UNIVERSITY OF EDINBURGH

COLLEGE OF SCIENCE AND ENGINEERING

SCHOOL OF INFORMATICS

INFR08014 INFORMATICS 1 - OBJECT-ORIENTED PROGRAMMING

Friday 6 th May 2016

14:30 to 16:30

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 `UG5Student` of `Student`, representing a special kind of `Student` who is in the fifth year of their degree.

   (a) Define the class `UG5Student`, extending `Student`. It should have a private instance variable `extraLevel9s` of type `int`. [*10 marks*]

   (b) Write two public constructors for `UG5Student`. 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 an `int` which should become the value of the student's `extraLevel9s` attribute. Invoke the `Student` constructor as appropriate: of course, this student's year is 5. Also write a zero argument constructor that invokes your first constructor with arguments "not set", "not set", 0. [*10 marks*]

   (c) Write a public instance method `level9s`, which takes no arguments and returns an `int` representing the number of level 9 courses that this student is taking. [*10 marks*]

   (d) 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. UG5 students are only normally permitted to take one level 9 course, or more with special permission; e.g. a student with `extraLevel9s = 1` could take two level 9 courses. Your code must first check that the student can take this course without exceeding this limit. If they can, pass the course to the superclass's `addCourse` method, and return the result; if not, simply return false. [*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. If the student is taking zero or one level 9 courses, there is nothing to add (even if the student has permission to take extra level 9 courses). If the student is taking more than one level 9 course, add a line saying so, followed by a list of *all* the courses that the student is taking (regardless of level), 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
Taking extra level 9s. Permission for 0 extra.
Operating Systems
Informatics 1 Object Oriented Programming
Logic Programming
Advanced Vision
```

QUESTION CONTINUES ON NEXT PAGE

Here the second line is your responsibility and must be produced exactly as shown (with 0 replaced by the student's actual `extraLevel9s`, of course). The other 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 `UG5Student.java`. Before you submit, check that it compiles and passes the basic JUnit tests provided, otherwise it will get 0.

2. *Remark: This question relates to a famous proof of Fermat's Little Theorem. However, you are not expected to know this result or its proof.*

You will make a program `Fermat` which will take an integer and a list of strings on the command line. Think of the strings as colour names. If the program is run as

```
java Fermat 3 red green
```

then we are investigating the bead necklaces that can be made using 3 beads, each of which can be red or green.

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 also find methods in the standard JDK class `Collections` useful.

Efficient code is not required: full marks may be obtained provided your methods return promptly when the number of beads and colours are both around 4.

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

(a) Write a public static method `threadings`, which takes an `int n` (representing the number of beads on each necklace) and a `Set` of `String`s (representing the available bead colours; your code must not alter this `Set`), and returns a `Set` of `ArrayList`s of `String`s, representing all the orders in which `n` beads of the given colours can be threaded. If `n < 1`, return a `Set` containing just one, empty, `ArrayList`.

Examples of correct behaviour:

- `threadings(0, {red,green}) = {[]}`
- `threadings(1, {red,green}) = {[red],[green]}`
- `threadings(2, {red,green})`
  `= {[red,red],[red,green],[green,red],[green,green]}`
- `threadings(3, {red}) = {[red,red,red]}`

Hint: you will probably want `threadings` to call itself recursively, although full marks are available for any correct method. [*20 marks*]

(b) Write a public static method `isEquiv`, which takes two `ArrayList`s of `String`s, representing two orders in which beads can be threaded, and returns true if they represent the same necklace, false otherwise. Your code may assume the two arguments have the same length. It must not alter its arguments.

QUESTION CONTINUES ON NEXT PAGE

QUESTION CONTINUED FROM PREVIOUS PAGE

Examples of correct behaviour:

- `isEquiv([red,red],[red,green]) = false` (different beads)
- `isEquiv([red,green,blue,pink],[blue,pink,red,green]) = true` (rotate the necklace)
- `isEquiv([red,green,blue,pink],[red,pink,blue,green]) = true` (turn the necklace over)
- `isEquiv([red,green,red,green],[red,red,green,green]) = false` (same beads, but not the same necklace)

[*10 marks*]

(c) Write a public static method `analyse`, which takes a `Set s` of `ArrayLists` of `Strings` representing some orders in which beads may be threaded, and identifies those orders which are singletons, that is, not equivalent (in the sense of the `isEquiv` method) to any other order in `s`. It returns no value, but prints to standard output. Your code must not alter its argument, and it may assume that both the `Set` and any `ArrayLists` in it are not `null`.

Examples of correct behaviour:

- `analyse({[red,red],[red,green],[green,red]})` should print

```
[red, red] is a singleton
```

- `analyse({[green,red],[red,green]})` should print nothing at all.
- `analyse({[red,red],[red,green]})` should print

```
[red, red] is a singleton
[red, green] is a singleton
```

The order in which the lines are printed does not matter, but each line should be exactly as shown. Note that `System.out.println`'s usual method of printing `ArrayLists` of `Strings` can be used. [*10 marks*]

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

- expects, as command line arguments, an integer followed by a list of strings, to represent the number and possible colours of beads on a necklace as above;
- builds an `int` and a `HashSet` of `Strings` from the arguments;
- invokes `threadings` on these values;
- invokes `analyse` on the result returned from `threadings`.

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 Fermat 3 red green
```

the output should be

QUESTION CONTINUES ON NEXT PAGE

```
[red, red, red] is a singleton
[green, green, green] is a singleton
```

(or the same lines in the other order). [*10 marks*]

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