, "Please enter a number.");
        return;
     }

     LogicA c = new LogicA();
     //numOne on the App is set as numOne from the Logic A class.
     c.setNumOne(numOne);
     //numTwo on the App is set as numTwo from the Logic A class.
     c.setNumTwo(numTwo);
     //sum is equal to the calc function in the Logic A class.
     // the calc function is basically adding number one and number two together and then ..
     // returning the value
     int sum = c.calc();
     //the integer sum is converted to string
     String numTxt = Integer.toString(sum);
     //the output in the comment text area is a string and the value of numTxt (which is the answer ..
     // to the calculation of the two numbers).
     commentTxt.setText("The sum of the two numbers is " + numTxt);

   }
}
```

Toby Gates – Computer Programming 1



```
class LogicA
{
   private int numOne, numTwo;

   public LogicA() {}

   public LogicA(int o, int t)
   {
      numOne = o;
      numTwo = t;
   }

      public void setNumOne(int o) {numOne = o; }
      public void setNumTwo(int t) {numTwo = t; }


   public int calc()

   {
      //sum is equal to the first number added to the second number.
      int sum = numOne+numTwo;
      //the sum is then returned to the Logic A App.
      return sum;
   }
}
```

The image to the left shows the code for the Auxiliary Class for Logic A. The Auxiliary Class takes care of the calculations that need to take place (in this case it's adding the first number with a second number).

Once the Auxiliary Class has finished with all the calculations it needs to do, it passes the solution of the calculation back to the Application to be displayed for the user to see on the JFrame.

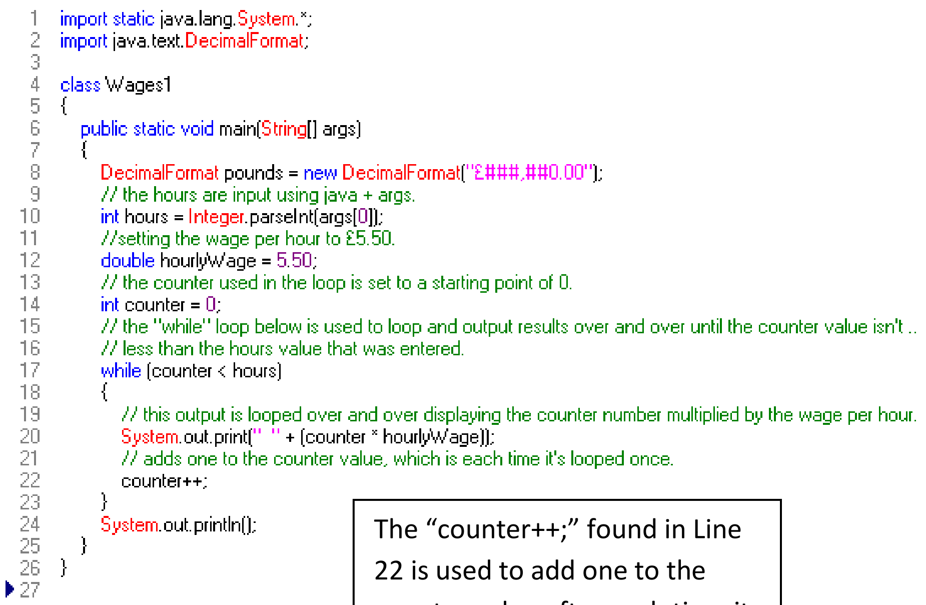
9 - Loops

9.1 “Wages1.java”

The image to the right shows the code for the “Wages1.java” program.

Java + Args are used in this program in order for the user to input the hours. The hourly wage is set as a double and given the value 5.50. The counter is set to the value of 0, because we want it to start at 0.

A while loop is then used to loop over and over until the counter isn’t less then the hours, the system then outputs the counter number multiplied by the hourly wage.

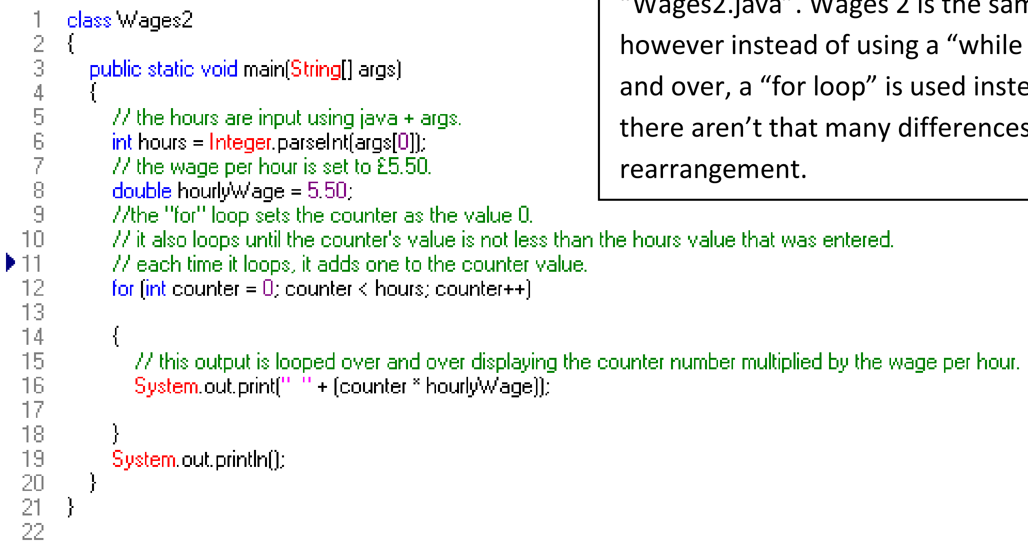


```
1   import static java.lang.System.*;
2   import java.text.DecimalFormat;
3   
4   class Wages1
5   {
6      public static void main(String[] args)
7      {
8         DecimalFormat pounds = new DecimalFormat("£###,##0.00");
9         // the hours are input using java + args.
10        int hours = Integer.parseInt(args[0]);
11        //setting the wage per hour to £5.50.
12        double hourlyWage = 5.50;
13        // the counter used in the loop is set to a starting point of 0.
14        int counter = 0;
15        // the "while" loop below is used to loop and output results over and over until the counter value isn't ..
16        // less than the hours value that was entered.
17        while (counter < hours)
18        {
19           // this output is looped over and over displaying the counter number multiplied by the wage per hour.
20           System.out.print("  " + (counter * hourlyWage));
21           // adds one to the counter value, which is each time it's looped once.
22           counter++;
23        }
24        System.out.println();
25     }
26  }
27
```

The “counter++;” found in Line 22 is used to add one to the counter value after each time it loops.

9.2 “Wages2.java”

The image below is an image of the code for “Wages2.java”. Wages 2 is the same as Wages 1 however instead of using a “while loop” to loop over and over, a “for loop” is used instead. As you can see there aren’t that many differences, only a bit of code rearrangement.



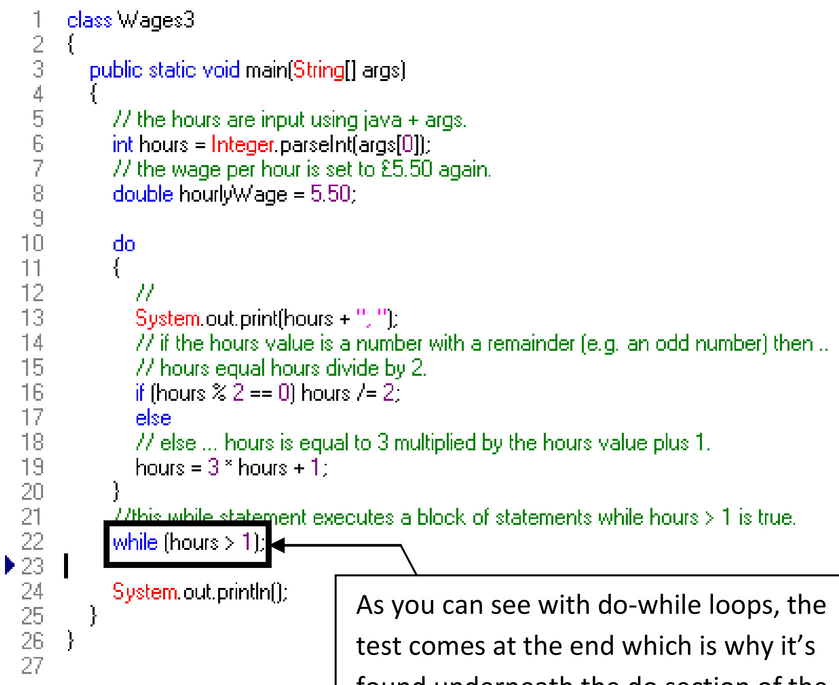
```
class Wages2
{
   public static void main(String[] args)
   {
      // the hours are input using java + args.
      int hours = Integer.parseInt(args[0]);
      // the wage per hour is set to £5.50.
      double hourlyWage = 5.50;
      //the "for" loop sets the counter as the value 0.
      // it also loops until the counter's value is not less than the hours value that was entered.
      // each time it loops, it adds one to the counter value.
      for (int counter = 0; counter < hours; counter++)

      {
         // this output is looped over and over displaying the counter number multiplied by the wage per hour.
         System.out.print("  " + (counter * hourlyWage));

      }
      System.out.println();
   }
}
```

9.3 "Wages3.java"

The code of "Wages 3.java" is the similar to Wages 1 and Wages 2 however this time it uses a "do-while" loop.

For the do section of the loop, you can see that the system outputs the hours + a comma string.

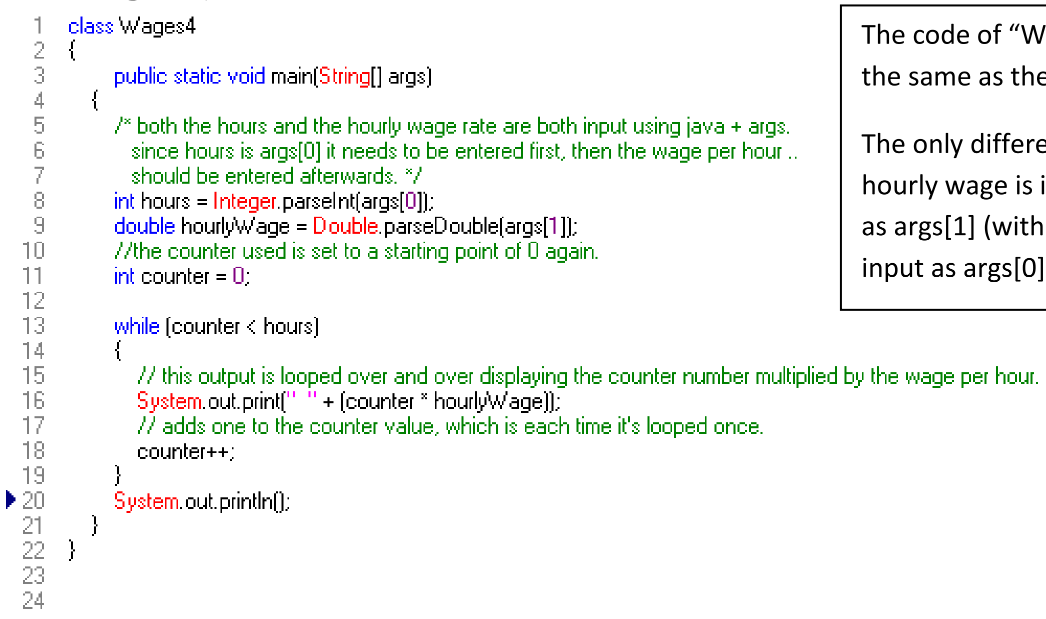
To determine the hours an if function is used. If the hours are a number with a remainder then "the hours are equal to the hours divided by 2". If it's not a number with a remainder then "the hours are equal to 3 which are then multiplied by the hour value + 1".



