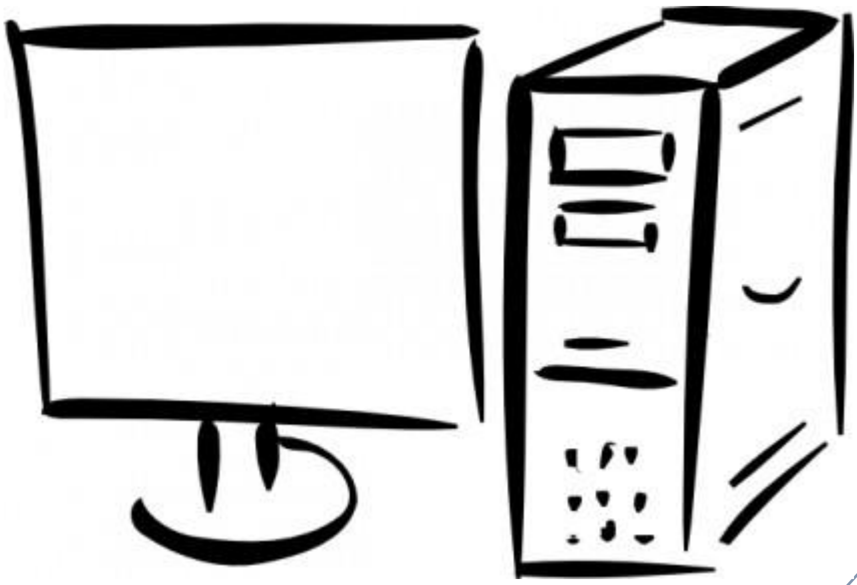
# Java & Distributed Systems

ITJA321 - 2018



**Matthew Van der Bijl**
XQ9X3WV31

## Table of Contents

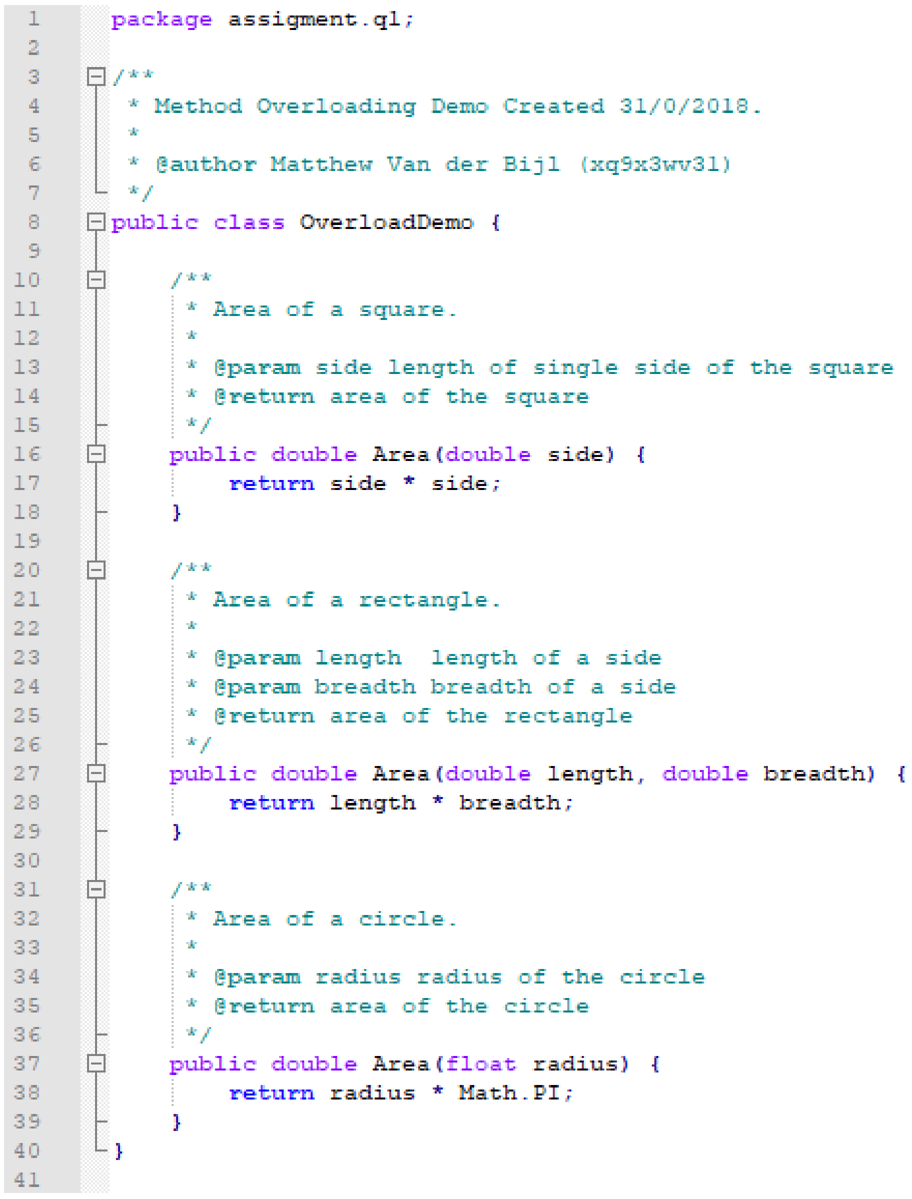
## Table of Figures

## Question 1

### 1.1

The code that follows makes use of method overloading. The java code below was constructed using modern Object-Orientated Programming (OOP) principles as presented by Gaddis (2016), Pretorius and Erasmus (2012), Deitel and Deitel (2012), Weisfeld (2013) and Drake (2014).

Please note that all methods below have a unique parameter signature.

