UNIVERSITY OF EDINBURGH

COLLEGE OF SCIENCE AND ENGINEERING

SCHOOL OF INFORMATICS

# INFR08014 INFORMATICS 1 - OBJECT-ORIENTED PROGRAMMING

**Friday 6 th May 2016**

**09:30 to 11:30**

## INSTRUCTIONS TO CANDIDATES

1. **Note that all questions are compulsory.**
2. **Remember that a file that does not compile, or does not pass the simple JUnit tests provided, will get no marks.**
3. **This is an Open Book exam. You may bring in your own material on paper. No electronic devices are permitted.**
4. **CALCULATORS MAY NOT BE USED.**

Convener: D. K. Arvind
External Examiner: C. Johnson

THIS EXAMINATION WILL BE MARKED ANONYMOUSLY

1. You are given, in files `Course.java` and `Student.java`, the code of classes `Course` and `Student`. Examine the code of these classes, but do not change it.

   Your task is to implement the subclass `UG1Student` of `Student`, representing a special kind of `Student` who is in the first year of their degree.

   (a) Define the class `UG1Student`, extending `Student`. It should have a private instance variable `mainSchedule` of type `char`. This represents the student's home school ('O' for Informatics, for example). [*10 marks*]

   (b) Write two public constructors for `UG1Student`. Your first constructor must take (in this order) a `String` which is the student's name, a `String` which is the student's UUN, and a `char` which should become the value of the student's `mainSchedule` attribute. Invoke the `Student` constructor as appropriate: of course, this student's year is 1. Also write a zero argument constructor that invokes your first constructor with arguments "not set", "not set", 'X'. [*10 marks*]

   (c) Write a public instance method `addCourse`, taking a `Course` representing a course that this student wishes to enroll on, and returning a `boolean` indicating success or failure. Since UG1 students are only permitted to take courses at level 7 or level 8, your code must first check that the level of the course is either 7 or 8. If it is, pass the course to the superclass's `addCourse` method, and return the result; if not, simply return false. [*10 marks*]

   (d) Write a public instance method `addCourses`, taking an array of `Course` objects representing a list of courses that this student wishes to enroll on, and returning a `boolean` indicating overall success or failure. You may assume that the array, and any `Course` objects in it, are not `null`, but do not assume anything about the length of the array. Use your `addCourse` method to add each course in turn. The return value must be the conjunction of all the return values from `addCourse`: that is, `true` if every `addCourse` succeeded, `false` if any failed. [*10 marks*]

   (e) Write a public instance method `toString`, which takes no arguments and returns a `String` which must begin with the `String` returned by `Student`'s `toString` method, followed by a line describing the student's main schedule, followed by a list of the courses *from that schedule* that the student is taking, each string being produced by `Course`'s `getName` and placed on a new line. For example, with the current versions of `Student` and `Course`, possible output is

```
David Parnas
Main schedule O courses:
Informatics 1 Object Oriented Programming
```

*QUESTION CONTINUES ON NEXT PAGE*

*QUESTION CONTINUED FROM PREVIOUS PAGE*

Here the middle line is your responsibility and must be produced exactly as shown (with the appropriate character for the student's actual `mainSchedule`, of course). The first and third lines are produced by code in `Student` and `Course`: if their developers change their implementations later, the output of your code must automatically incorporate the changes. Do not add a new line at the very end.

[*10 marks*]

The file you must submit for this question is `UG1Student.java`. Before you submit, check that it compiles and passes the basic JUnit tests provided, otherwise it will get 0.

2. In this question you will make a program `Beads` which will take a list of integers on the command line. Imagine that each of these numbers is painted onto a bead, and the beads are threaded onto a circular wire. The class's methods relate to the integers that can be made by adding the numbers painted on one or more adjacent beads.

   In this question you will use the collection class `ArrayList` which is familiar from lectures. You will also use `HashSet`, which is an implementation of `Set`. You may need to look at the JDK documentation for these. Note that the types of your methods will involve `Set`, thus hiding, from clients, which implementation of the `Set` interface is used; your code is expected to create, concretely, `HashSet`s.

   You may assume that none of the arguments to your methods are `null`.

   (a) Write a public static method `sums` that takes an `ArrayList` of `Integer`s representing the beads (which your code must not alter), and an `int`, say $n$, representing the number of beads to be summed, and returns a `Set` of `Integer`s representing the numbers that can be formed by adding up exactly $n$ adjacent beads. If $n < 1$, or the list of beads is empty, return an empty `Set`.

   Examples of correct behaviour:

   - `sums(<1,2,3,4>,2)` $= \{3, 5, 7\}$ because $1 + 2 = 3$, $2 + 3 = 4 + 1 = 5$, $3 + 4 = 7$.
   - `sums(<1,1,1>,4)` $= \{4\}$ because $1 + 1 + 1 + 1 = 4$ (that is, using the same bead twice is allowed, if the second argument is greater than the length of the first argument).
   - `sums(<6,1,3>,1)` $= \{6, 1, 3\}$.
   - `sums(<6,1,3>,0)` $= \{\}$.

   *[20 marks]*

   (b) Write a public static method `allSums` that takes an `ArrayList` of `Integer`s, say $a$, again representing the beads (which your code must not alter). By using your method `sums`, with the same `ArrayList` it has been given and each integer in turn from 1 to the length of $a$, `allSums` must return a `Set` of `Integer`s representing the numbers that can be formed by adding up any number of adjacent beads; this time using the same bead twice is not allowed, i.e. you can't add up more beads than there are.

   Examples of correct behaviour:

   - `allSums(<1,2,3,4>)` $= \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$.
   - `allSums(<1,1,1>)` $= \{1, 2, 3\}$.
   - `allSums(<6,1,3>)` $= \{1, 3, 4, 6, 7, 9, 10\}$.

   *[10 marks]*

*QUESTION CONTINUES ON NEXT PAGE*

(c) Write a public static method `findMax` that takes a `Set` of `Integers`. It calculates and returns the greatest positive `int` $n$ such that every integer from 1 to $n$ inclusive occurs in the set. If 1 does not occur in the set, return 0.

Examples of correct behaviour:

- `findMax`$(\{1, 2, 4\}) = 2$.
- `findMax`$(\{6\}) = 0$.
- `findMax`$(\{1, 3, 4, 6, 7, 9, 10\}) = 1$.

[*10 marks*]

(d) Write a `main` method, with the usual header, which

- expects, as command line arguments, a list of integers, to represent the values of beads on a circular wire as above;
- builds an `ArrayList` of `Integers` from the arguments;
- uses your methods to find the greatest positive integer $n$ such that every integer from 1 to $n$ inclusive can be made by summing some adjacent beads (or 0 if 1 cannot be made);
- prints this integer $n$ (and nothing else, except for optional whitespace).

You are not required to write code to handle incorrect arguments.

Thus, for example, if the program is run from the command line as

```
java Beads 6 1 3
```

the output should be

```
1
```

[*10 marks*]

The file you must submit for this question is `Beads.java`. Before you submit, check that it compiles and passes the basic JUnit tests provided, otherwise it will get 0.