```
class Wages3
{
  public static void main(String[] args)
  {
    // the hours are input using java + args.
    int hours = Integer.parseInt(args[0]);
    // the wage per hour is set to £5.50 again.
    double hourlyWage = 5.50;

    do
    {
      //
      System.out.print(hours + ", ");
      // if the hours value is a number with a remainder (e.g. an odd number) then ..
      // hours equal hours divide by 2.
      if (hours % 2 == 0) hours /= 2;
      else
      // else ... hours is equal to 3 multiplied by the hours value plus 1.
      hours = 3 * hours + 1;
    }
    //this while statement executes a block of statements while hours > 1 is true.
    while (hours > 1);

    System.out.println();
  }
}
```

As you can see with do-while loops, the test comes at the end which is why it's found underneath the do section of the loop.

9.4 "Wages4.java"

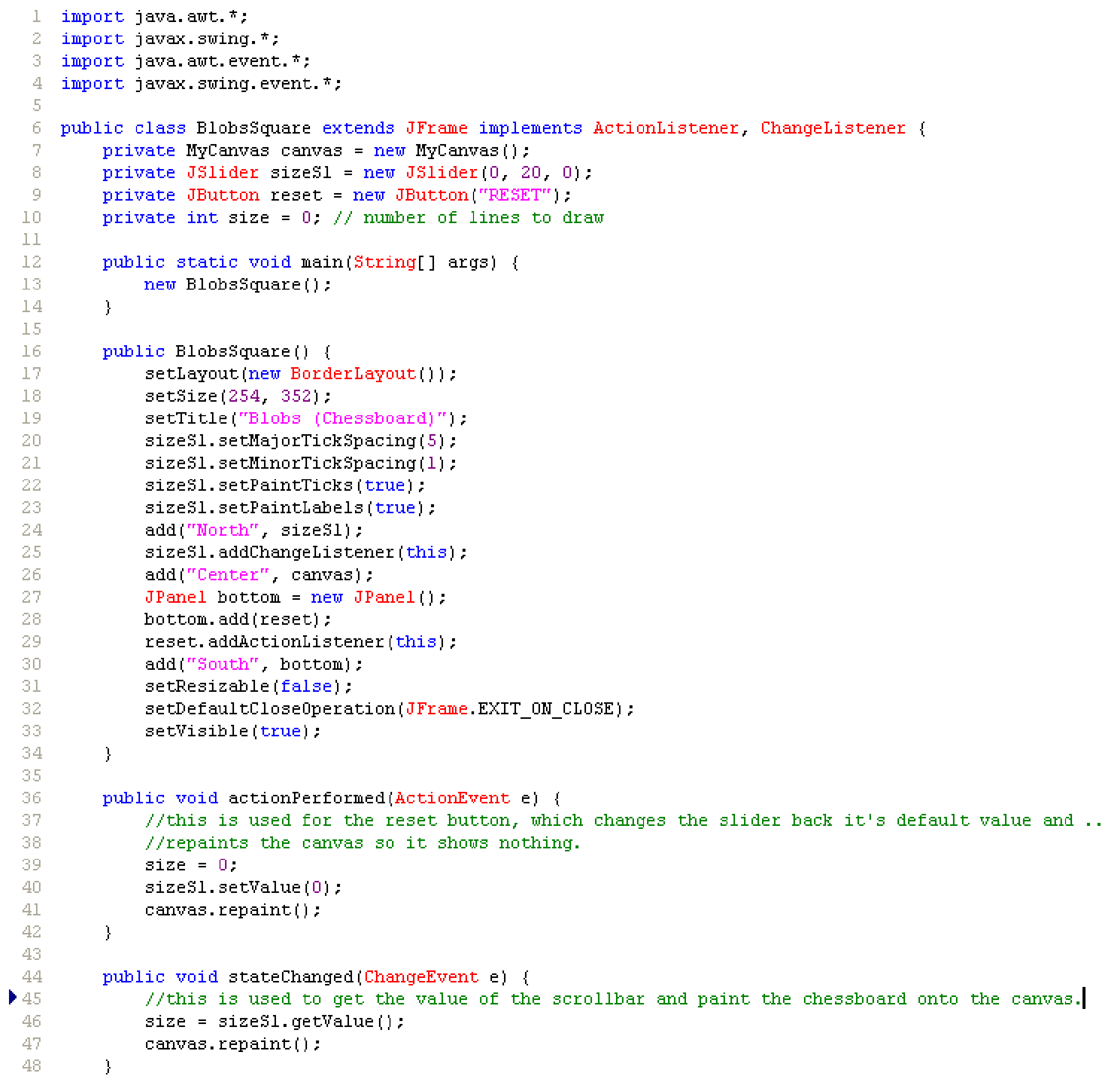


```
class Wages4
{
    public static void main(String[] args)
  {
    /* both the hours and the hourly wage rate are both input using java + args.
       since hours is args[0] it needs to be entered first, then the wage per hour ..
       should be entered afterwards. */
    int hours = Integer.parseInt(args[0]);
    double hourlyWage = Double.parseDouble(args[1]);
    //the counter used is set to a starting point of 0 again.
    int counter = 0;

    while (counter < hours)
    {
      // this output is looped over and over displaying the counter number multiplied by the wage per hour.
      System.out.print("  " + (counter * hourlyWage));
      // adds one to the counter value, which is each time it's looped once.
      counter++;
    }
    System.out.println();
  }
}
```

The code of "Wages4.java" is exactly the same as the code from Wages 1.

The only difference is that the hourly wage is input after the hours as args[1] (with the hours still being input as args[0]).

9.5 “BlobsSquare.java”



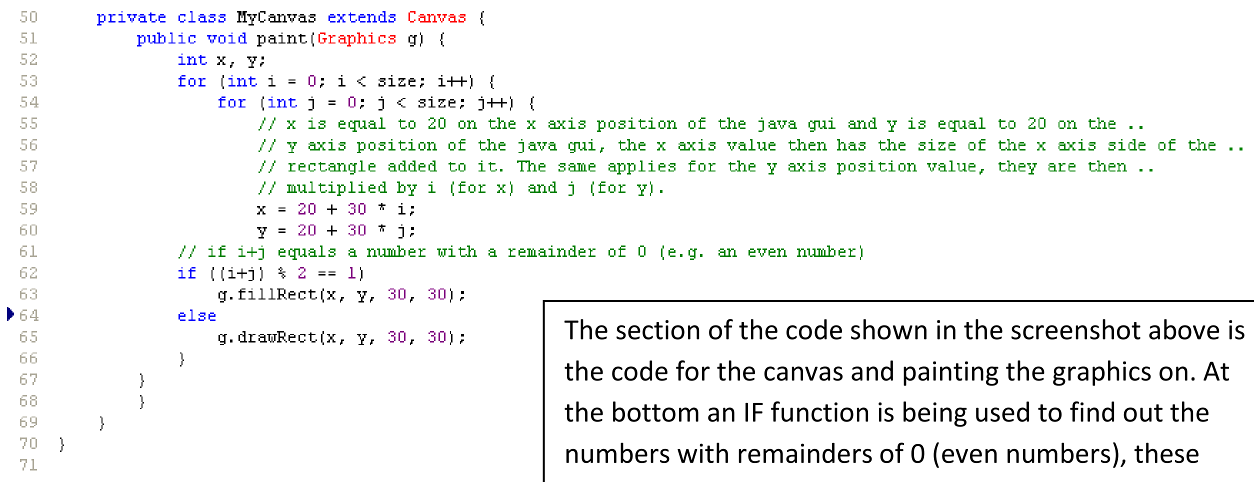
```
 1 import java.awt.*;
 2 import javax.swing.*;
 3 import java.awt.event.*;
 4 import javax.swing.event.*;
 5
 6 public class BlobsSquare extends JFrame implements ActionListener, ChangeListener {
 7     private MyCanvas canvas = new MyCanvas();
 8     private JSlider sizeSl = new JSlider(0, 20, 0);
 9     private JButton reset = new JButton("RESET");
10     private int size = 0; // number of lines to draw
11
12     public static void main(String[] args) {
13         new BlobsSquare();
14     }
15
16     public BlobsSquare() {
17         setLayout(new BorderLayout());
18         setSize(254, 352);
19         setTitle("Blobs (Chessboard)");
20         sizeSl.setMajorTickSpacing(5);
21         sizeSl.setMinorTickSpacing(1);
22         sizeSl.setPaintTicks(true);
23         sizeSl.setPaintLabels(true);
24         add("North", sizeSl);
25         sizeSl.addChangeListener(this);
26         add("Center", canvas);
27         JPanel bottom = new JPanel();
28         bottom.add(reset);
29         reset.addActionListener(this);
30         add("South", bottom);
31         setResizable(false);
32         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
33         setVisible(true);
34     }
35
36     public void actionPerformed(ActionEvent e) {
37         //this is used for the reset button, which changes the slider back it's default value and ..
38         //repaints the canvas so it shows nothing.
39         size = 0;
40         sizeSl.setValue(0);
41         canvas.repaint();
42     }
43
44     public void stateChanged(ChangeEvent e) {
45         //this is used to get the value of the scrollbar and paint the chessboard onto the canvas.
46         size = sizeSl.getValue();
47         canvas.repaint();
48     }
```

The image to the left shows the code of “BlobsSquare.java”. As you can see most of it is creating buttons, sliders (another type of scroll bar) and other objects and adding them to the JPanels created on the JFrame.

Line 7 is the creation of a new canvas on the JFrame. Line 10 is the declaration of the integer variable called “size” which helps determine the number of lines to draw.

When Line 36 of the code is reached you come to the Action Performed section, in here the size is set to 0 and the value for the size slider is set as 0. The canvas is then repainted to the values of 0 (This is used for when the reset button is clicked).

When Line 44 of the code is reached you get up to the State Changed part of the code, this is when the program gets the value of the slider after it’s state has been changed by the user and repaints the chessboard on the canvas depending on what the value of the slider was.