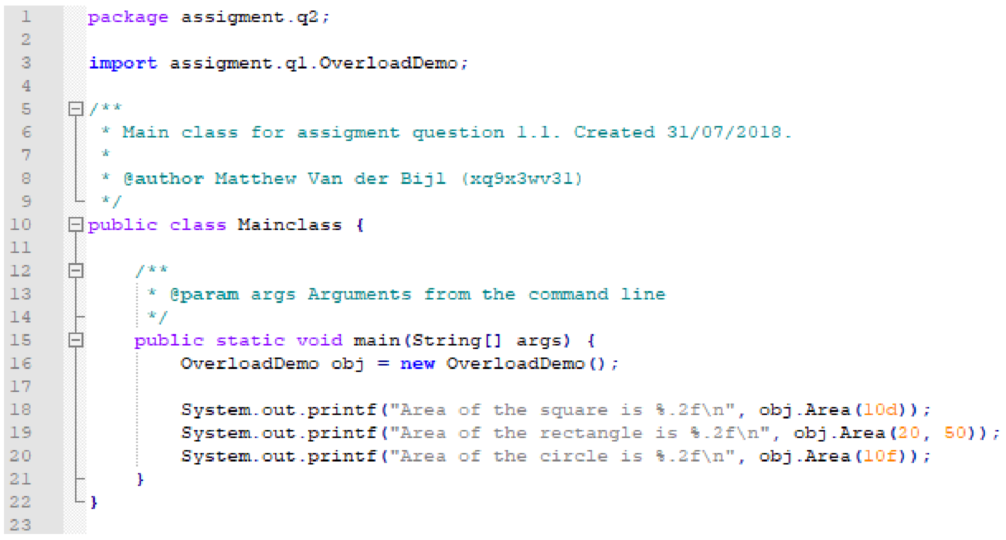


```java
package assigment.q1;

/**
 * Method Overloading Demo Created 31/0/2018.
 *
 * @author Matthew Van der Bijl (xq9x3wv31)
 */
public class OverloadDemo {

    /**
     * Area of a square.
     *
     * @param side length of single side of the square
     * @return area of the square
     */
    public double Area(double side) {
        return side * side;
    }

    /**
     * Area of a rectangle.
     *
     * @param length  length of a side
     * @param breadth breadth of a side
     * @return area of the rectangle
     */
    public double Area(double length, double breadth) {
        return length * breadth;
    }

    /**
     * Area of a circle.
     *
     * @param radius radius of the circle
     * @return area of the circle
     */
    public double Area(float radius) {
        return radius * Math.PI;
    }
}

```

*Figure 1 Overload Demo class*

As seen above, each method has been successfully created.

The class below contains the main method of the programme and is used to test the programme.



```java
package assigment.q2;

import assigment.q1.OverloadDemo;

/**
 * Main class for assigment question 1.1. Created 31/07/2018.
 *
 * @author Matthew Van der Bijl (xq9x3wv31)
 */
public class Mainclass {

    /**
     * @param args Arguments from the command line
     */
    public static void main(String[] args) {
        OverloadDemo obj = new OverloadDemo();

        System.out.printf("Area of the square is %.2f\n", obj.Area(10d));
        System.out.printf("Area of the rectangle is %.2f\n", obj.Area(20, 50));
        System.out.printf("Area of the circle is %.2f\n", obj.Area(10f));
    }
}
```

*Figure 2 Main class used to test area formulas*

As seen above, each method is individually called, and the result is printed to the command line.

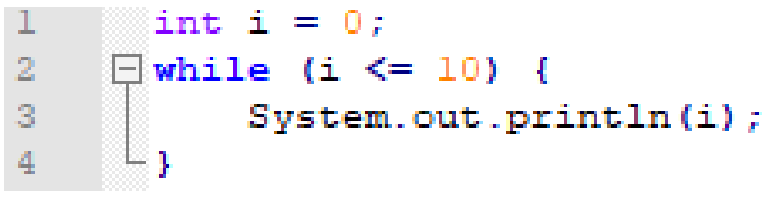
## 1.2

### a)

In programming, a loop a select set of statements than indefinitely until a certain condition is met (Liang, 2011). There are two main types of loops, namely while loops and for loops (Liang, 2011). Reges and Stepp (2014) note that looping may also be referred to as code repetition.

With regards to programming, a while loop is a form of repetition structure (Reges & Stepp, 2014). There are two primary types of while loops, while loops and do-while loops. According to Bailey (2005), there are while loop and do-while loops. A Do-While loop is a form of post-test loop whereas while loops are pre-test loops (Bailey, 2005).

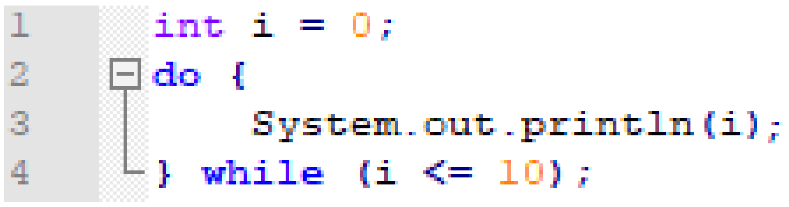
The code that follows is a basic while loop implemented in java. The code was constructed using a presentation format presented by Liang (2011).



```java
int i = 0;
while (i <= 10) {
     System.out.println(i);
}
```

*Figure 3 Basic java while loop*

As seen above, the condition of the loop is tested before the statements are run. This is why while loops are known as pretesting loops. The code that follows is a basic do-while loop implemented in java. The code was constructed using a presentation format presented by Liang (2011).



```java
int i = 0;
do {
     System.out.println(i);
} while (i <= 10);
```

*Figure 4 Basic do-while loop*

As seen above, the condition of the loop is tested after the statements are run at least once. This is why while loops are known as pretesting loops. Though each loop operates in a similar manner, each loop serves its own purpose and should be used when appropriate.

Loops, like many other programming structures, can be represented using flow charts. The flow chart below was created following a format presented by Gaddis (2016).

