INTPROG TB2: OBJECT ORIENTED PROGRAMMING

Practical Worksheet 4: Creating and Using Classes in Java

1. Introduction

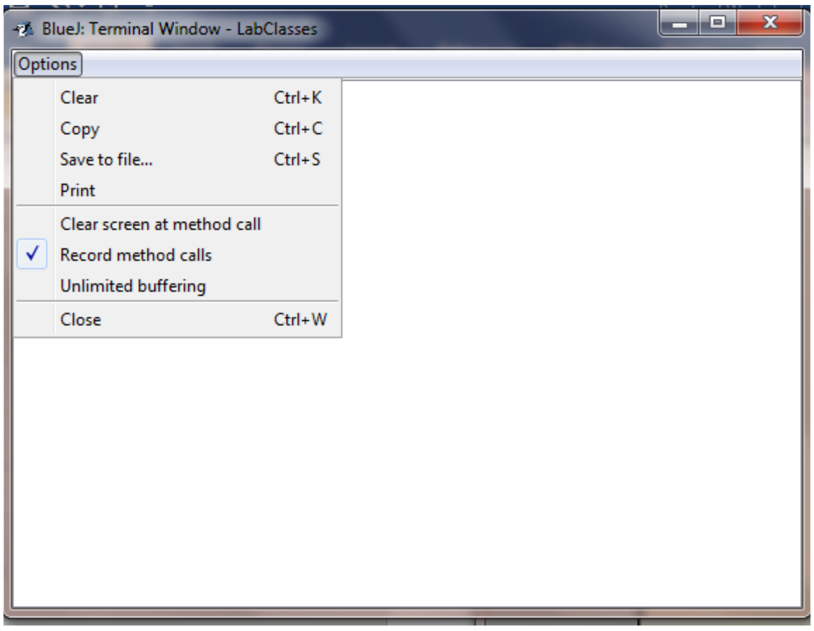
During the previous practical session, you started to use BlueJ. During this practical you will continue to use BlueJ to create and use a classes. The example used within this practical will not be completed today – it will be used at other points in the unit to demonstrate new concepts.

Work through this practical session at your own speed. Make sure you understand everything included within the example before you move onto the questions at the end of the worksheet. If you have questions, either ask your practical tutor, or ask a question using the anonymous question form on Moodle.

2. Example – Lab Classes

Start by opening BlueJ and create a new project called "LabClasses". During today's practical, we will be creating the Student class. This class will store the student's name, their ID number, and how many credits they have passed.

Before you start creating the class, we need to make sure that the Terminal Window is recording method calls. In the View menu, select "Show Terminal". Within the options for the terminal, make sure that "Record Method Calls" is selected.



We are going to start by creating the "Student" class. Click the "New Class" button, and call it "Student". The student class stores the student name, the student ID, and the number of credits studied for (which will initially be set to 0). Therefore, we need to create three instance variables: name (String), id (String), and creditsPassed (int). Using these instance variables, we can develop the constructor of the class, which takes the student's name and ID as parameters, and sets the credits to 0.

Double click on the class to open the editor. Delete the existing code and enter the following (comments are not required, unless they will support your revision):

```java
public class Student
{
    // instance variables
    private String name;
    private String idNumber;    // string so that the ID number can start with a 0 if needed
    private int creditsPassed;

    // constructor for student class
    public Student(String studentName, String studentID)
    {
        name = studentName;
        idNumber = studentID;
        creditsPassed = 0;
    }

}
```

Now that we have created the variables and instantiated them through the constructor, we can consider the accessor and mutator methods for each of the variables.

For the student name, we need both accessor and mutator methods as we will need to get the student name, and require the ability to change it.