```
    private class MyCanvas extends Canvas {
        public void paint(Graphics g) {
            int x, y;
            for (int i = 0; i < size; i++) {
                for (int j = 0; j < size; j++) {
                    // x is equal to 20 on the x axis position of the java gui and y is equal to 20 on the ..
                    // y axis position of the java gui, the x axis value then has the size of the x axis side of the ..
                    // rectangle added to it. The same applies for the y axis position value, they are then ..
                    // multiplied by i (for x) and j (for y).
                    x = 20 + 30 * i;
                    y = 20 + 30 * j;
            // if i+j equals a number with a remainder of 0 (e.g. an even number)
            if ((i+j) % 2 == 1)
                g.fillRect(x, y, 30, 30);
            else
                g.drawRect(x, y, 30, 30);
            }
        }
        }
    }
}
```

The section of the code shown in the screenshot above is the code for the canvas and painting the graphics on. At the bottom an IF function is being used to find out the numbers with remainders of 0 (even numbers), these even numbers have their rectangles filled in black while the other rectangles (the odd numbers) are left white to make it look like a chessboard.

Term 2

11 – Arrays

11.1 "Quick and Not So Quick Exercise Review"

See: Introduction to Programming in Java (Part 2), Pages 15 – 17.

11.2 "Bubble Sort Trace Table"

i	j	Swap?	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	
			3	21	31	8	29	4	
5	0	**N**	3	21	31	8	29	4	
5	1	**N**	3	21	31	8	29	4	
5	2	**Y**	3	21	8	31	29	4	
5	3	**Y**	3	21	8	29	31	4	
5	4	**Y**	3	21	8	29	4	31	End of 1st pass
4	0	**N**	3	21	8	29	4	31	
4	1	**Y**	3	8	21	29	4	31	
4	2	**N**	3	8	21	29	4	31	
4	3	**Y**	3	8	21	4	29	31	End of 2nd pass
3	0	**N**	3	8	21	4	29	31	
3	1	**N**	3	8	21	4	29	31	
3	2	**Y**	3	8	4	21	29	31	End of 3rd pass
2	0	**N**	3	8	4	21	29	31	
2	1	**Y**	3	4	8	21	29	31	End of 4th pass
1	0	**N**	3	4	8	21	29	31	End of 5th pass

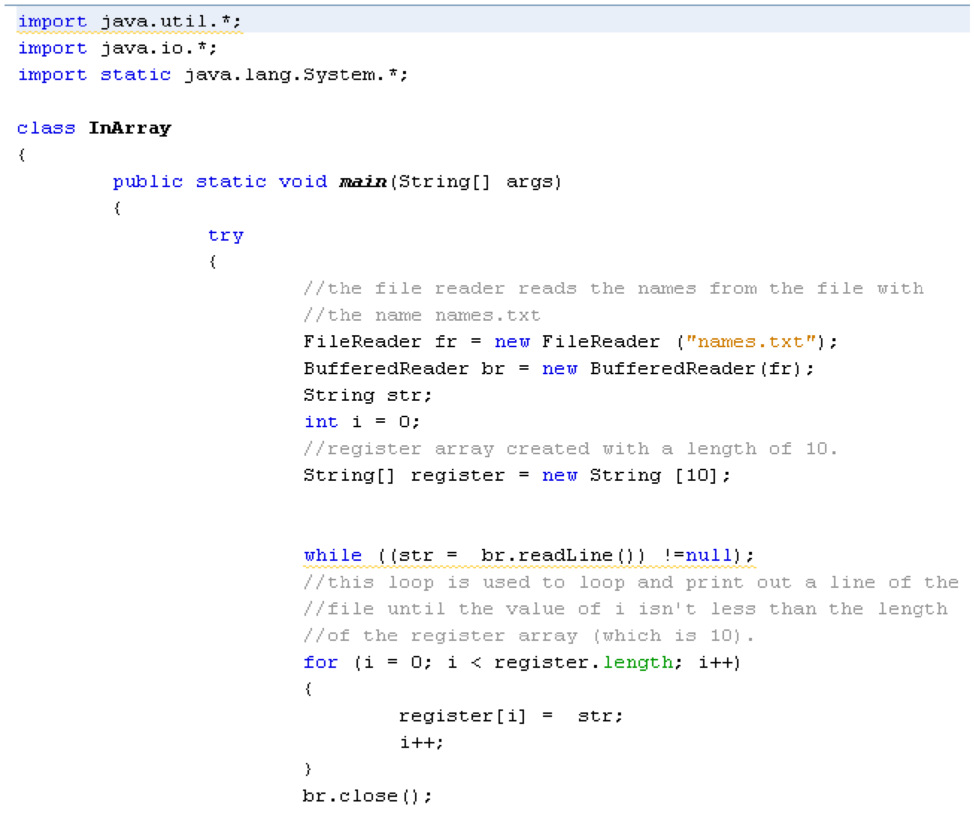
Above is the Bubble Sort Trace Table. It shows how data is rearranged in order as the Array is going through a Bubble Sort. This is done by moving the highest number encounted in the Array towards the end of the Array, thus sort the numbers and making the larger numbers appear at the end of the Array. The smaller numbers are then placed towards the start of the array because the larger numbers are being moved towards the end of it.

12 – Exceptions and Files

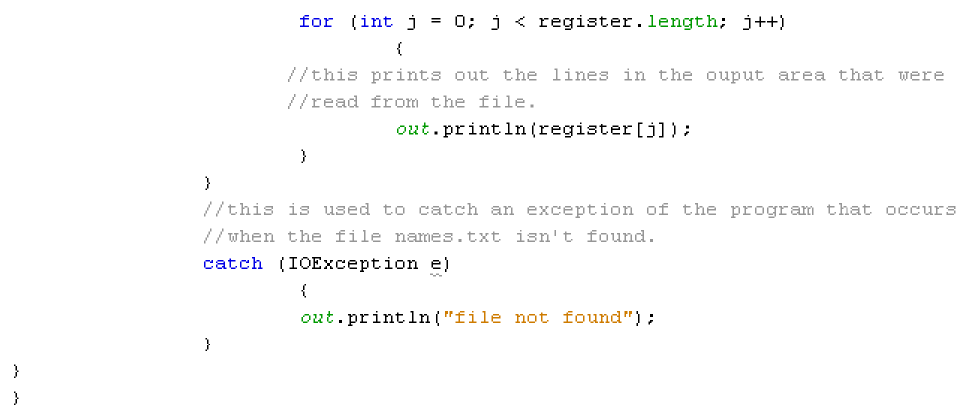
12.1 “InArray.java”

This screenshot to the left shows the file “InArray.java”.

It shows the code for using a FileReader to read text files as well as the loop used to loop and print out a line of the file until the register array is full.



```
import java.util.*;
import java.io.*;
import static java.lang.System.*;

class InArray
{
        public static void main(String[] args)
        {
                try
                {
                        //the file reader reads the names from the file with
                        //the name names.txt
                        FileReader fr = new FileReader ("names.txt");
                        BufferedReader br = new BufferedReader(fr);
                        String str;
                        int i = 0;
                        //register array created with a length of 10.
                        String[] register = new String [10];


                        while ((str =  br.readLine())  !=null);
                        //this loop is used to loop and print out a line of the
                        //file until the value of i isn't less than the length
                        //of the register array (which is 10).
                        for (i = 0; i < register.length; i++)
                        {
                                register[i]  =  str;
                                i++;
                        }
                        br.close();
```

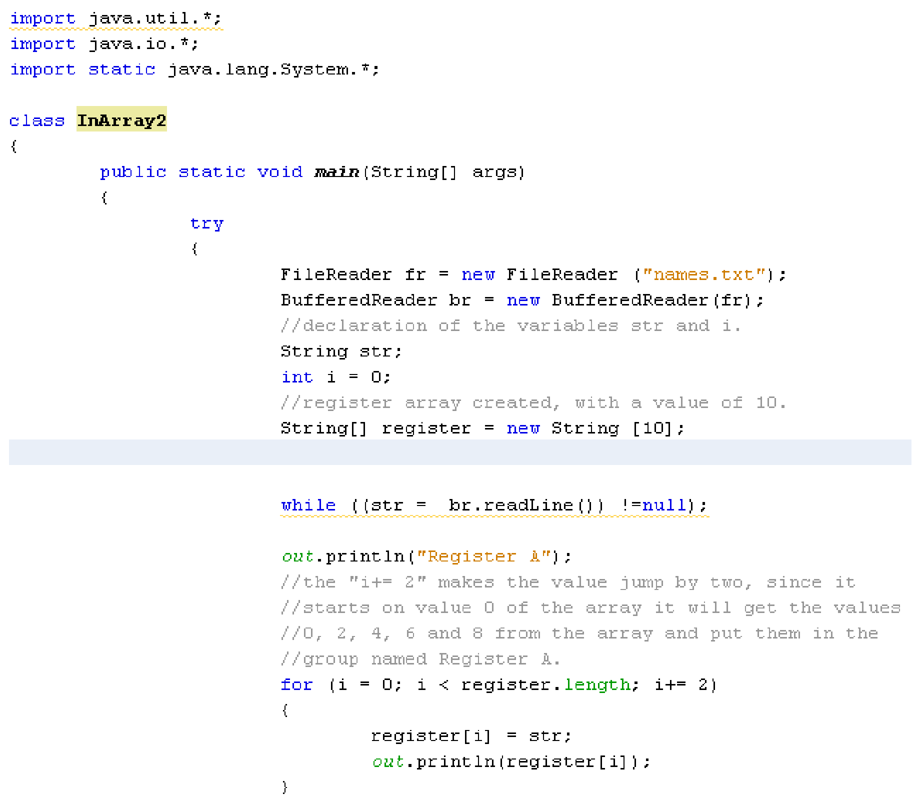


```
                         for (int j = 0; j < register.length; j++)
                                 {
                        //this prints out the lines in the ouput area that were
                        //read from the file.
                                 out.println(register[j]);
                         }
                }
                //this is used to catch an exception of the program that occurs
                //when the file names.txt isn't found.
                catch (IOException e)
                         {
                         out.println("file not found");
                }
        }
}
```

This section of the code for “InArray.java” includes another loop which prints the lines that were read from the file out in the output area.

It also contains a “catch” that catches Input/Output exceptions that are thrown.

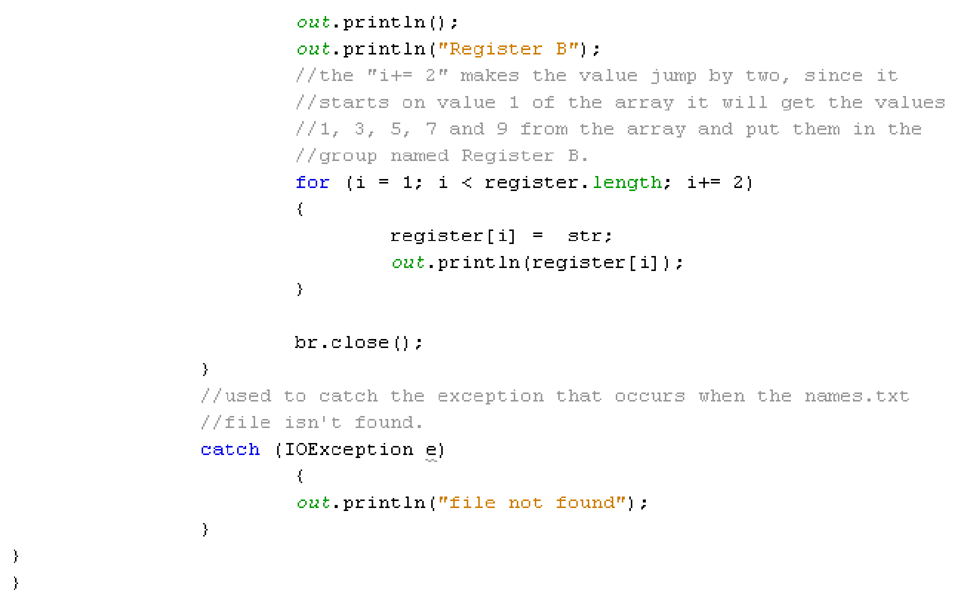
12.2 “InArray2.java”



```
import java.util.*;
import java.io.*;
import static java.lang.System.*;

class InArray2
{
        public static void main(String[] args)
        {
                try
                {
                        FileReader fr = new FileReader ("names.txt");
                        BufferedReader br = new BufferedReader(fr);
                        //declaration of the variables str and i.
                        String str;
                        int i = 0;
                        //register array created, with a value of 10.
                        String[] register = new String [10];


                        while ((str =  br.readLine()) !=null);

                        out.println("Register A");
                        //the "i+= 2" makes the value jump by two, since it
                        //starts on value 0 of the array it will get the values
                        //0, 2, 4, 6 and 8 from the array and put them in the
                        //group named Register A.
                        for (i = 0; i < register.length; i+= 2)
                        {
                                register[i] = str;
                                out.println(register[i]);
                        }
```

The screenshot to the left shows “InArray2.java”.

This time instead of just adding the data to the Array, the even lines of the file are added. They are then listed under the heading “Register A” and are printed out in the system output area.

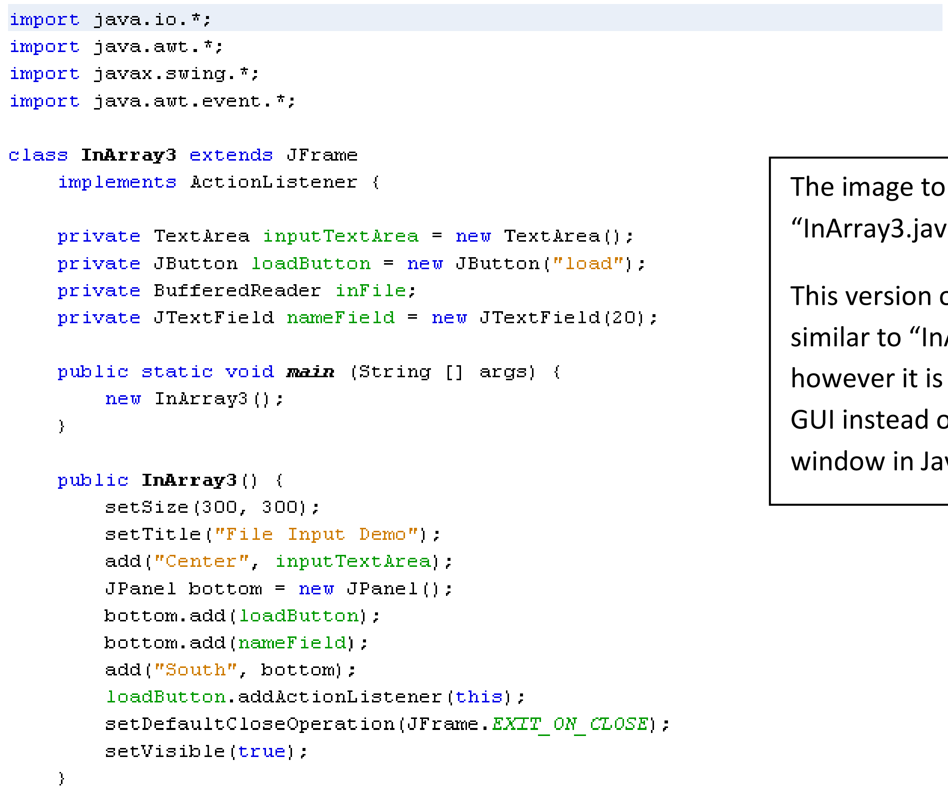


