

|  |
| --- |
| **COURSEWORK ASSESSMENT SPECIFICATION** |

|  |  |
| --- | --- |
| **Module Title:** | *Programming 2* |
| **Module Number:** | *KV4001* |
| **Module Tutor Name(s):** | *Alan Maughan* |
| **Academic Year:** | *2018/19* |
| **% Weighting (to overall module):** | *7.5%* |
| **Coursework Title:** | *Homework 2* |
| **Average Study Time Required by Student:** | *5 hours* |

**Dates and Mechanisms for Assessment Submission and Feedback**

|  |
| --- |
| **Date of Handout to Students:**  Week 5 |
| **Mechanism for Handout to Students:**  *via elp* |
| **Date and Time of Submission by Student:**  During Week 7 Lab Class |
| **Mechanism for Submission of Work by Student:**  Papers collected in lab |
| **Date by which Work, Feedback and Marks will be returned to Students:**  Marks & feedback will be given as the assessment is marked in the lab week 7. |
| **Mechanism for return of assignment work, feedback and marks to students:**  Marks & feedback will be given as the assessment is marked in the lab |

**Further Information**

*(Please ensure the assessment specification includes the following items)*

**Learning Outcomes tested in this assessment (from the Module Descriptor):**

1. Make use of predefined classes in the development of programs.
2. Understand and develop simple Object-Oriented programs.
3. Understand the different strengths / weaknesses of both the procedural and the object-oriented programming paradigms.

**Nature of the submission required:**

Paper copies of source code. Code execution in lab.

**Instructions to students:**

*This is an individual piece of work.*

**Referencing Style:**

*N/A*

**Expected size of the submission**:

Printed copies of source code

**Academic Conduct:**

You must adhere to the university regulations on academic conduct. Formal inquiry proceedings will be instigated if there is any suspicion of misconduct or plagiarism in your work. Refer to the University’s regulations on assessment if you are unclear as to the meaning of these terms. The latest copy is available on the university website.

# Homework 2

This work is due to be marked at the start of your lab in week 7. It counts for 7.5% towards the module mark.

You must bring with you a printed copy of your source code (the .java files). These should be produced before you come to the lab. Do not come to the lab and attempt to print a copy then. The code will be collected by a tutor when they mark your homework (make sure that it has your name / id on it - @author your name / your id). This will be retained for audit and internal moderation. If this file is not available (and printed before the lab) then you will score 0 (zero)!

You must work on the program on your own, outside any formal classes and it must be ready to execute at the start of the scheduled laboratory class. All code must be completed using the BlueJ IDE. Any work utilising other IDEs will score zero.

All code (in this and in all subsequent homeworks) must:

* Have the class header and all methods commented to ‘Javadoc’ standards using @author, @version, @param and @return tags as appropriate.
* Be coded to required layout (e.g. indentation) and naming standards.

Notes on the above were supplied in week 1 of KV4000 and there are numerous examples in the code you will have seen. Failure to meet these standards will result in loss of marks.

You may be asked questions about your program to confirm your understanding and that it is your own work. Failure to answer the questions may result in a deduction or total loss of marks.

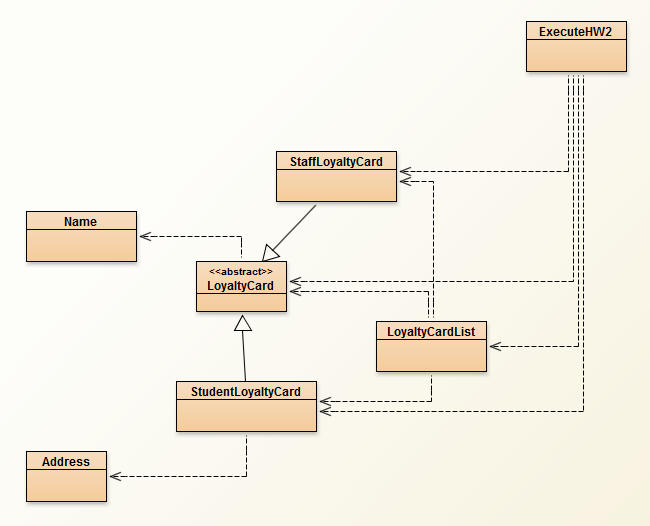
***The work must be wholly your own. Collusion counts as academic misconduct and will be punished according to the University’s regulations detailed in “Assessment Regulations for Taught Awards” (ARTA) a copy of which is available on the University website.***