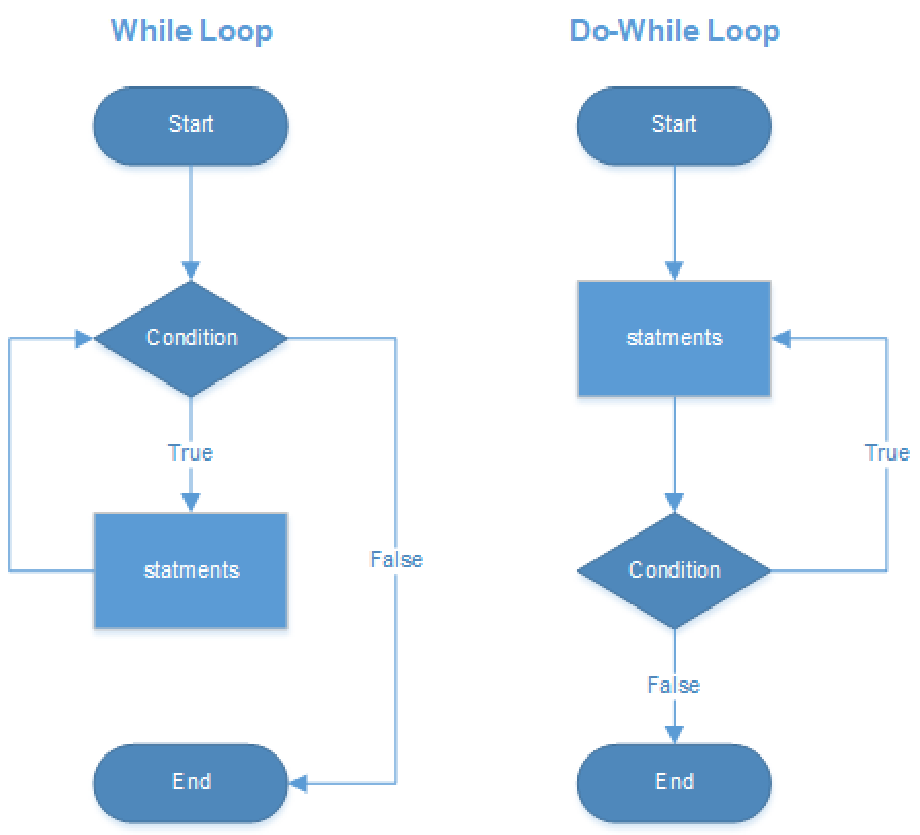
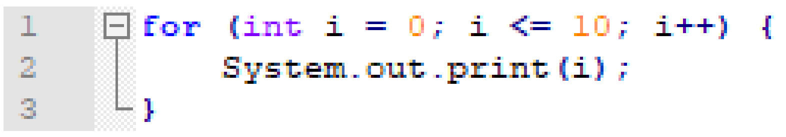


*Figure 5 While loop vs Do-While loop*

As clearly shown above, with a while loop the condition is run before the statements are run compared to a do while loop where the statements run before the condition is tested. For this reason, a do while loop will run the statements at least once regardless of the condition.

The code snippets below show a basic for loop respectively in java as suggested by Gaddis (2016).



```
for (int i = 0; i <= 10; i++) {
    System.out.print(i);
}
```

*Figure 6 Basic java for loop*

b)

Multi-threading allows for multiple statements to run simultaneously (Liang, 2011). In Java, Runnables and threads form the backbone on multithreading. In Java, programmers extend threads and implement runnables.

Task are objects that implement the Runnable interface (Liang, 2011). In Java the runnable interfaces as a functional interface, an interface that only contains a single method, that contains the run method (Liang, 2011). Interfaces require a special type of inheritance for all non-abstract classes that implement them (Weisfeld, 2013). Once implemented, the run method is used to determine what actions are performed on the thread (Liang, 2011).

According to Liang (2011), tasks need to be executed on a thread. In Java, a thread is an object that is used to execute runnable (Liang, 2011). Threads are executed by calling the start method. As the Thread class implements the Runnable interfaces all threads can be seen as tasks (Liang, 2011).

The custom thread can be created. The class diagram that follows illustrates the relationship that exists between the Runnable interface, the Thread class and custom threads and custom tasks. The class diagram was constructed following a format suggested by Bennet, et al. (2010) and Valacich, et al. (2015).

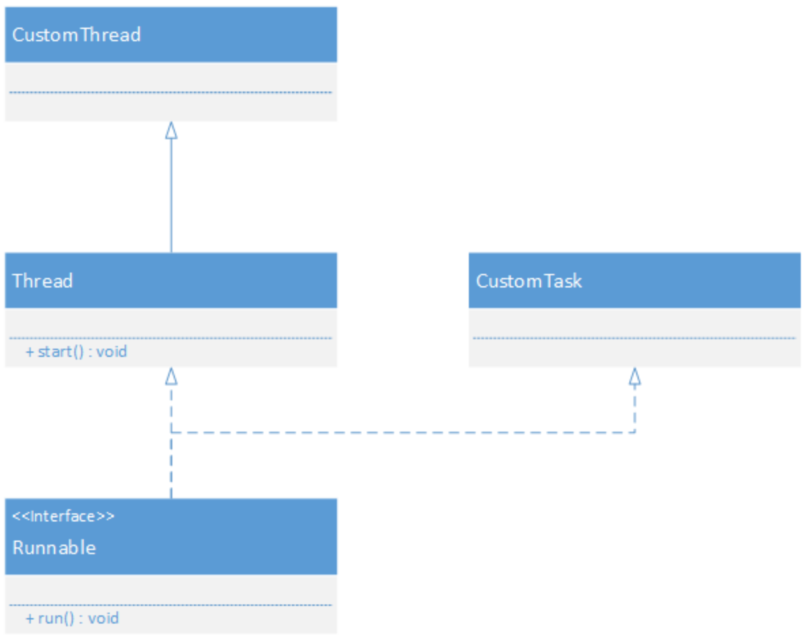


*Figure 7 Task & thread class diagram*

As seen above all threads implement the Runnable interface and all tasks and custom threads need to implement the run method.