```
                         out.println();
                         out.println("Register B");
                         //the "i+= 2" makes the value jump by two, since it
                         //starts on value 1 of the array it will get the values
                         //1, 3, 5, 7 and 9 from the array and put them in the
                         //group named Register B.
                         for (i = 1; i < register.length; i+= 2)
                         {
                                 register[i] =  str;
                                 out.println(register[i]);
                         }

                         br.close();
                }
                //used to catch the exception that occurs when the names.txt
                //file isn't found.
                catch (IOException e)
                         {
                         out.println("file not found");
                }
}
}
```

This section of the code in “InArray2.java” shows the odd lines of the file added under the heading “Register B”.

There’s also a “catch” to catch the Input/Output error “File not Found” which may occur from the code inside the “try”.

12.3 "InArray3.java"



```
import java.io.*;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

class InArray3 extends JFrame
    implements ActionListener {

    private TextArea inputTextArea = new TextArea();
    private JButton loadButton = new JButton("load");
    private BufferedReader inFile;
    private JTextField nameField = new JTextField(20);

    public static void main (String [] args) {
        new InArray3();
    }

    public InArray3() {
        setSize(300, 300);
        setTitle("File Input Demo");
        add("Center", inputTextArea);
        JPanel bottom = new JPanel();
        bottom.add(loadButton);
        bottom.add(nameField);
        add("South", bottom);
        loadButton.addActionListener(this);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }
```

The image to the left shows "InArray3.java".

This version of the file is similar to "InArray2.java" however it is done with a GUI instead of the output window in Java.

The code for reading the file in InArray3 is found under the actionPerformed() section.

A Buffered Reader is used to read the contents of the text file, it provides the efficient reading of the file unlike the normal File Reader.

An array is then created named register (like it is in “InArray.java” and “InArray2.java”) which is used to store the lines of the text file which are read in.

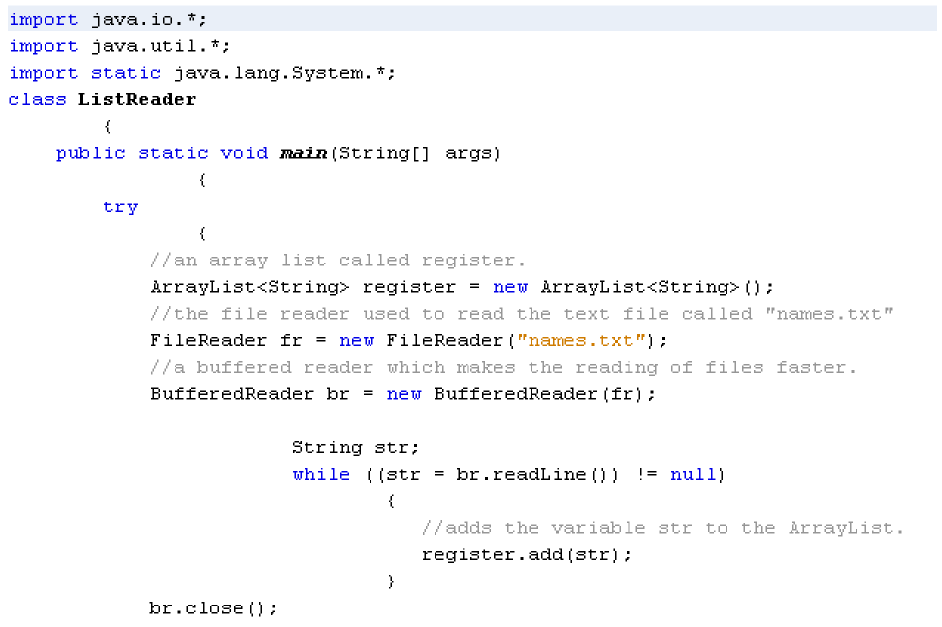
Two loops are then used to split the data , the even lines of the text file read in by the Array are output in the text area with the text “Coursework A will be done by:” while the odd lines of the text file that was read in by the Array are output in the text area with the text “Coursework B will be done by:”.

There is still also a “catch” to catch the “file not found” Input/Output error.

```
public void actionPerformed(ActionEvent evt) {
    if (evt.getSource() == loadButton) {
        String fileName;
        fileName = nameField.getText();
        try {
            //inFile is the name of the buffered reader that speeds up
            //the file reader which is reading the file.
            inFile = new BufferedReader(new FileReader(fileName));
            //
            inputTextArea.setText(""); // clear the input area
            //variables
            String name;
            int i = 0;
            //the created array called register.
            String[] register = new String[10];
            while ((name = inFile.readLine()) != null) {
                //the 0 value of the register array is equal to the ..
                //variable called name which reads the lines with the ..
                //file reader.
                register[i] = name;
                // i++ adds 1 to i.
                i++;
            }
            //text displayed in the text area with a line break.
            inputTextArea.append("Coursework A will be done by:\n");
            //
            for (int j = 0; j < (register.length / 2); j++) {
                //the text area displays the values of the register array
                //starting with j (0) and multiplying it by 2, which
                //displays the even values of the array.
                inputTextArea.append(register[2 * j] + "\n");
            }
            // \n = line break
            inputTextArea.append("\n");
            //more text displayed in the text area with a line break.
            inputTextArea.append("Coursework B will be done by:\n");
            //
            for (int k = 0; k < (register.length) / 2; k++) {
                //the text area displays the values of the register array
                //starting with k (0) multiplied by 2 and adding 1 which
                //displays the odd values of the array.
                inputTextArea.append(register[(2 * k) + 1] + "\n");
            }

           inFile.close();


        }
        catch (IOException e) {
            JOptionPane.showMessageDialog(this, "File not found");
            nameField.setText("");
        }
    }
}
}
```

13 – Collections

13.1 "ListReader.java"



```
import java.io.*;
import java.util.*;
import static java.lang.System.*;
class ListReader
        {
    public static void main(String[] args)
                {
        try
                {
            //an array list called register.
            ArrayList<String> register = new ArrayList<String>();
            //the file reader used to read the text file called "names.txt"
            FileReader fr = new FileReader("names.txt");
            //a buffered reader which makes the reading of files faster.
            BufferedReader br = new BufferedReader(fr);

                        String str;
                        while ((str = br.readLine()) != null)
                                {
                                   //adds the variable str to the ArrayList.
                                   register.add(str);
                                }
            br.close();
```

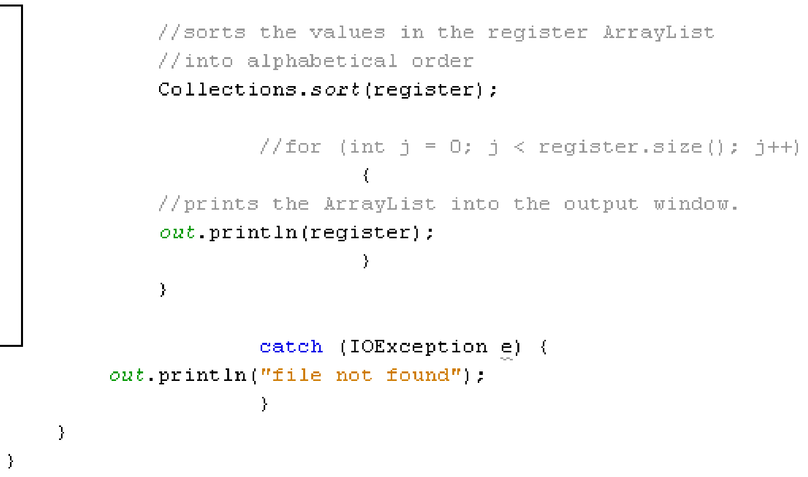
The screenshot to the left shows the program "ListReader.java".

An Array List called register is created along with a File Reader to read the text file called "names.txt".

The lines that are read from the file are then added to the register Array List.

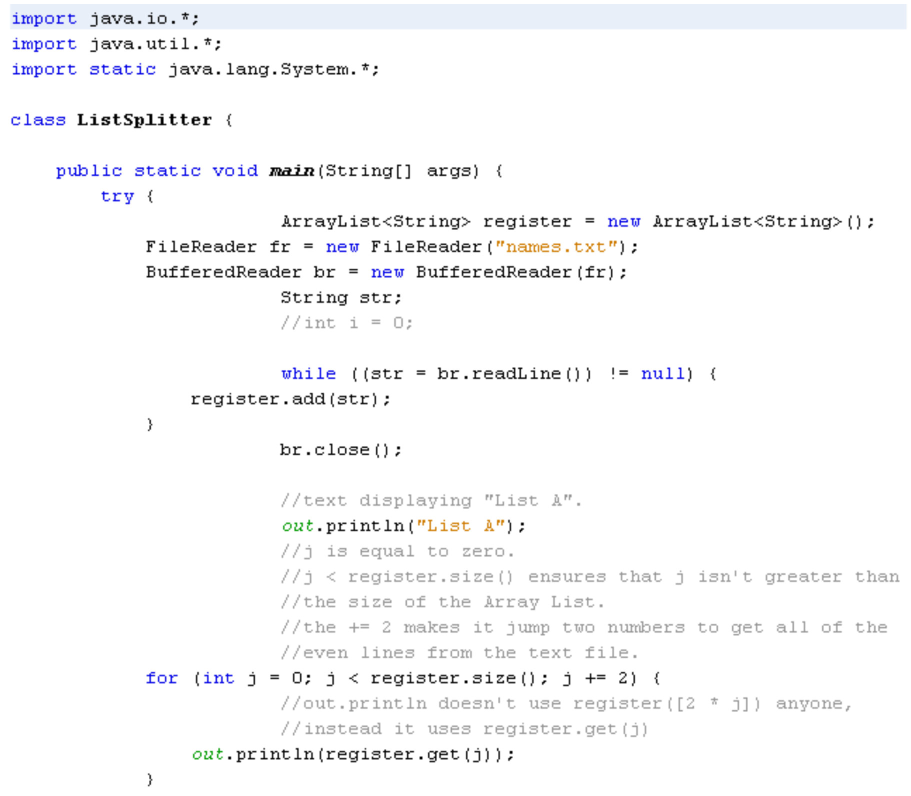
Collections.sort is used to sort the values of the Array List into alphabetical order.

The "System.out.println" is used to print out the sorted register in the output window.



```
            //sorts the values in the register ArrayList
            //into alphabetical order
            Collections.sort(register);

                    //for (int j = 0; j < register.size(); j++)
                            {
            //prints the ArrayList into the output window.
            out.println(register);
                            }
            }

                    catch (IOException e) {
        out.println("file not found");
                    }
        }
    }
```

13.2 “ListSplitter.java”



```
import java.io.*;
import java.util.*;
import static java.lang.System.*;

class ListSplitter {

    public static void main(String[] args) {
        try {
                        ArrayList<String> register = new ArrayList<String>();
            FileReader fr = new FileReader("names.txt");
            BufferedReader br = new BufferedReader(fr);
                        String str;
                        //int i = 0;

                        while ((str = br.readLine()) != null) {
                register.add(str);
            }
                        br.close();