The `getName()` method will require a return type of String (as this is the type of the variable name) and does not require any parameters. Enter the following code after the constructor and before the final curly bracket:

```java
// method to get the student's name
public String getName()
{
    return name;
}
```

The `changeName()` method will update the student's name. It will not require anything to be returned, so will have a void return type. In addition, it will require a String parameter, which is the new student's name. Enter the following code after the `getName()` method.

```java
// method to change the student's name
public void changeName(String newName)
{
    name = newName;
}
```

As the student ID cannot be changed, therefore, only an accessor method will be required. Enter the following code beneath the `changeName()` method.

```java
// method to get the student's id number
public String getStudentID()
{
    return idNumber;
}
```

The user will need to be able to get the number of credits which have been passed, and also to increase the number of credits obtained. Therefore, both accessor and mutator methods will be required. Enter the following code after the `getStudentID()` method:

```java
// method to get the number of credits passed by the student
public int getCreditsPassed()
{
    return creditsPassed;
}

// method to increase the number of credits which have been passed by the student
public void addCreditsPassed(int numberOfCredits)
{
    creditsPassed += numberOfCredits;
}
```

Now that all the accessor and mutator methods have been developed for each variable, we need to consider any other operations of the class. In this case, we require two additional methods: one to get the student's computer login name, and another to print all the information related to the student.

The `getLoginName()` method will require a return type of string – as this is the format of the login name. The login name follows the format of "UP123456" (i.e. "UP" and their student ID number). No parameters will be required, as all the information needed is contained within the class

```java
// method to get the student's computer login
public String getComputerLogin()
{
    return "UP" + idNumber;
}
```

The `printInformation()` method will display the student name, ID number, the computer login name, and the number of credits passed. This method will have a void return type, as it will not require anything to be returned.

```java
// method to print the information about the student
public void printInformation()
{
    System.out.println("Student Information");
    System.out.println("Student name: " + name);
    System.out.println("Student ID: " + idNumber);
    System.out.println("Student computer login: " + getComputerLogin());
    System.out.println("Number of credits passed: " + creditsPassed);
}
```

The completed Student class should be as follows:

```
public class Student
{
    private String name;
    private String idNumber;
    private int creditsPassed;

    public Student(String studentName, String studentID)
    {
        name = studentName;
        idNumber = studentID;
        creditsPassed = 0;
    }

    public String getName()
    {
        return name;
    }

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

    public String getStudentID()
    {
        return idNumber;
    }

    public int getCreditsPassed()
    {
        return creditsPassed;
    }

    public void addCreditsPassed(int numberOfCredits)
    {
        creditsPassed += numberOfCredits;
    }

    public String getComputerLogin()
    {
        return "UP" + idNumber;
    }

    public void printInformation()
    {
        System.out.println("Student Information");
        System.out.println("Student name: " + name);
        System.out.println("Student ID: " + idNumber);
        System.out.println("Student computer login: " + getComputerLogin());
        System.out.println("Number of credits passed: " + creditsPassed);
    }
}
```

Compile the class and correct any errors which are flagged up by BlueJ.

Create an object using the Student class which represents you. Run each of the methods and inspect the output to ensure they have the anticipated functionality. You will notice in the terminal window that the method calls have been recorded in red.

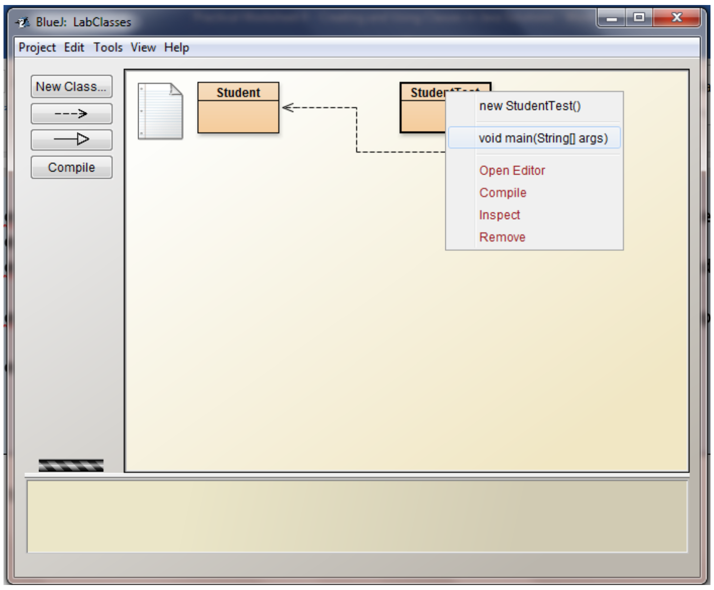
We can now create another file which we can use to run all the methods within the Student class. This will demonstrate how we can create and use objects within Java.