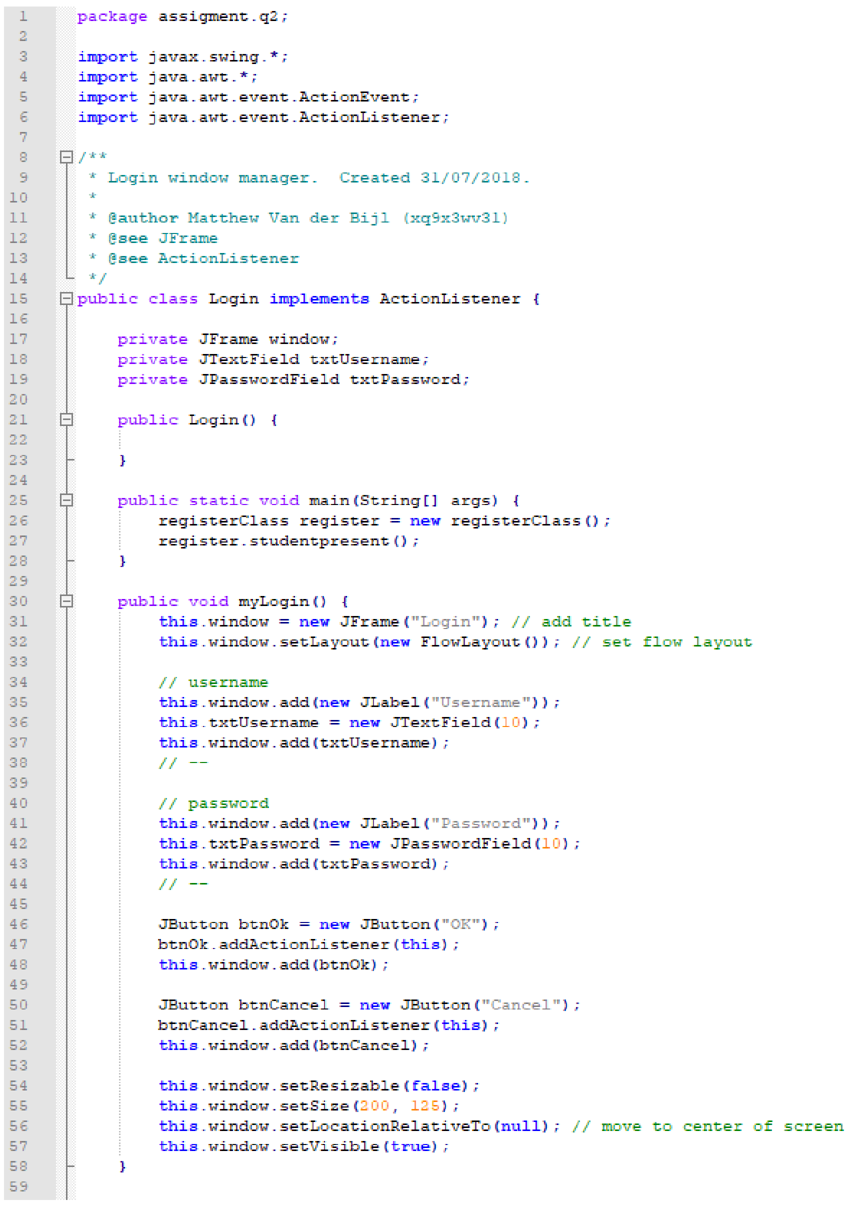
In Java, classes extend other class and implement interfaces. In order to create a custom task, programmers need to implement the Runnable interfaces. To create a custom thread, programmers need to extend the Thread class.

## Question 2

The code that follows is a potential solution to the scenario presented. The database was developed based on information presented by Connolly and Begg (2015), Groff and Weinberg (2002) and Beyon-Davies (2008). The java code below was constructed using modern Object-Orientated Programming (OOP) principles as presented by Gaddis (2016), Pretorius and Erasmus (2012), Deitel and Deitel (2012), Weisfeld (2013) and Drake (2014).

### 2.1

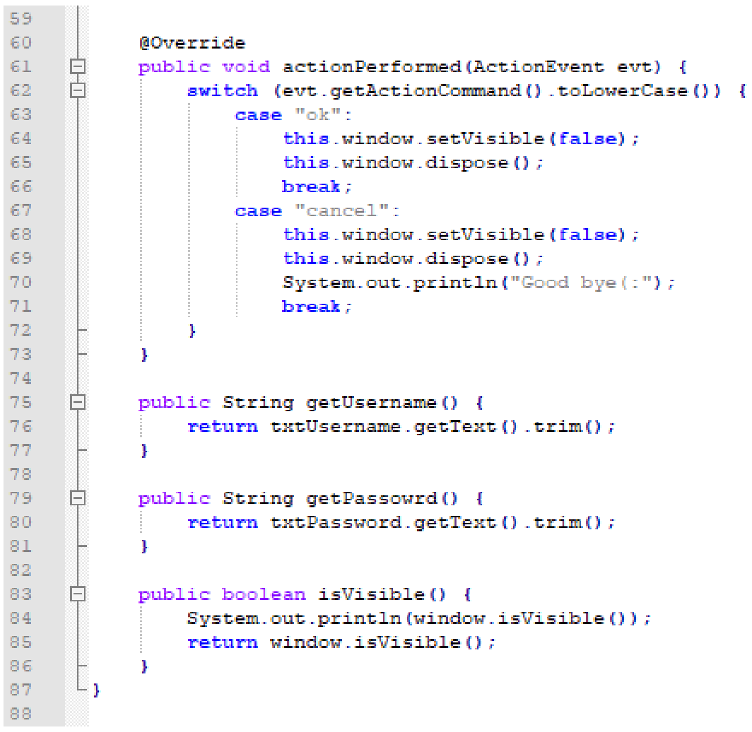
The code below creates a Graphical User Interface (GUI). It is important to evaluate the usability of user interfaces (UI) (Preece, et al., 2015).



```java
package assigment.q2;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

/**
 * Login window manager.  Created 31/07/2018.
 *
 * @author Matthew Van der Bijl (xq9x3wv31)
 * @see JFrame
 * @see ActionListener
 */
public class Login implements ActionListener {

    private JFrame window;
    private JTextField txtUsername;
    private JPasswordField txtPassword;

    public Login() {

    }

    public static void main(String[] args) {
        registerClass register = new registerClass();
        register.studentpresent();
    }

    public void myLogin() {
        this.window = new JFrame("Login"); // add title
        this.window.setLayout(new FlowLayout()); // set flow layout

        // username
        this.window.add(new JLabel("Username"));
        this.txtUsername = new JTextField(10);
        this.window.add(txtUsername);
        // --

        // password
        this.window.add(new JLabel("Password"));
        this.txtPassword = new JPasswordField(10);
        this.window.add(txtPassword);
        // --

        JButton btnOk = new JButton("OK");
        btnOk.addActionListener(this);
        this.window.add(btnOk);

        JButton btnCancel = new JButton("Cancel");
        btnCancel.addActionListener(this);
        this.window.add(btnCancel);

        this.window.setResizable(false);
        this.window.setSize(200, 125);
        this.window.setLocationRelativeTo(null); // move to center of screen
        this.window.setVisible(true);
    }

```

*Figure 8 Login class part 1*

The code snippet below handle’s user input contains two getters and a boolean value testing whether the window is closed or not.