                        //text displaying "List A".
                        out.println("List A");
                        //j is equal to zero.
                        //j < register.size() ensures that j isn't greater than
                        //the size of the Array List.
                        //the += 2 makes it jump two numbers to get all of the
                        //even lines from the text file.
            for (int j = 0; j < register.size(); j += 2) {
                        //out.println doesn't use register([2 * j]) anyone,
                        //instead it uses register.get(j)
                out.println(register.get(j));
            }
```

The screenshot to the left shows the program “ListSplitter.java”. This also has an Array List by the name of register as well as a file reader.

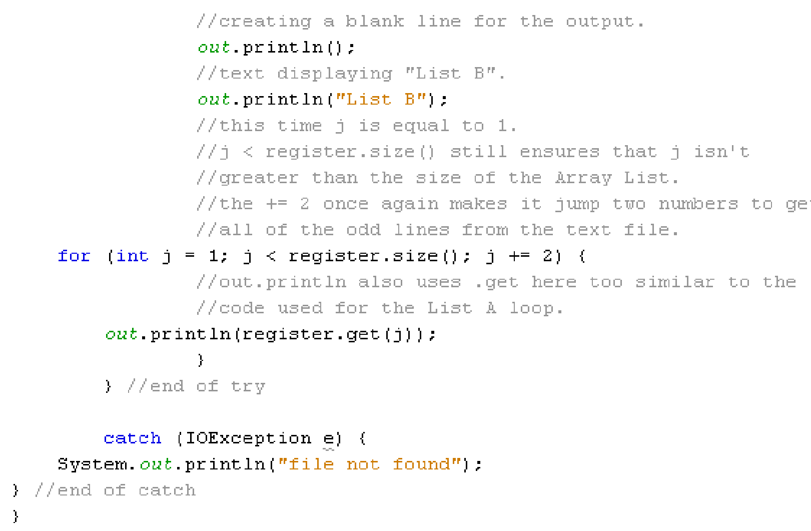
The file is read in the same way too; however it is printed out in a different way. The first list (List A) displays all the even lines in the text file, so “j+= 2” is used instead of “j++” to jump two numbers instead of increase by 1.

This is then output with “System.out.println”.

“System.out.println()” is used to create a blank line to be displayed in the output.

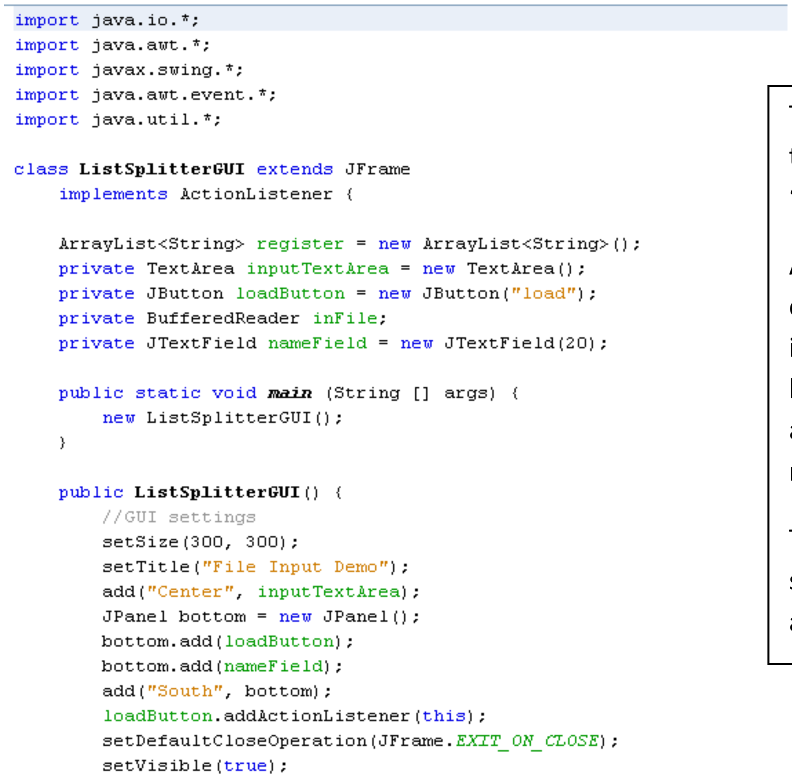
List B is done for the odd numbers, this time however the value used was “j = 1” in the for loop instead of “j = 0” in order to get the odd numbered lines from the text file.

It is then output with the use of “System.out.println”.



```
                //creating a blank line for the output.
                out.println();
                //text displaying "List B".
                out.println("List B");
                //this time j is equal to 1.
                //j < register.size() still ensures that j isn't
                //greater than the size of the Array List.
                //the += 2 once again makes it jump two numbers to get
                //all of the odd lines from the text file.
    for (int j = 1; j < register.size(); j += 2) {
                //out.println also uses .get here too similar to the
                //code used for the List A loop.
        out.println(register.get(j));
                }
        } //end of try

        catch (IOException e) {
    System.out.println("file not found");
} //end of catch
}
```

13.3 "ListSplitterGUI.java"

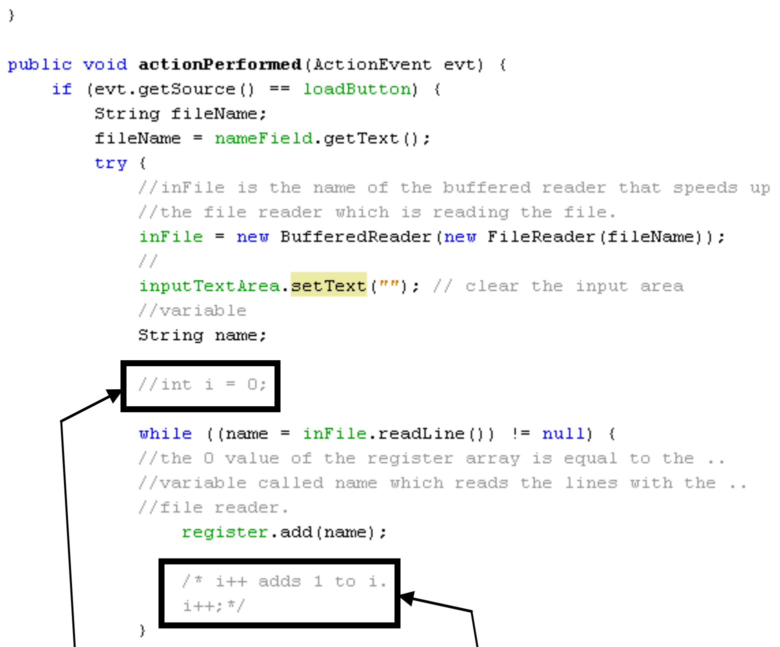


```java
import java.io.*;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import java.util.*;

class ListSplitterGUI extends JFrame
    implements ActionListener {

    ArrayList<String> register = new ArrayList<String>();
    private TextArea inputTextArea = new TextArea();
    private JButton loadButton = new JButton("load");
    private BufferedReader inFile;
    private JTextField nameField = new JTextField(20);

    public static void main (String [] args) {
        new ListSplitterGUI();
    }

    public ListSplitterGUI() {
        //GUI settings
        setSize(300, 300);
        setTitle("File Input Demo");
        add("Center", inputTextArea);
        JPanel bottom = new JPanel();
        bottom.add(loadButton);
        bottom.add(nameField);
        add("South", bottom);
        loadButton.addActionListener(this);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
```

The screenshot to the left shows the program "ListSplitterGUI.java".

As you can see it has an ArrayList called register, a text area called inputTextArea, a JButton called loadButton, a buffered reader and a JTextField called nameField.

The other stuff in general GUI stuff that's set such as JPanels and setTitle etc.

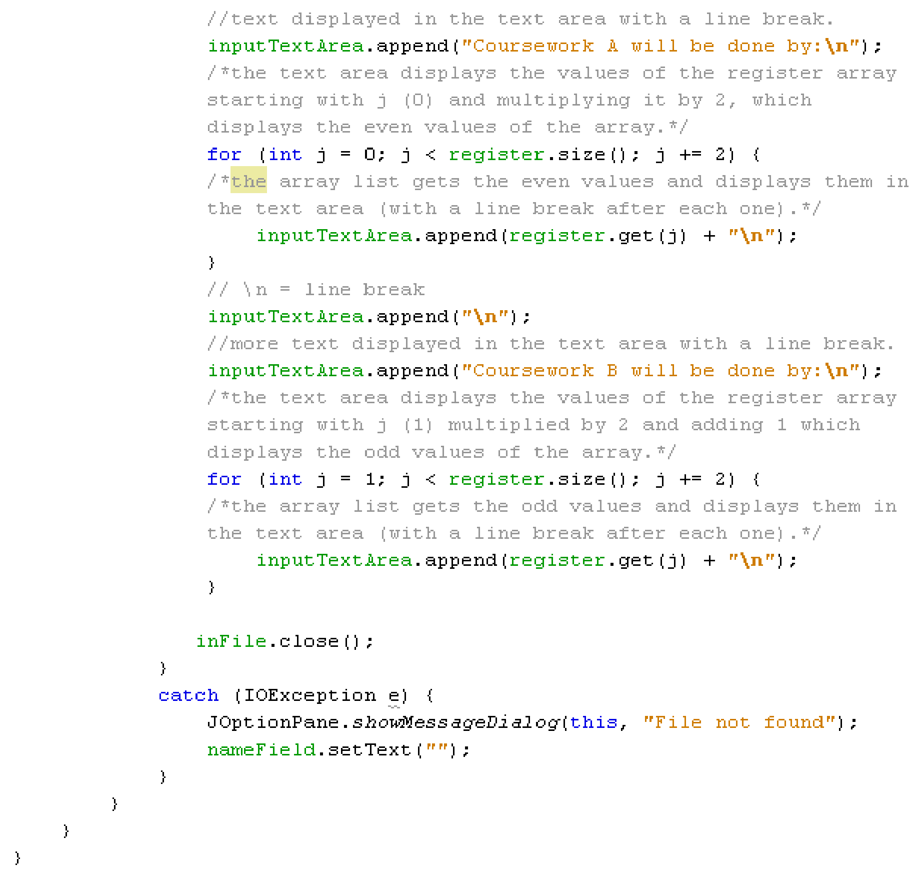


```java
}

public void actionPerformed(ActionEvent evt) {
    if (evt.getSource() == loadButton) {
        String fileName;
        fileName = nameField.getText();
        try {
            //inFile is the name of the buffered reader that speeds up
            //the file reader which is reading the file.
            inFile = new BufferedReader(new FileReader(fileName));
            //
            inputTextArea.setText(""); // clear the input area
            //variable
            String name;

            //int i = 0;

            while ((name = inFile.readLine()) != null) {
            //the 0 value of the register array is equal to the ..
            //variable called name which reads the lines with the ..
            //file reader.
                register.add(name);

                /* i++ adds 1 to i.
                i++;*/
            }
```

The screenshot to the right shows the actionPerformed part of the code. It shows that if the Load Button is clicked on then the String variable fileName gets the value from the JTextField (called nameField).

The content of the file that has been read is then added to the Array List called register.

The un-used code found in the program. (It has been commented out.)



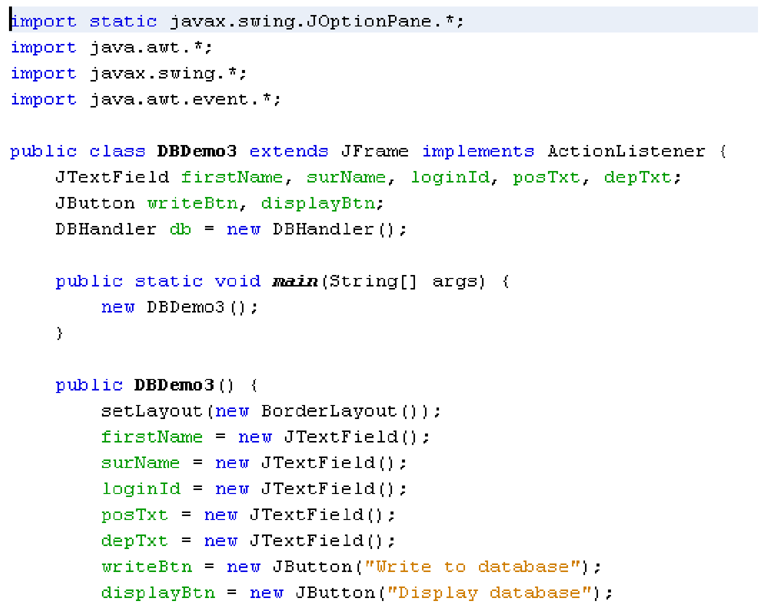
```
                //text displayed in the text area with a line break.
                inputTextArea.append("Coursework A will be done by:\n");
                /*the text area displays the values of the register array
                starting with j (0) and multiplying it by 2, which
                displays the even values of the array.*/
                for (int j = 0; j < register.size(); j += 2) {
                /*the array list gets the even values and displays them in
                the text area (with a line break after each one).*/
                    inputTextArea.append(register.get(j) + "\n");
                }
                // \n = line break
                inputTextArea.append("\n");
                //more text displayed in the text area with a line break.
                inputTextArea.append("Coursework B will be done by:\n");
                /*the text area displays the values of the register array
                starting with j (1) multiplied by 2 and adding 1 which
                displays the odd values of the array.*/
                for (int j = 1; j < register.size(); j += 2) {
                /*the array list gets the odd values and displays them in
                the text area (with a line break after each one).*/
                    inputTextArea.append(register.get(j) + "\n");
                }