## The Task

This homework is intended to test your understanding of inheritance, polymorphism and abstract classes. NOTE: This homework includes material from weeks 3 - 5. You may start the homework now but you will not be able to complete it until after the week 5 classes.

For this homework, you are expected to write a program representing subtypes of LoyaltyCard. A LoyaltyCard may be either a **StaffLoyaltyCard** or a **StudentLoyaltyCard**. Both should inherit from the **LoyaltyCard** class. However, you should not be able to create instances of type **LoyaltyCard** and your code should ensure this.

The structure of the program is:

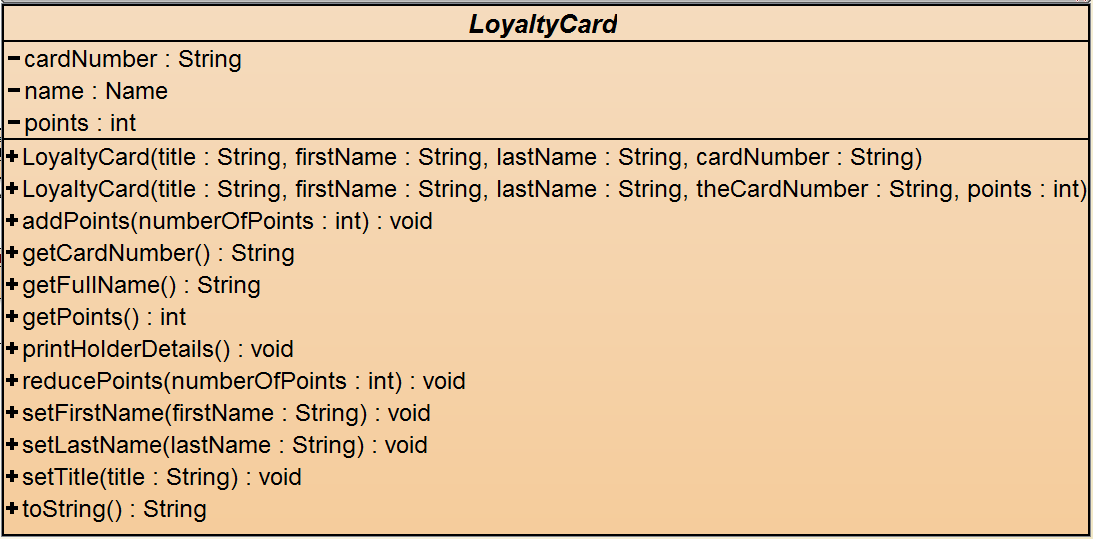


The Name and Address classes are provided for your use. They must be used and ***may not be*** amended.

You must write the classes LoyaltyCard, StaffLoyaltyCard, StudentLoyaltyCard and LibraryList. The LoyaltyCard class is different from that used previously. Please ensure that the classes you write conform to the class diagrams and the style guide.