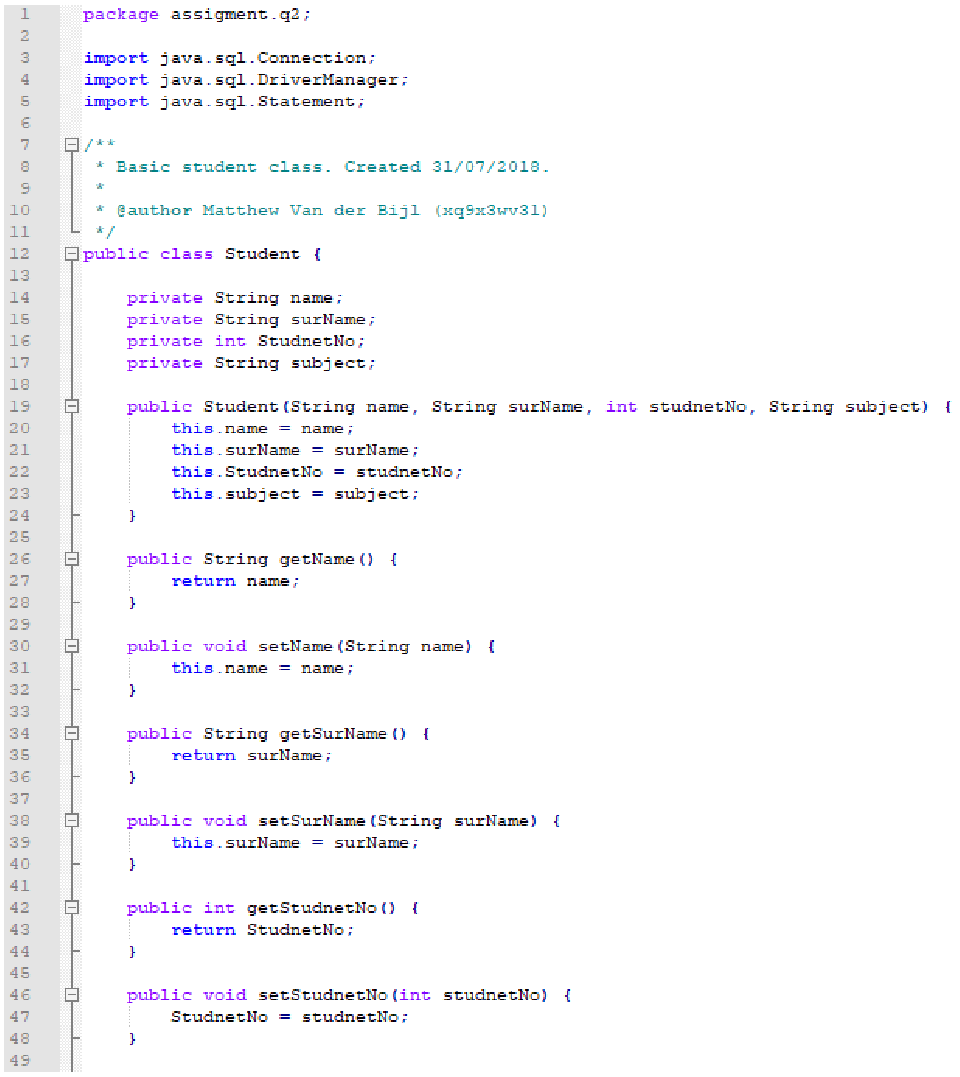
```java

    @Override
    public void actionPerformed(ActionEvent evt) {
        switch (evt.getActionCommand().toLowerCase()) {
            case "ok":
                this.window.setVisible(false);
                this.window.dispose();
                break;
            case "cancel":
                this.window.setVisible(false);
                this.window.dispose();
                System.out.println("Good bye(:");
                break;
        }
    }

    public String getUsername() {
        return txtUsername.getText().trim();
    }

    public String getPassowrd() {
        return txtPassword.getText().trim();
    }

    public boolean isVisible() {
        System.out.println(window.isVisible());
        return window.isVisible();
    }
}

```

*Figure 9 Login class part 2*

As seen above, there are no errors in the code.

## 2.2

The snippets below are of the student class. The student class file will be submitted along site this document.