               inFile.close();
            }
            catch (IOException e) {
                JOptionPane.showMessageDialog(this, "File not found");
                nameField.setText("");
            }
        }
    }
}
```

The screenshot to the left shows how the output is laid out, with even numbers underneath “Coursework A will be done by:” and odd numbers underneath “Coursework B will be done by:”.

It also shows a catch method for an Input/output exception.

15 – Databases

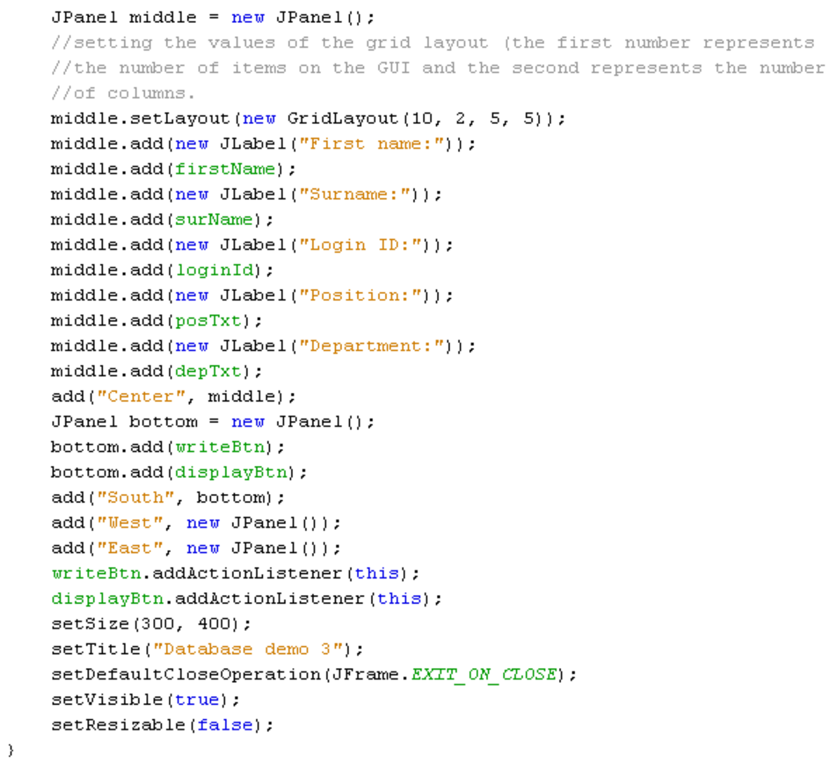
15.1 Connecting to a Database (“DBDemo3 .java” and “DBHandler .java”)



```java
import static javax.swing.JOptionPane.*;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class DBDemo3 extends JFrame implements ActionListener {
    JTextField firstName, surName, loginId, posTxt, depTxt;
    JButton writeBtn, displayBtn;
    DBHandler db = new DBHandler();

    public static void main(String[] args) {
        new DBDemo3();
    }

    public DBDemo3() {
        setLayout(new BorderLayout());
        firstName = new JTextField();
        surName = new JTextField();
        loginId = new JTextField();
        posTxt = new JTextField();
        depTxt = new JTextField();
        writeBtn = new JButton("Write to database");
        displayBtn = new JButton("Display database");
```

The screenshot to the left shows the program “DBDemo3.java”.

As you can see multiple JTextFields have been made along with two JButtons with the captions “Write to Database” and “Display Database”.



```java
        JPanel middle = new JPanel();
        //setting the values of the grid layout (the first number represents
        //the number of items on the GUI and the second represents the number
        //of columns.
        middle.setLayout(new GridLayout(10, 2, 5, 5));
        middle.add(new JLabel("First name:"));
        middle.add(firstName);
        middle.add(new JLabel("Surname:"));
        middle.add(surName);
        middle.add(new JLabel("Login ID:"));
        middle.add(loginId);
        middle.add(new JLabel("Position:"));
        middle.add(posTxt);
        middle.add(new JLabel("Department:"));
        middle.add(depTxt);
        add("Center", middle);
        JPanel bottom = new JPanel();
        bottom.add(writeBtn);
        bottom.add(displayBtn);
        add("South", bottom);
        add("West", new JPanel());
        add("East", new JPanel());
        writeBtn.addActionListener(this);
        displayBtn.addActionListener(this);
        setSize(300, 400);
        setTitle("Database demo 3");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
        setResizable(false);
    }
```

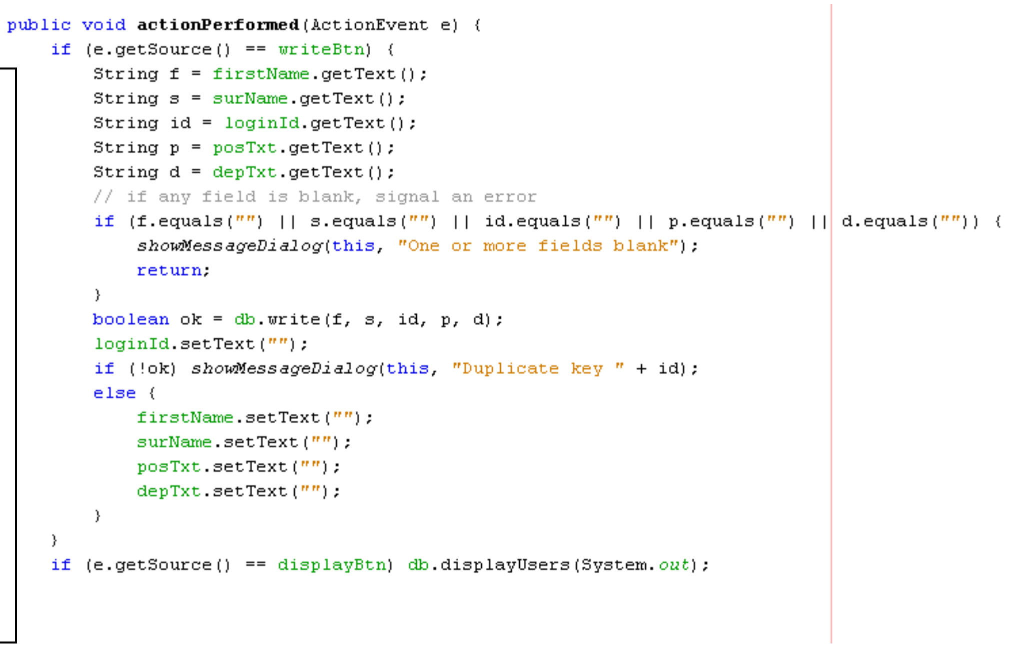
The JTextFields and JButtons are added to the GUI (which is set as a Grid Layout).

Other GUI features are also coded here such as setSize, setTitle, setVisible, setResizable etc.

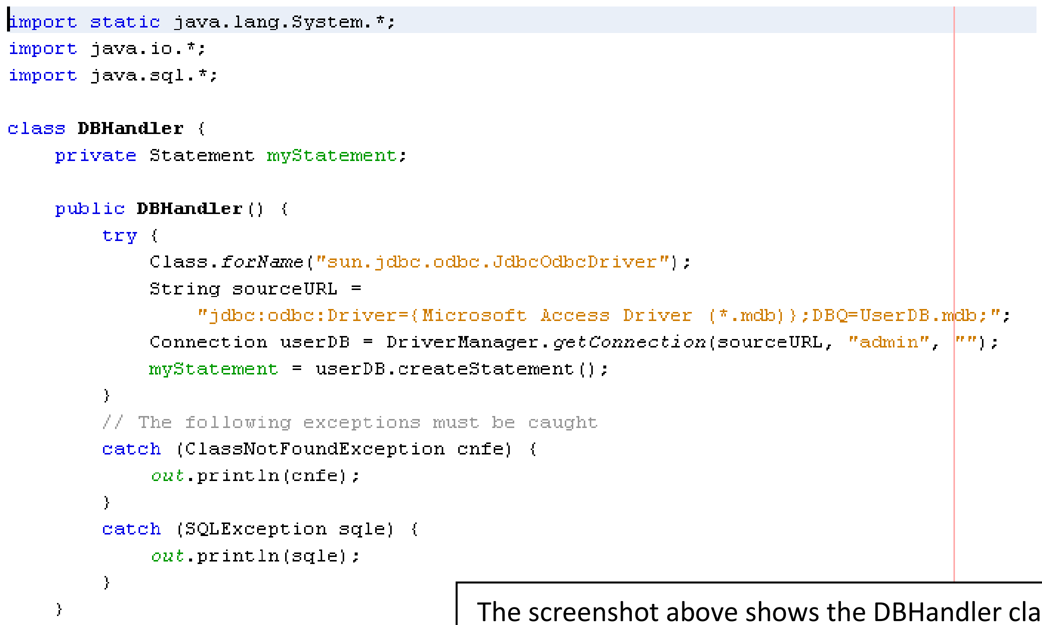
The screenshot to the right shows the code for the actionPerformed section.

When the Write Button is clicked the text from each of the JTextFields is taken by using getText() and written to the database.

When the Display Button is clicked, the information from the database is displayed.



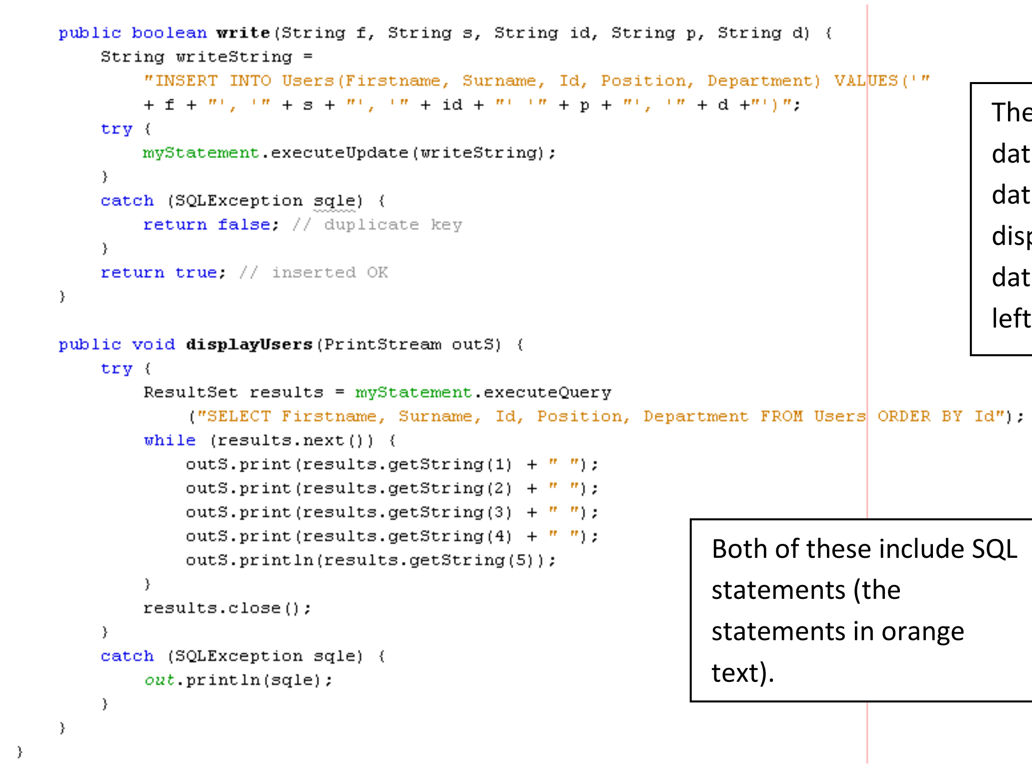
```
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == writeBtn) {
        String f = firstName.getText();
        String s = surName.getText();
        String id = loginId.getText();
        String p = posTxt.getText();
        String d = depTxt.getText();
        // if any field is blank, signal an error
        if (f.equals("") || s.equals("") || id.equals("") || p.equals("") || d.equals("")) {
            showMessageDialog(this, "One or more fields blank");
            return;
        }
        boolean ok = db.write(f, s, id, p, d);
        loginId.setText("");
        if (!ok) showMessageDialog(this, "Duplicate key " + id);
        else {
            firstName.setText("");
            surName.setText("");
            posTxt.setText("");
            depTxt.setText("");
        }
    }
    if (e.getSource() == displayBtn) db.displayUsers(System.out);
```



