NEWCASTLE UNIVERSITY

ASSESSMENT 2020-21

CSC8404 Advanced Programming in Java

Submission deadline: 18.30 on Friday 6th November 2020

Submit your completed solution and doc (MS Word, or pdf) via NESS, as a single zip file.

Instructions:

This assessment contributes 100% towards the total mark for this module. This is an individual exercise: You must not work with anyone else to produce any parts of the solution or documentation.

CSC8404 Advanced Programming in Java - Assessed Coursework

The deliverable for this coursework must be submitted via NESS on or before:

- **18.30 on Friday 6th November 2020**

This is a hard deadline. Work uploaded after this time (even by 1 second) will be flagged as late. Make sure you submit AT LEAST 15 minutes before this deadline.

This coursework is worth 100% of CSC8404's assessment.

---

1. Aim

The aim of this coursework is for you to practice the design principles covered in lectures. You will develop interfaces and classes to demonstrate that you have learned and understood the module material, including:

- appropriate overriding of `Object` class methods, including overriding `toString` and providing a static `valueOf` method when appropriate
- design of interface-based hierarchies, programming through interfaces and late binding
- the use of factories to control instantiation of objects, including how to guarantee the instantiation of unique instances
- defensive programming including the use of immutability
- the use of appropriate interfaces and classes from the Collections framework

The coursework is **not** algorithmically challenging. The focus is on good design and good practice.

The coursework is **not about development of an end-user application**. **You are developing interfaces and classes that could be used for the development of an application.** You should **not** develop a graphical user interface or a command line interface. They are not necessary and you will be given no credit for doing so. **You should provide test cases for your interfaces and classes.** You should submit both your solution and the classes that you use to test the solution.

Note the student support system specified below is a deliberate simplification. It is not an accurate model of real world University systems. **Your solution should correspond to the simplicity of the specification.** You risk losing marks if you attempt to provide a more realistic model or provide a solution that is more complicated than necessary.

---

2. System overview

A University Department needs a set of interfaces and classes to manage student data.

The Department has different types of students. These are undergraduate (UG), postgraduate taught (PGT) and postgraduate research (PGR) students. Students cannot be more than one type. For this coursework, the significant difference between undergraduate and postgraduate taught students is that undergraduate students take 120 credits worth of courses in a year whereas postgraduate taught students take 180. Furthermore, the pass mark for undergraduate modules is 40% but for postgraduate taught modules the pass mark is 50%. Postgraduate research students have a supervisor but do not register for modules.

The University needs to maintain a record of modules and supervisors for students in an academic year. The system should allow module and supervisor information to be read from appropriately defined data files. The files should contain one data entry per line with fields separated by a comma e.g. CSC8404, Advanced Programming, 20. The system should allow an appropriate number of modules to be added to a student record and to record whether or not a student is correctly registered (are they taking the right number of credits or do they have a supervisor allocated).

In addition, the University needs to be able to issue smart cards and login ID's to all students on their courses. The following provides more detail on the required functionality:

noOfStudents (typeOfStudent)
: This method returns the number of students of the specified type that are currently enrolled.

registerStudent (Student)
: This method registers a new student onto the system and allocates a student ID (see below).

amendStudentData(studentID, studentData)
: This method changes a student record.

terminateStudent (studentID)
: This method removes the student record associated with the given student number. In effect, the student is leaving the University.

When issuing a smart card, the following rules must be observed.

- An undergraduate student must be at least 17 years old.
- A postgraduate student must be at least 20 years old.
- A student cannot be issued with more than one smartcard (i.e. do not try to deal with lost cards!).

3. Implementation

To complete the system outlined in Section 2 you will need to provide interfaces and classes for the functionality described in this section. You must also use Junit testing to unit test your solution.

Student

All students have the following **public** functionality:

- a method to get the student's ID.
- a method to get the student's type (UG, PGT, or PGR).
- a method to list the modules that the student is registered for. A module consists of a name (e.g. Advanced Programming in Java), a module code (e.g. CSC8404) and the number of credits associated with the module (e.g. 20).
- A method which returns true if the student is currently registered for enough credits (120 for undergraduate, 180 for postgraduate taught, 0 for postgraduate research) and false otherwise.