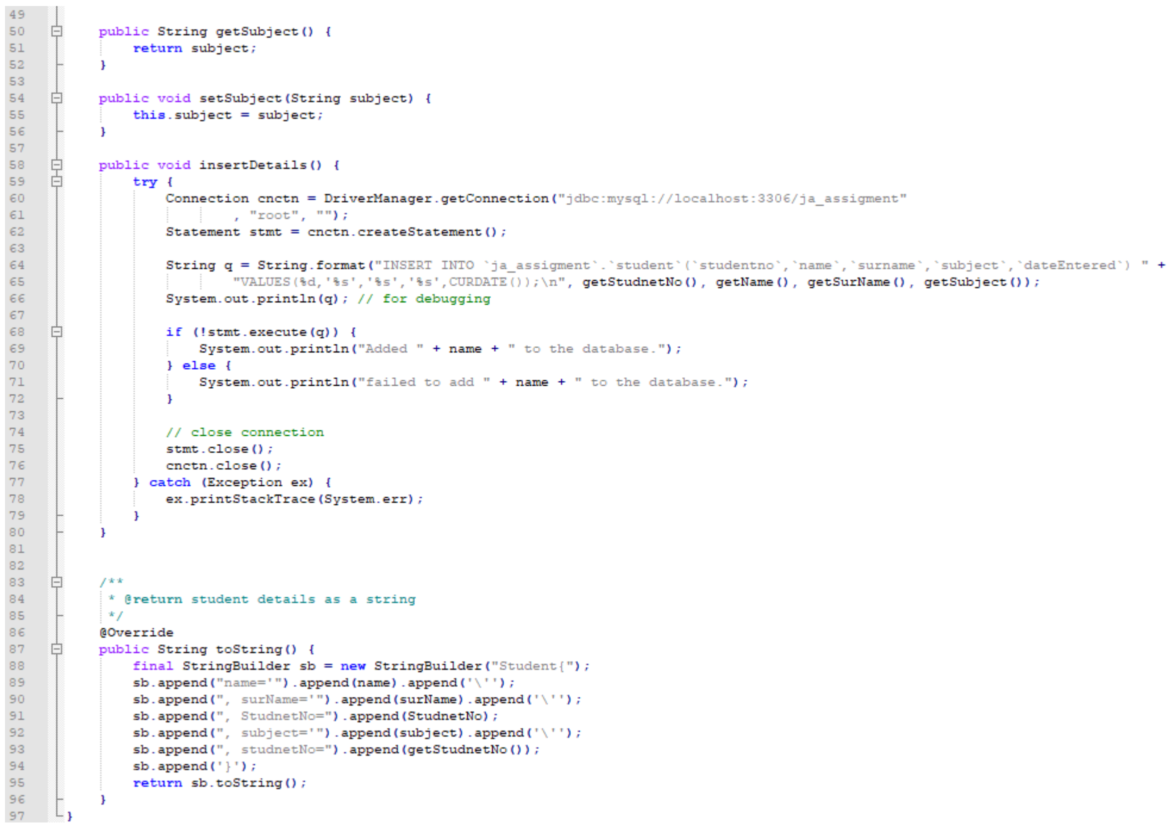


```java
package assigment.q2;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;

/**
 * Basic student class. Created 31/07/2018.
 *
 * @author Matthew Van der Bijl (xq9x3wv31)
 */
public class Student {

    private String name;
    private String surName;
    private int StudnetNo;
    private String subject;

    public Student(String name, String surName, int studnetNo, String subject) {
        this.name = name;
        this.surName = surName;
        this.StudnetNo = studnetNo;
        this.subject = subject;
    }

    public String getName() {
        return name;
    }

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

    public String getSurName() {
        return surName;
    }

    public void setSurName(String surName) {
        this.surName = surName;
    }

    public int getStudnetNo() {
        return StudnetNo;
    }

    public void setStudnetNo(int studnetNo) {
        StudnetNo = studnetNo;
    }

```

*Figure 10 Student java class part 1*

The code about contains variable declarations, a constructor and getters and setters.

The code below is the second part of the student class.



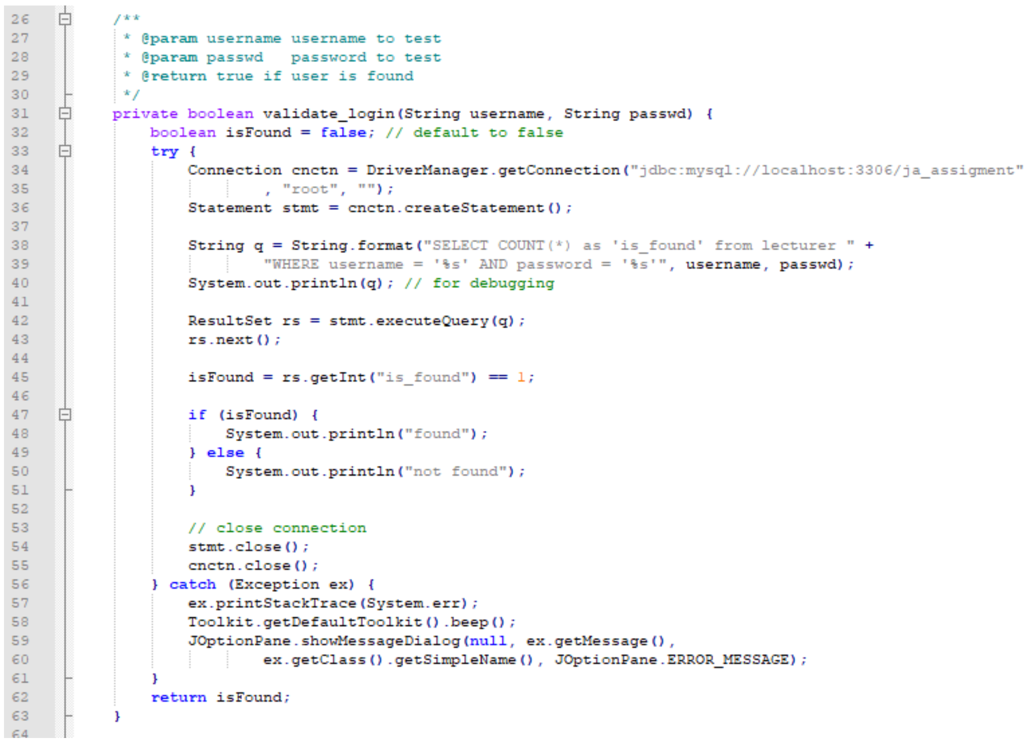
```java

    public String getSubject() {
        return subject;
    }

    public void setSubject(String subject) {
        this.subject = subject;
    }

    public void insertDetails() {
        try {
            Connection cnctn = DriverManager.getConnection("jdbc:mysql://localhost:3306/ja_assigment"
                    , "root", "");
            Statement stmt = cnctn.createStatement();

            String q = String.format("INSERT INTO `ja_assigment`.`student`(`studentno`,`name`,`surname`,`subject`,`dateEntered`) " +
                    "VALUES(%d,'%s','%s','%s',CURDATE());\n", getStudnetNo(), getName(), getSurName(), getSubject());
            System.out.println(q); // for debugging

            if (!stmt.execute(q)) {
                System.out.println("Added " + name + " to the database.");
            } else {
                System.out.println("failed to add " + name + " to the database.");
            }

            // close connection
            stmt.close();
            cnctn.close();
        } catch (Exception ex) {
            ex.printStackTrace(System.err);
        }
    }


    /**
     * @return student details as a string
     */
    @Override
    public String toString() {
        final StringBuilder sb = new StringBuilder("Student{");
        sb.append("name='").append(name).append('\'');
        sb.append(", surName='").append(surName).append('\'');
        sb.append(", StudnetNo=").append(StudnetNo);
        sb.append(", subject='").append(subject).append('\'');
        sb.append(", studnetNo=").append(getStudnetNo());
        sb.append('}');
        return sb.toString();
    }
}
```

*Figure 11 Student java class part 2*

The code above contains the to string method and the insert details method.