```
import static java.lang.System.*;
import java.io.*;
import java.sql.*;

class DBHandler {
    private Statement myStatement;

    public DBHandler() {
        try {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            String sourceURL =
                "jdbc:odbc:Driver={Microsoft Access Driver (*.mdb)};DBQ=UserDB.mdb;";
            Connection userDB = DriverManager.getConnection(sourceURL, "admin", "");
            myStatement = userDB.createStatement();
        }
        // The following exceptions must be caught
        catch (ClassNotFoundException cnfe) {
            out.println(cnfe);
        }
        catch (SQLException sqle) {
            out.println(sqle);
        }
    }
```

The screenshot above shows the DBHandler class. This is where the database is connected to the java program using JDBC (Java Database Connectivity) and ODBC (Open Database Connectivity).

There are also two catch methods that bring up error messages for the errors that are thrown by the try statement.



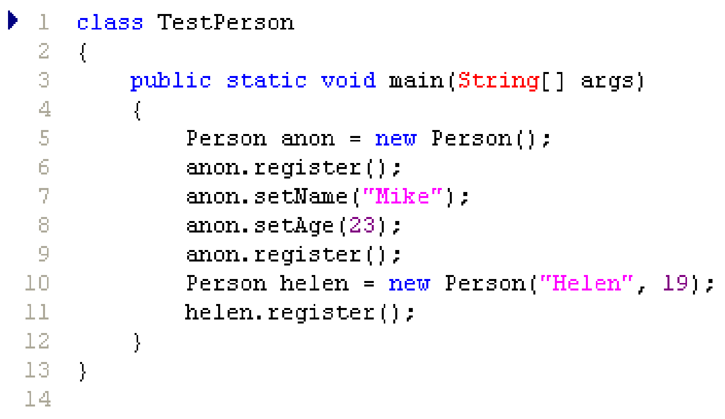
```
    public boolean write(String f, String s, String id, String p, String d) {
        String writeString =
            "INSERT INTO Users(Firstname, Surname, Id, Position, Department) VALUES('"
            + f + "', '" + s + "', '" + id + "' '" + p + "', '" + d +"')";
        try {
            myStatement.executeUpdate(writeString);
        }
        catch (SQLException sqle) {
            return false; // duplicate key
        }
        return true; // inserted OK
    }

    public void displayUsers(PrintStream outS) {
        try {
            ResultSet results = myStatement.executeQuery
                ("SELECT Firstname, Surname, Id, Position, Department FROM Users ORDER BY Id");
            while (results.next()) {
                outS.print(results.getString(1) + " ");
                outS.print(results.getString(2) + " ");
                outS.print(results.getString(3) + " ");
                outS.print(results.getString(4) + " ");
                outS.println(results.getString(5));
            }
            results.close();
        }
        catch (SQLException sqle) {
            out.println(sqle);
        }
    }
}
```

The code for how the database writes to the database and how it displays from the database is shown to the left.

Both of these include SQL statements (the statements in orange text).

16 – *Inheritance*

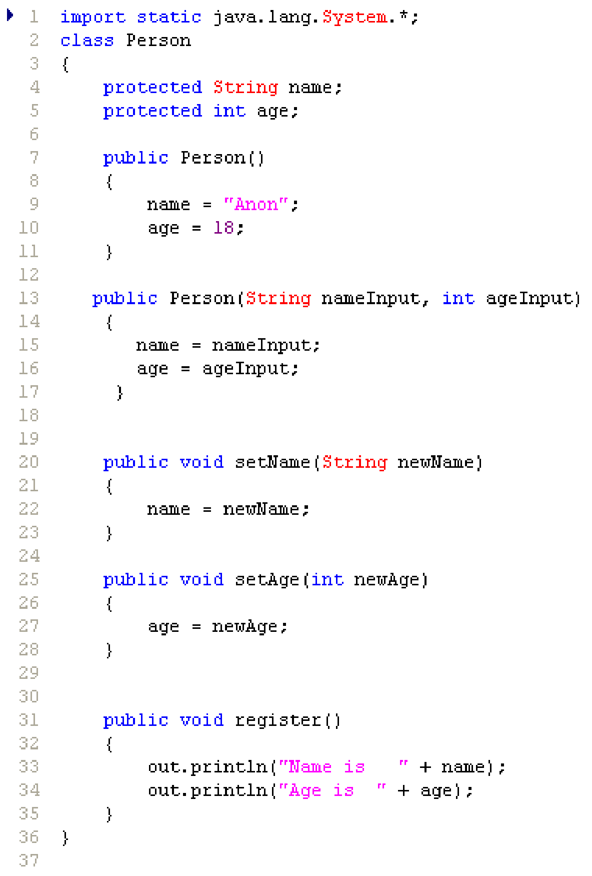
16.1 Inheritance (Person.java and TestPerson.java)



```
class TestPerson
{
    public static void main(String[] args)
    {
        Person anon = new Person();
        anon.register();
        anon.setName("Mike");
        anon.setAge(23);
        anon.register();
        Person helen = new Person("Helen", 19);
        helen.register();
    }
}
```

The screenshot to the left shows the program "TestPerson.java".

As you can see this creates new objects by referencing the methods from the "Person.java" program such as; register(), setName() and setAge().



```
import static java.lang.System.*;
class Person
{
    protected String name;
    protected int age;

    public Person()
    {
        name = "Anon";
        age = 18;
    }

   public Person(String nameInput, int ageInput)
    {
       name = nameInput;
       age = ageInput;
     }


    public void setName(String newName)
    {
        name = newName;
    }

    public void setAge(int newAge)
    {
        age = newAge;
    }


    public void register()
    {
        out.println("Name is   " + name);
        out.println("Age is  " + age);
    }
}
```

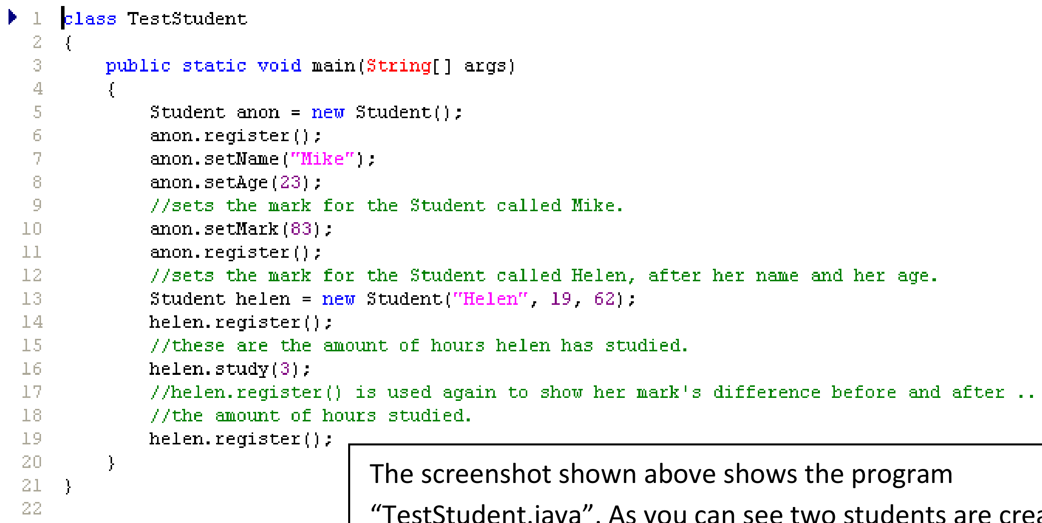
The screenshot to the left shows the program called "Person.java". As you can see this is where all of the methods used in "TestPerson.java" are located.

The string variable name and the integer variable age are both protected, protected means that the variable is visible to code in any subclass of its defining class.

An example of this would be the code is visible to the Student class since it's a subclass of the defining class Person.

As you can see the method register is used to print the names out (most likely at the end) when it is called.

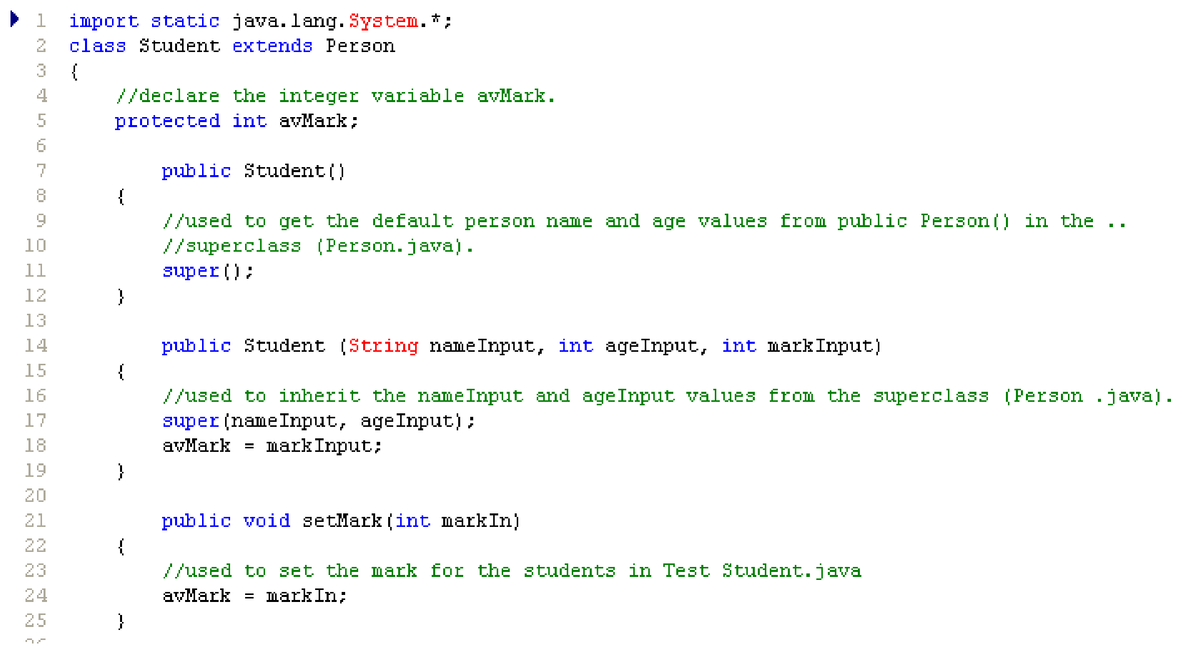
16.2 Inheritance from the Person class ("Student.java and TestStudent.java")



```
class TestStudent
{
    public static void main(String[] args)
    {
        Student anon = new Student();
        anon.register();
        anon.setName("Mike");
        anon.setAge(23);
        //sets the mark for the Student called Mike.
        anon.setMark(83);
        anon.register();
        //sets the mark for the Student called Helen, after her name and her age.
        Student helen = new Student("Helen", 19, 62);
        helen.register();
        //these are the amount of hours helen has studied.
        helen.study(3);
        //helen.register() is used again to show her mark's difference before and after ..
        //the amount of hours studied.
        helen.register();
    }
}
```

The screenshot shown above shows the program "TestStudent.java". As you can see two students are created by the names of Mike and Helen, with their details such as their age and also their mark (grade).

The "helen.study()" is the amount of hours Helen has studied, the register method is called again to show the difference of her mark before and after the amount of hours studied.

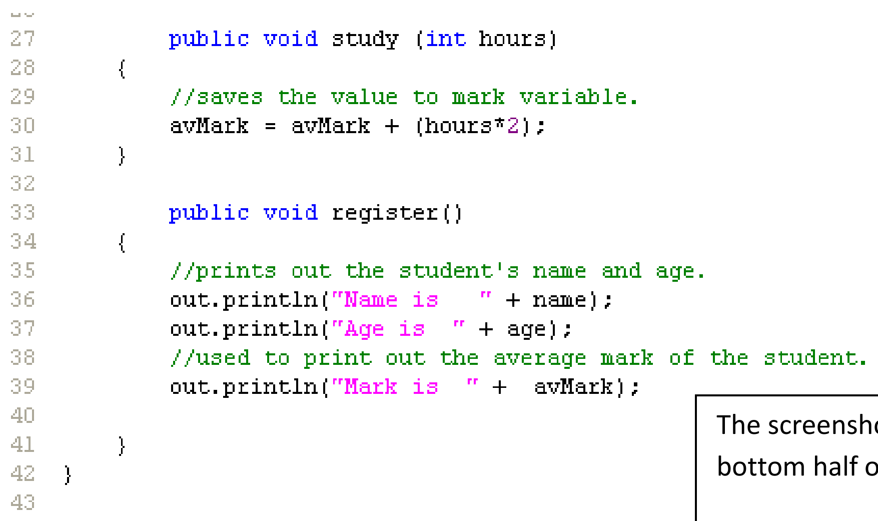


```
 1  import static java.lang.System.*;
 2  class Student extends Person
 3  {
 4      //declare the integer variable avMark.
 5      protected int avMark;
 6
 7          public Student()
 8      {
 9          //used to get the default person name and age values from public Person() in the ..
10          //superclass (Person.java).
11          super();
12      }
13
14          public Student (String nameInput, int ageInput, int markInput)
15      {
16          //used to inherit the nameInput and ageInput values from the superclass (Person .java).
17          super(nameInput, ageInput);
18          avMark = markInput;
19      }
20
21          public void setMark(int markIn)
22      {
23          //used to set the mark for the students in Test Student.java
24          avMark = markIn;
25      }
```

The screenshot above shows the top half of “Student.java”. As you can see, the Student class extends for the Person class as it says “class Student extends Person” on Line 2.

The “super()” on Line 11 is used to get the default person name and age values from the Person method in the superclass “Person.java”. The “super(nameInput, ageInput)” on Line 17 is used to inherit the nameInput and ageInput values from the superclass “Person.java”.

The average mark is also set for the students in “TestStudent.java”.