Create a new class called "StudentTest". Delete the existing code and enter the following:

```
public class StudentTest
{
    public static void main(String[] args)
    {
        // create a new student object - you will notice that the variable type is the
        // name of the class

        Student student1;
        student1 = new Student("Bob", "123456");

        System.out.println("Name before: " + student1.getName());
        student1.changeName("Alice");
        System.out.println("Name after: " + student1.getName());

        System.out.println("Student ID: " + student1.getStudentID());

        System.out.println("Credits before: " + student1.getCreditsPassed());
        student1.addCreditsPassed(20);
        System.out.println("Credits after: " + student1.getCreditsPassed());

        System.out.println("Student Computer Login: " + student1.getComputerLogin());

        student1.printInformation();
    }
}
```

Compile the class to check for errors.

Right click the StudentTest block, and select "void main(String[] args)" option. Then click OK in the popup box. This will run the testing method.



Try to guess the outputs before running the method. Were you correct?

3. Practice Questions

The following questions will allow you to practice developing class files in Java. For each new project, write a test file (like the StudentTest file in the example) which will create at least one object, and check that the functionality of that object is operating as anticipated. If you are adding to a previously developed class, make sure you update the relevant testing file to take into account the new methods.

1. Add another instance variable to the Student class which stores the student's favourite subject. You will need to update the constructor to take this new variable as a parameter. The user needs to be able to get the favourite subject, and change the favourite subject. In addition, you will need to update the `printInformation()` method to reflect this new instance variable.

2. In the Student class, add another method which returns true or false depending on whether the student has passed based on the number of credits they have obtained. You can assume that students require 120 credits to pass the year. The method should return true if they have passed, and false if they have failed.
   Hint: look at the `withdraw()` method from the Account class in the previous practical

3. In BlueJ, create a new project called "HeaterExercise". You will need to create a "Heater" class that will contain a single instance variable, which is called "temperature". Write the constructor which takes no parameters, and initially sets the temperature to 15^{o}C. Write an accessor method for the temperature. Finally, write two methods: `warmer()` and `cooler()` which increases or decreases the temperature by 5^{o}C each time it is called.

4. Modify the Heater class so that there is another instance variable called "increment", which is set to 5 within the constructor. You will also need to add the accessor and mutator methods for the increment variable. Adapt the `warmer()` and `cooler()` methods to take the new instance variable into account.

5. What happens if you change the increment so that it is a negative number? Adapt the mutator method so that the user cannot change the increment to a negative number.

6. Add two more instance variables to the Heater class: one to represent the minimum temperature and another to hold the maximum temperature. Update the constructor so that both instance variables are instantiated using parameters. You will also need to write the relevant accessor and mutator methods. Modify both the `warmer()` and `cooler()` methods so that these new instance variables are used in the appropriate places.

7. Create a new project called "CatExercise". Within this project, create a class called "Cat". The Cat class stores the cat's name, age, fur colour, and disposition (whether it is happy or sad). Create the constructor for the Cat class, assuming that it is happy when an object is created. Write the relevant accessor and mutator methods for the instance variables. In addition to the accessor and mutator methods, the cat also has methods to make it purr when it is happy and miaow. The method for purring will print "purr purr purr purr" in the Terminal window. The miaow method will print "miaow miaow miaow miaow" in the Terminal window.

8. Create a new project called "DogExercise". You will need to create a "Dog" class which stores the following information: the dog's name, the weight in kilograms, and preferred dog food. Develop the constructor for this class, and the relevant accessor and mutator methods. The dog will also have methods which simulates the dog barking (printing "woof woof woof woof" in the Terminal window) and calculates how much food to feed the dog based on the following rules:
   - If the preferred food is Pedigree Chum, feed the dog 4g of food for every 750g the dog weighs
   - If the preferred food is Barker's Chicken, feed the dog 8g of food for every 750g the dog weighs
   - If the preferred food is Woofer's Beef, feed the dog 10g of food for every 750g the dog weighs
   - Otherwise, feed the dog 6g of food for every 750g the dog weighs