## 2.3

The method below is used to validate a user's details in order to log in to the system. The method two parameters and returns a boolean result. The method makes a connection to the database in a manner suggested by Bawankule and Raut (2014).

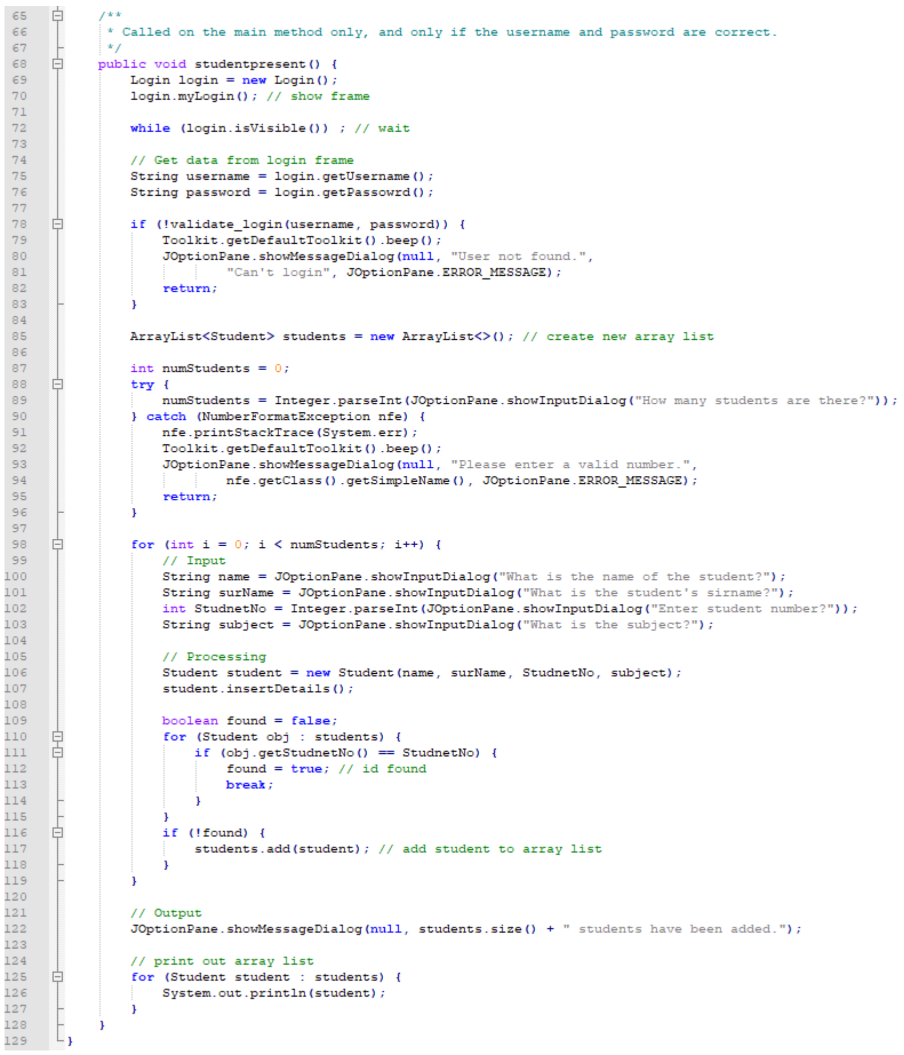


```java
/**
 * @param username username to test
 * @param passwd   password to test
 * @return true if user is found
 */
private boolean validate_login(String username, String passwd) {
    boolean isFound = false; // default to false
    try {
        Connection cnctn = DriverManager.getConnection("jdbc:mysql://localhost:3306/ja_assigment"
                , "root", "");
        Statement stmt = cnctn.createStatement();

        String q = String.format("SELECT COUNT(*) as 'is_found' from lecturer " +
                "WHERE username = '%s' AND password = '%s'", username, passwd);
        System.out.println(q); // for debugging

        ResultSet rs = stmt.executeQuery(q);
        rs.next();

        isFound = rs.getInt("is_found") == 1;

        if (isFound) {
            System.out.println("found");
        } else {
            System.out.println("not found");
        }

        // close connection
        stmt.close();
        cnctn.close();
    } catch (Exception ex) {
        ex.printStackTrace(System.err);
        Toolkit.getDefaultToolkit().beep();
        JOptionPane.showMessageDialog(null, ex.getMessage(),
                ex.getClass().getSimpleName(), JOptionPane.ERROR_MESSAGE);
    }
    return isFound;
}
```

*Figure 12 Method to validate login*

## 2.4

The method below is used to take a roll call. It creates and displays a new login frame.