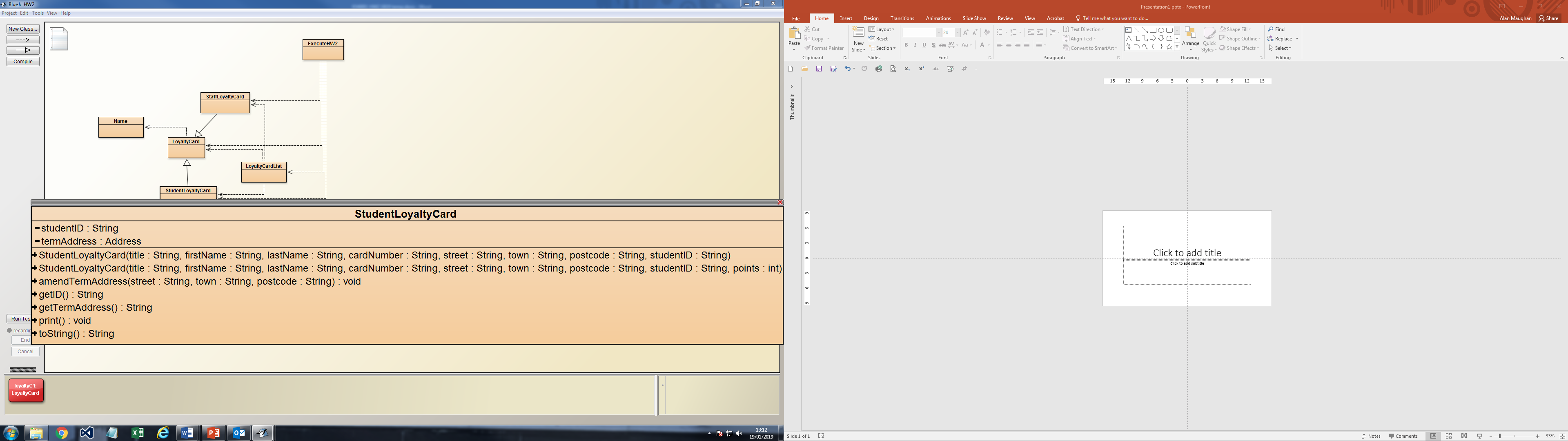
**ExecuteHW2** is supplied and will be used to test elements of your program. An explanation of this class is given below.

**LoyaltyCard**



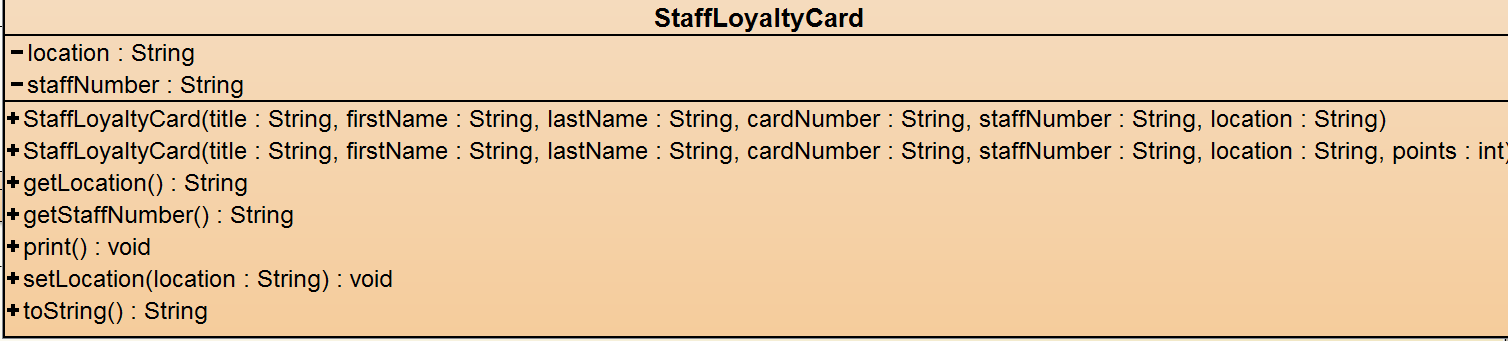
|  |  |
| --- | --- |
| getCardNumber | Simple accessor. No output required. |
| getFullName | Returns the full name using functionality in the Name class |
| getPoints | Simple accessor. No output required. |
| addPoints | Should update the points held. It should also output to the terminal window the message: Points now: [the revised points value] |
| reducePoints | Should update the points held if this would not reduce the number of points below zero. It should also output to the terminal window the message: Points now: [the revised points value].  If the points would be reduced below zero, instead it should output the message: Transaction refused. Only [points] points available. |
| setFirstName | Simple mutators. No output required. |
| setLastName |
| setTitle |
| toString | Should return a formatted string in the form:  [title] [firstname] [lastname]  Card Number: [cardnumber]  Points available: [points] |
| printHolderDetails | Should print the formatted string to the terminal window. |

