**Week 2 Lab A: Creating a small Java class**

**Purpose of this lab:**

Show how to use BlueJ ( Some basic knowledge of BlueJ is assumed). 

Reinforce work on Classes and Objects.

**Introduction**

In this lab we are going to build a simple Student class to be used in a simple student administration system. The admin system needs to keep the following information on a student: 

Full name;

ID;

Number of credits taken.

Initially a Student object will contain the student’s name as a String and ID (Question: What type of data should this be?).

The number of credits will be set to 0.

During the operation of the admin system it may be necessary to obtain the student’s name, the ID and the current number of credits. Because it is possible that a student may change their name, by getting married say, the class should allow for the name to be changed. It will also be necessary, from time-to-time to add to the number of credits.

Another function is also required: printing out the student’s details in a readable formatted output.

**Task 1**

Analyse the requirements for Student and write out its responsibilities.

**Initial Actions**

*Setting up the project*

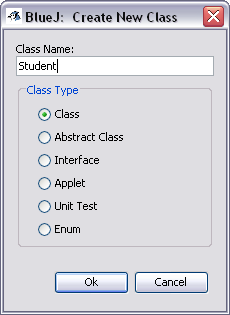
On your machine start BlueJ. The actual steps will depend on the configuration of your PC.

Click on *Project* in the menu bar and then select *New Project*. You will be presented with a file dialogue box. Select the location on disc where you want to keep the project. Type in a name for the project. Click the create button. You have now created the project folder.

**Student Class**

*Creating the Student Class*

In BlueJ click the New Class button. In the resulting dialogue box type the word Student. This is the name of the class we are going to produce. In Java the convention is to start the name of a class witha capital letter. The rest of the name will be lower case. Make sure the radio button for Class is selected and then click the OK button.



BlueJ will now create the class Student

based on the standard template that

comes with BlueJ. This standard template

is intended to help you remember how to

layout the Java in a class. Much of it will

need to be replaced as you build you

Student class.

Click on the class box, open the editor and

examine the code.

**Task 2**

*Building your Student class*

Edit the header comments so that they describe the purpose of the Class, add your name to the

@author tag and put in the date for the @version tag. You should do this every time you create a new class.

Next you need to identify and declare the three attributes of a student we need for the purpose of this exercise. The form of a declaration is:

private type name ;

Attributes are declared private in order to ensure that a client class does not accidentally change

their values in a non-controlled way.

The following shows the start of the class:

public class Student

{

**// the student's full name**

private String name;

**// the student ID**

private String id;

**// the amount of credits for study taken so far**

private int credits;

}

Next you need to consider the constructor for Student. It is good practice to initialise all the

attributes when a new object is being created. This should be done in a constructor.

Using constructors makes the creation of an object easier. It also provides the possibility of flexibility since we may have several constructors. (More on this later)

**Constructor** 

Used to create new objects 

Must have same name as class 

May have parameters 

No return type 

May have several but each must have a unique

signature 

Constructor with no parameters called no-arg

constructor

Write your constructor. It should receive values for the name & id attributes via the constructor’s parameter list.

Write your constructor:

|  |  |
| --- | --- |
| public Student (String fullName, String studentID)  {  name = fullName;  id = studentID;  credits = 0;  } |  |

If we do not define a constructor Java will assume a default constructor. However, once we

define a constructor of our own the default is only available if we write it.

Thus in our example

Student aStudent = new Student();

Is NOT Allowed because we have provided a constructor. The following is permitted:

Student aStudent = new Student(“Fred Bloggs”,”123456”);

Now we start to consider the services our Student class should provide. For this exercise we are

going to have the following accessor (query) methods:

public String getName()

public String getID()

public int getCredits()

We leave these as an exercise for you. Do not forget the Javadoc commenting.

Additionally there are various mutator (command) methods to be provided. Because spelling mistakes can happen we have decided to allow a user of the class to change the student’s name.

|  |  |
| --- | --- |
| public void changeName(String replacementName)  {  name = replacementName;  } |  |

There will also be a method

public void addCredits(int additionalPoints)

to permit the value of credits to be increased.

We will provide a method print() which will print the students name and id number to the terminal window.

|  |  |
| --- | --- |
| public void printStudent()  {  System.out.println(name + “ (“ + id + “)”);  } |  |

Implement each of these methods.

*In many OO programs output is placed in a separate class so that changes to the destination of the output can be made more easily. For the moment, for the sake of convenience, we will ignore this convention*.

Finally, we will provide a method that returns the login name of this student. The login name is a combination of the first four characters of the student's name and the first three characters of the student's ID number.

|  |  |
| --- | --- |
| public String getLoginName()  {  return name.substring(0, 4) + id.substring(0, 3);  } |  |

This example demonstrates the use of the String method substring(startIndex, stopIndex).

|  |
| --- |
| **NOTE:**   * The individual characters of a String can be accessed by their position in the string starting at index position 0 (zero) so that the string “program” is 7 characters long; ‘p’ is at position 0 and ‘m’ is at position 6. * substring in this case takes two arguments:   + startIndex – the index position from which to start the sub string   + stopIndex – the index position ***after*** the last you require. So if the value contained in name was “Peter”; using substring(0, 4) you would get “Pete” (the characters at positions 0, 1, 2 and 3). |

Once you have entered the code for Student you should compile it. In fact if you have been sensible you will have been compiling as you have been adding code! It is useful to sort out syntax errors as you write the code.

**Task 3**

*Ensuring Student does what it is meant to*

When you compile a class you are just checking to ensure we have made correct use of Java. You are looking for and correcting any syntax errors.

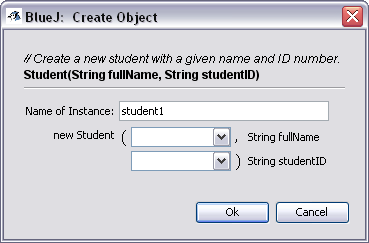
Before starting to look at the next class you need to be confident that your Student class is logically correct, i.e. it does what it is supposed to do. You must exercise the Student.

Prepare a test plan, i.e. write out a list of the actions you will perform and the data you will use to convince yourself that the class works as required.

|  |  |
| --- | --- |
| Can create student? | y |
| Can add credits? | y |
| Can change name? | y |
| Can fetch login name, id, credits | y |
| Print student info? Name, | y |

This is where BlueJ’s facilities come into their own.

Start by right-clicking on the Student class in the BlueJ window to create a new object. You will be presented with the following dialogue box:



Type in a name for the student and

an id. Since both are Strings they

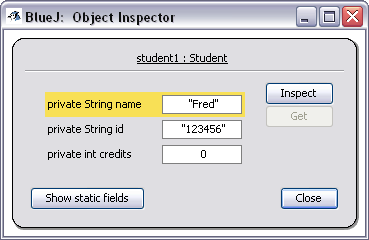
should be enclosed in double

quotes, e.g. “Fred”

Click the ok button

You will then have created a Student object called student1. It is placed in the object workbench and appears as a red box.

If you right click on student1 you will be presented with a list of methods you could execute plus theoption to ***Inspect***the object*.* Right click on Inspect and you should see the following :



This shows the current state of student1. Keeping this box open, click on student1 and start to

execute the methods.

Start with the accessors and ensure they return the expected values. Include the print method in with this group. Once you are convinced the accessors are correct start to exercise the methods that set values.

As you execute the mutator methods the values displayed in the Object Inspector box will change appropriately.

If you find any bugs correct them.