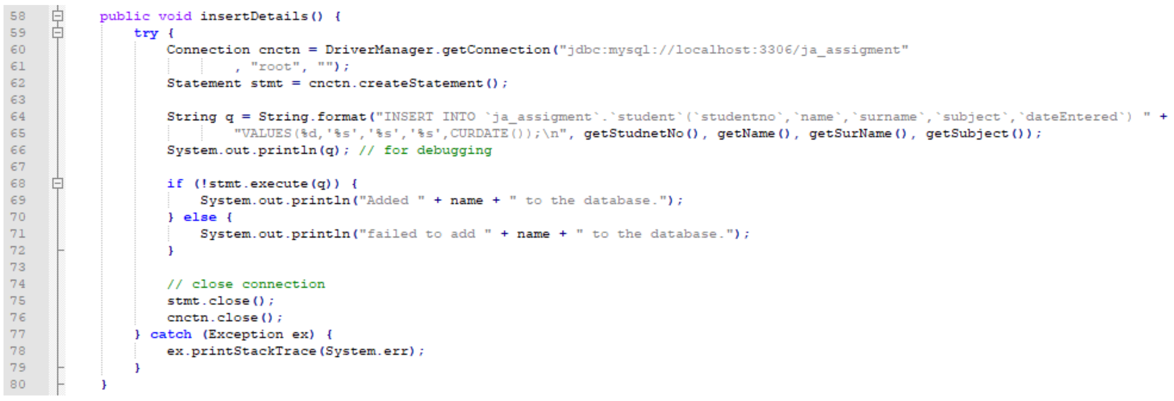


```java
    /**
     * Called on the main method only, and only if the username and password are correct.
     */
    public void studentpresent() {
        Login login = new Login();
        login.myLogin(); // show frame

        while (login.isVisible()) ; // wait

        // Get data from login frame
        String username = login.getUsername();
        String password = login.getPassowrd();

        if (!validate_login(username, password)) {
            Toolkit.getDefaultToolkit().beep();
            JOptionPane.showMessageDialog(null, "User not found.",
                    "Can't login", JOptionPane.ERROR_MESSAGE);
            return;
        }

        ArrayList<Student> students = new ArrayList<>(); // create new array list

        int numStudents = 0;
        try {
            numStudents = Integer.parseInt(JOptionPane.showInputDialog("How many students are there?"));
        } catch (NumberFormatException nfe) {
            nfe.printStackTrace(System.err);
            Toolkit.getDefaultToolkit().beep();
            JOptionPane.showMessageDialog(null, "Please enter a valid number.",
                    nfe.getClass().getSimpleName(), JOptionPane.ERROR_MESSAGE);
            return;
        }

        for (int i = 0; i < numStudents; i++) {
            // Input
            String name = JOptionPane.showInputDialog("What is the name of the student?");
            String surName = JOptionPane.showInputDialog("What is the student's sirname?");
            int StudnetNo = Integer.parseInt(JOptionPane.showInputDialog("Enter student number?"));
            String subject = JOptionPane.showInputDialog("What is the subject?");

            // Processing
            Student student = new Student(name, surName, StudnetNo, subject);
            student.insertDetails();

            boolean found = false;
            for (Student obj : students) {
                if (obj.getStudnetNo() == StudnetNo) {
                    found = true; // id found
                    break;
                }
            }
            if (!found) {
                students.add(student); // add student to array list
            }
        }

        // Output
        JOptionPane.showMessageDialog(null, students.size() + " students have been added.");

        // print out array list
        for (Student student : students) {
            System.out.println(student);
        }
    }
}
```

*Figure 13 Student present method*

## 2.5

The code below is used to insert a student into the database. If an error occurs the details of the error are printed out. The method makes use of getters. The method makes a connection to the database in a manner suggested by Bawankule and Raut (2014).

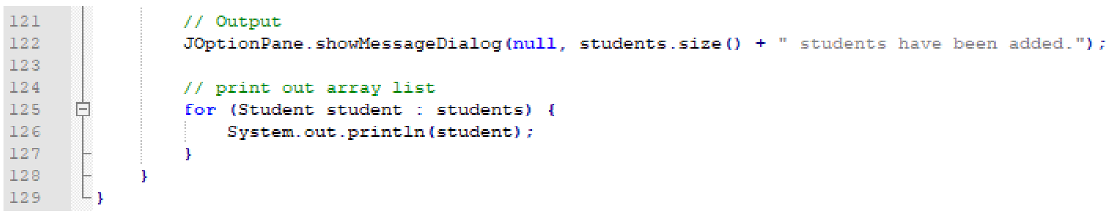


```java
public void insertDetails() {
    try {
        Connection cnctn = DriverManager.getConnection("jdbc:mysql://localhost:3306/ja_assigment"
                , "root", "");
        Statement stmt = cnctn.createStatement();

        String q = String.format("INSERT INTO `ja_assigment`.`student`(`studentno`,`name`,`surname`,`subject`,`dateEntered`) " +
                "VALUES(%d,'%s','%s','%s',CURDATE());\n", getStudnetNo(), getName(), getSurName(), getSubject());
        System.out.println(q); // for debugging

        if (!stmt.execute(q)) {
            System.out.println("Added " + name + " to the database.");
        } else {
            System.out.println("failed to add " + name + " to the database.");
        }

        // close connection
        stmt.close();
        cnctn.close();
    } catch (Exception ex) {
        ex.printStackTrace(System.err);
    }
}
```

Figure 14 Method to insert student details

## 2.6

The code below displays the number of students registers and then prints out all their details to the command line. All data is taken from the array list.



```java
        // Output
        JOptionPane.showMessageDialog(null, students.size() + " students have been added.");

        // print out array list
        for (Student student : students) {
            System.out.println(student);
        }
    }
}
```

Figure 15 Method to display array list

## Bibliography

Bailey, T., 2005. *An Introduction to the C Programming Language and Software Design.* Sydney: The University of Sydney.

Bawankule, K. L. & Raut, N. B., 2014. Design and implementation of massive MYSQL data intelligent export system to excel by using Apache –POI libraries. *Journal of Computer Engineering,* 16(5), pp. 2278-8727.

Bennet, S., McRobb, S. & Farmer, R., 2010. *Object-Oriented Systems Analysis and Design Using UML.* 4th ed. London: McGraw Hill.

Beyond-Davies, P., 2008. *Database Design.* 3rd ed. Basingstoke: Palgrave Macmillan.

Connolly, T. M. & Begg, C. E., 2015. *Database Solutions: A step-by-step approach to building databases.* 6th ed. Harlow: Pearson Education Limited.

Deitel, P. & Deitel, H., 2012. *Java: How to Program.* 9th ed. Boston: Prentice Hall.

Drake, P., 2014. *Data Structures and Algorithms in Java.* 1st ed. Harlow: Pearson Education.

Gaddis, T., 2016. *Programming Logic & Design.* 4th ed. Boston: Pearson.

Groff, J. R. & Weinberg, P. N., 2002. *SQL: The Complete Reference.* 2nd ed. New York City: McGraw-HIll.

Liang, Y. D., 2011. *Introduction to Java Programming.* 8th ed. Boston: Prentice Hall.

Preece, J., Rogers, Y. & Sharp, H., 2015. *Interaction Design: Beyond Human-Computer Interaction.* New Jersey: John Wiley & Sons.

Pretorius, C. M. & Erasmus, G. H., 2012. *Basic Programming Principles.* 2nd ed. Cape Town: Pearson Education South Africa.

Reges, S. & Stepp, M., 2014. *Building Java programs: a back to basics.* 3rd ed. Boston: Addison-Wesley Publishing.

Valacich, J. S., George, J. F. & Hoffer, J. A., 2015. *Essentials of Systems Analysis and Design.* Global Edition ed. New Jersey: Prentice Hall.

Weisfeld, M., 2013. *The Object-Oriented Thought Process.* 4th ed. Boston: Addison-Wesley.