```
27          public void study (int hours)
28      {
29          //saves the value to mark variable.
30          avMark = avMark + (hours*2);
31      }
32
33          public void register()
34      {
35          //prints out the student's name and age.
36          out.println("Name is   " + name);
37          out.println("Age is  " + age);
38          //used to print out the average mark of the student.
39          out.println("Mark is  " +  avMark);
40
41      }
42  }
43
```

The screenshot to the top left shows the bottom half of “Student.java”.

This includes the Study method which adds the hours studied to the mark variable and the register method which prints out the Name, Age and Mark of the Student.

EX – Extra Stuff

EX.1 Auxiliary Classes (shown with “UpdateLibrary.java” from the Coursework)

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import java.util.*;
public class UpdateLibrary extends JFrame
                  implements ActionListener
{

    LibraryData db = new LibraryData();

    //this is the text field where the Track Number is entered.
    JTextField trackNo = new JTextField(2);
    //this is the text area where certain information in shown, depending on ..
    //what you click on.
    JTextArea libraryUpdate = new JTextArea(16, 50);
    //this is the text field where the Rating is entered.
    JTextField ratingEntry = new JTextField(1);
    //this button is used to make updates to the library.
    JButton update = new JButton("Update Library");
    //this button is used to display the library.
    JButton display = new JButton("Display Library");
    //this button is used to unselect the radio buttons.
    JButton clear = new JButton("Unselect Radio Buttons");
    //the buttons created below are radio buttons, there are three which are used to ..
    //sort, artist and reverse the songs in the library.
    JRadioButton sort = new JRadioButton("Sort");
    JRadioButton song = new JRadioButton("Sort All Songs");
    JRadioButton artist = new JRadioButton("Sort All Artists");
    JRadioButton reverse = new JRadioButton("Reverse");
    //creation of the button group called "ssar", this button group ..
    //contains the sort, artist and reverse radio buttons and only allows ..
    //one of them to be selected at a time.
    ButtonGroup ssar = new ButtonGroup();
    //these are JLabels and they are used to label certain objects on the UpdateLibrary screen.
    JLabel enterTrack  = new JLabel("Enter a Track Number:");
    JLabel enterRating = new JLabel("Enter a New Rating (1-5):");
```

As you can see in order to call methods and to work an Auxiliary class, you must write a line in the class that creates an instance of it in the class you’re going to call from.

The example shown here is a constructor used to create a LibraryData object called db (this is written in the UpdateLibrary class).

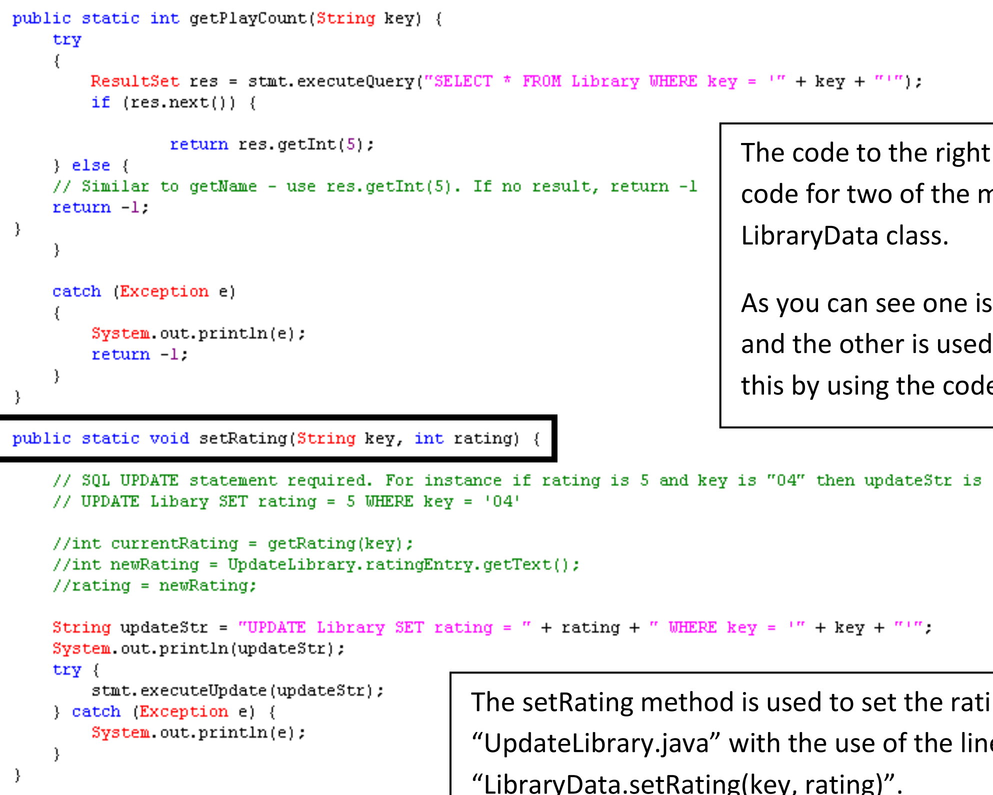
```
        public void actionPerformed(ActionEvent e)
        {


            if (e.getSource() == update)
            {
           try {
                //defines the string variables "key" and "ratingStr", which both get their values from the ..
                //text fields "trackNo" and "ratingEntry".
                String key = trackNo.getText();
                String ratingStr = ratingEntry.getText();
                //a new variable is defined called "rating" which converts the text value of "ratingStr" ..
                //to an integer value.
                int rating = Integer.parseInt(ratingStr);
                //uses the setRating method in the LibraryData class along with the ..
                //variables "key" and "rating" to set the Rating to the newly entered one ..
                //in the database.
                LibraryData.setRating(key, rating);
                //displays an output to show the changes made to the rating of a certain track have ..
                //taken place.
                libraryUpdate.setText("Updates made to the rating of the Track below are;");
                libraryUpdate.append("\n\n" + "Song Name: " + LibraryData.getName(key));
                libraryUpdate.append("\n" + "New Rating: " + stars(LibraryData.getRating(key)));
```

The String variable key is declared here, while the int variable rating is declared as a String first then converted to an integer.

Calling a method from an Auxiliary class is shown above, it should be written as “classname.name of method(values included)”, the values included should be separated with commas.

An example of this would be “LibraryData.setRating(key, rating);” (as shown in the image above) with “LibraryData” as the name of the class, “setRating” as the name of the method with “key” and “rating” as the values included in the method.

EX.2 Auxiliary Classes (shown with “LibraryData.java” from the Coursework)



```
public static int getPlayCount(String key) {
    try
    {
        ResultSet res = stmt.executeQuery("SELECT * FROM Library WHERE key = '" + key + "'");
        if (res.next()) {

                return res.getInt(5);
    } else {
    // Similar to getName - use res.getInt(5). If no result, return -1
    return -1;
}
    }

    catch (Exception e)
    {
        System.out.println(e);
        return -1;
    }
}
public static void setRating(String key, int rating) {

    // SQL UPDATE statement required. For instance if rating is 5 and key is "04" then updateStr is
    // UPDATE Libary SET rating = 5 WHERE key = '04'

    //int currentRating = getRating(key);
    //int newRating = UpdateLibrary.ratingEntry.getText();
    //rating = newRating;

    String updateStr = "UPDATE Library SET rating = " + rating + " WHERE key = '" + key + "'";
    System.out.println(updateStr);
    try {
        stmt.executeUpdate(updateStr);
    } catch (Exception e) {
        System.out.println(e);
    }
}
```

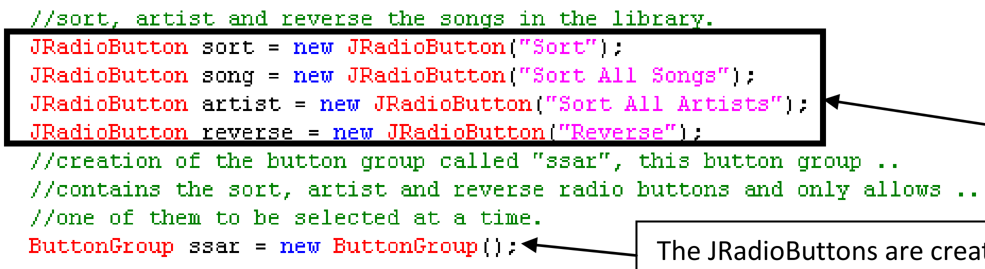
The code to the right in the screenshot is the code for two of the methods written in the LibraryData class.

As you can see one is used to get the play count and the other is used to set the rating, both do this by using the code inside their own method.

The setRating method is used to set the rating in the class “UpdateLibrary.java” with the use of the line “LibraryData.setRating(key, rating)”.

As you can see in the LibraryData class where all the methods are kept, the variable is written with the line “public static void setRating(String key, int rating)“ this shows that when we want to call this method a string variable called key and an integer variable called rating must be used as the values included.

EX.3 Radio Buttons and Button Groups (shown with “UpdateLibrary.java” from the Coursework)

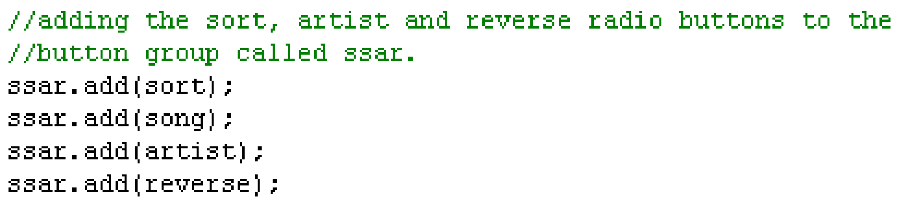


```
//sort, artist and reverse the songs in the library.
JRadioButton sort = new JRadioButton("Sort");
JRadioButton song = new JRadioButton("Sort All Songs");
JRadioButton artist = new JRadioButton("Sort All Artists");
JRadioButton reverse = new JRadioButton("Reverse");
//creation of the button group called "ssar", this button group ..
//contains the sort, artist and reverse radio buttons and only allows ..
//one of them to be selected at a time.
ButtonGroup ssar = new ButtonGroup();
```

The JRadioButtons are created here using this format; the ButtonGroup is also created here using this format.

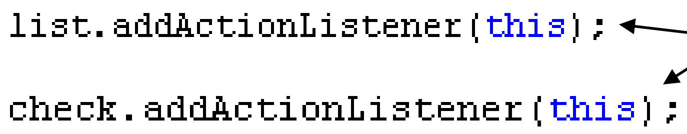
The JRadioButton text in brackets is what the RadioButton will say next to it when the program is run.

The screenshot to the right shows how the buttons are added to the button group.



```
//adding the sort, artist and reverse radio buttons to the
//button group called ssar.
ssar.add(sort);
ssar.add(song);
ssar.add(artist);
ssar.add(reverse);
```

EX.4 Adding Action Listeners to Buttons (using CheckLibrary.java from the Coursework)



```
list.addActionListener(this);

check.addActionListener(this);
```

This is how Action Listeners are added to buttons in Java.

The screenshot to the left shows the buttons list and check having Action Listeners added to them.

Remember to add the word “this” inside the brackets.