**StaffLoyaltyCard – sub type of Loyalty Card**



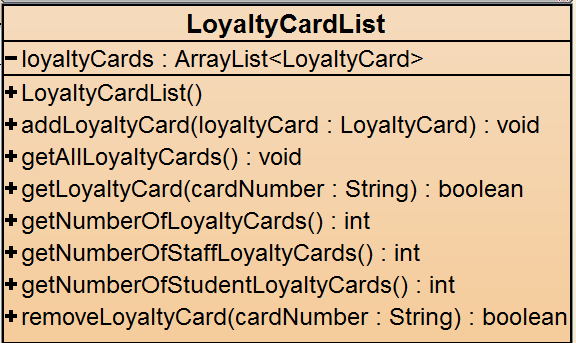
|  |  |
| --- | --- |
| amendTermAddress | Simple mutator utilising Address class functionality. No output required. |
| getID | Simple accessor. No output required. |
| getTermAddress | Simple accessor utilising Address class functionality. No output required. |
| toString | Returns a formatted string in the form:  [title] [firstname] [lastname]  Card Number: [cardnumber] // student cards should be characters beginning ‘stu’ e.g. stu12345  Points available: [points]  Student ID: [id] // 8 characters e.g. 12345678  [street] [ town] [postcode] |
| print | Outputs the formatted string to the terminal window |

**StudentLoyaltyCard – sub type of LoyaltyCard**

****

|  |  |
| --- | --- |
| getLocation, getStaffNumber, setLocation | Simple accessor / mutators. No output required |
| toString, print | Similar to student version. E.G.:  [title] [ firstname] [lastname]  Card Number: [cardnumber] // 8 characters beginning ‘sta’ e.g. sta12345  Points available: [points]  Staff Number: [staffnumber] // 5 characters e.g. 12345  Location: [location] // string e.g. CIS001 |

**LoyaltyCardList**



|  |  |
| --- | --- |
| addLoyaltyCard | Add a staff or student card |
| getAllLoyaltyCards | Print details of all cards to the terminal window (in the order they were added to the collection). |
| getLoyaltyCard | Print either details of the specified card or, if not found  "Loyalty Card with card number: [cardNumber] not found."  Return true of false accordingly. |
| getNumberOfStaffLoyaltyCards, getNumberOfStudentLoyaltyCards, getNumberOfLoyaltyCards | Return the number of cards belonging to staff, students, or both as an int. |
| removeLoyaltyCard | Remove the specified card from the collection if found then return true of false accordingly. |

**ExecuteHW2**

ExecuteHW2 will be used to test your program and is supplied so you can test your solution. A class with a different name (but the same composition) may be supplied to your lab class to ensure that no one can modify this class.

* The variables **id** and **name** should hold your details and are passed to the constructor.
* The variable **mark** will hold the result of running the program.
* **runTests()** will call each of the other methods in turn to test that aspect of the expected functionality. At present each method call and the methods themselves are commented out. It is suggested that you implement each of these tests in turn removing the commenting as you go. If you cannot implement a test then ensure that the relevant code remains commented out.

Marking Scheme: (12Marks / 7.5% of module total)

|  |  |
| --- | --- |
| Mark | Criteria |
| **No serious attempt / Does not compile - will score zero** | |
| -1 | Unsatisfactory use of Javadoc, appropriate layout, etc. |
| 9 | Successfully performs each test |
| 2 | Both subtypes correctly extend and use **LoyaltyCard** functionally |
| +1 | All above correct |