- Postgraduate research students have a method to return the name of their supervisor (a Name)

You must provide an appropriate hierarchy for students.

Student ID

A student ID has two components - a single letter followed by a four digit number. For example:

- a1234

You must provide access to each component and an appropriate string representation of the ID.

Student ID's are unique. You must guarantee that no two students have the same ID.

Smart Card

A Smart Card has the student's name (comprising a first and last name), the date of birth of the student, a unique Smart Card number and a date of issue.

The Smart Card number has three components. The first component is the concatenation of the initial of the first name of the student with the initial of the last name of the student. The second component is the year of issue of the card. The third component is an arbitrary serial number. For example, the string representation of the Smart Card number for a card issued to John Smith in 2018 would have the form:

- JS-2018-10

where the 10 is a serial number that, with the initials and year, guarantees the uniqueness of the card number as a whole.

Your smart card class must provide methods to access the student's name, the student's date of birth, the student ID and the date of issue of the Card.

You should provide appropriate classes for a student's name and for a smart card number.

You must guarantee the uniqueness of smart card numbers.

You should use the **java.util.Date** class to represent dates. However, you must not use deprecated methods of the Date class. So, for example, in your test classes use **java.util.Calendar** to construct dates of birth and dates of issue of Smart Cards. You can assume default time zone and locale. Note: Java may provide a much more satisfactory way of handling dates and time with immutable classes. However, in this coursework I want to see that you can use sub-optimal classes such as Date, and so its use in this project is **mandatory**.

The smart card should have the following **private** method:

- setExpiryDate(); which sets an expiry date for the card. If the smart card is held by a UG student, the expiry date is set to the issue date plus four years. If the smart card is held by a PGT student, the expiry date is set to the issue date plus two years. If the smart card is held by a PGR student, the expiry date is set to the issue date plus five years.

The smart card should have the following **public** method:

- getExpiryDate(); which returns the expiry date of the card.

---

4. Deliverables

Your solution should include your interfaces and classes that comprise the implementation of the system and types outlined in Sections 2 and 3. You must annotate your code with appropriate Javadocs. In addition, you should provide separate unit test classes that demonstrate thorough testing of your solution.

Also, write a document (MS Word, or PDF) that explains the structure of your overall design. This document should have a picture containing UML diagrams of all your classes and interfaces and their relationships depicted by annotated arrows (extends, implements, uses). For each class and interface, the UML diagram should show the class/interface name, fields and methods (where appropriate). Also, each diagram should indicate whether it is a class, abstract class or an interface.

You must submit your solution through NESS as a single zip archive that contains your Java source code files and your document.

---

5. Assessment & mark allocation

Marks will be allocated for:

- Overall structure (e.g. interfaces, classes and their relationships), **15%**
- Correct implementation of rules specified in Sections 2 and 3, as well as for choice and use of maps and collections, **40%**.

- Following good practice guidance: maintenance of invariants and defensive programming, use of immutability, appropriate overriding of `Object` methods, use of Javadoc comments, **25%**.
- Evidence of testing by implementation of appropriate test classes that test the normal case, boundary conditions, and exceptional cases. **20%**.

---

6. Style guidelines

Adopt a consistent style, do not violate naming conventions (when to use upper/lower case letters in names) and make appropriate use of whitespace (indentation and other spacing).

---

7. Further notes and Hints

Start early!! This is not a project to hack together during the last 1-2 days before the deadline.

Break the coursework down into separate tasks. Start with the simpler classes first (e.g. Module, Name, StudentID, SmartCardNumber and SmartCard) but leave the imposition of uniqueness and immutability until we cover these topics in lectures. You can implement the different types of student before implementing the management system class. Unit test classes as you progress through the coursework.

For each class you implement you should consider:

1. whether to override Object methods (equals, toString etc.),
2. whether to use an interface-based hierarchy, and
3. whether the class should be immutable.
4. whether to use an object factory.

You may have to defer parts of the coursework (or the implementation of certain aspects of a class) until we have covered material in lectures. In this case, you can make a start with a simpler solution that can be extended later.

For any questions, please email: Ellis.Solaiman@ncl.ac.uk