UNIVERSITY OF EDINBURGH

COLLEGE OF SCIENCE AND ENGINEERING

SCHOOL OF INFORMATICS

INFR08014 INFORMATICS 1 - OBJECT-ORIENTED PROGRAMMING

Wednesday 1st May 2013

09:30 to 12:30

INSTRUCTIONS TO CANDIDATES

1. **Note that all questions are compulsory.**
2. **This is an Open Book exam.**

Convener: J Bradfield
External Examiner: A Preece

THIS EXAMINATION WILL BE MARKED ANONYMOUSLY

Question 1

In this question you will implement part of a simple project tracking application. **Important: your work will be marked partly by automated tests, so you must follow the instructions precisely. For example, where your program produces strings, use exactly the format specified.**

(a) Implement a class `Engineer` as follows. `Engineer` must have private attributes `name : String` and `salary : int` and the following public interface:

`public class Engineer`

`Engineer(String name, int salary)`	*constructor*
`String getName()`	*getter*
`void setName(String name)`	*setter*
`int getSalary()`	*getter*
`void setSalary(int salary)`	*setter*
`String toString()`	*return "Name (Salary)" e.g. "Jo Bloggs (15000)"*

[15 marks]

(b) Implement the class `Manager` to extend `Engineer`. This class should have an additional private attribute `team` of type `ArrayList<Engineer>`, initialised, in the constructor, to be empty. (Remember you will need the line

```
import java.util.ArrayList;
```

at the top of your file.) Provide a getter and a setter method for this new attribute. Override the method `toString()` so that it gives the manager's own name, then on a new line by itself gives "Manages:", then lists the members of the `Manager`'s team exactly as the team member's own `toString()` does, each on a new line. For example, if Sue Smith, on a salary of 30000, manages Jo Bloggs, on a salary of 15000, then sending `toString()` to an object representing Sue Smith should result in:

```
Sue Smith (30000)
Manages:
Jo Bloggs (15000)
```

In summary `Manager` has this public interface, with behaviour as explained above:

`public class Manager`

`Manager(String name, int salary)`
`ArrayList<Engineer> getTeam()`
`void setTeam(ArrayList<Engineer> team)`
`String toString()`

[20 marks]

(c) Next, implement class `ProjectDate`. This has two private integer attributes, `weekNo` and `dayNo`, to represent the date of a project relative to its start date. E.g. if a `ProjectDate` has `weekNo==3` and `dayNo==4`, it represents day 4 of week 3 of the project. Your code must ensure that `dayNo` is between 1 and 5 inclusive. (In contrast, `weekNo` may be any integer, including negative; a negative `weekNo` represents a date before the official project start date.)

Your class must have a zero-argument constructor that sets `weekNo` to 0 and `dayNo` to 1, and getters and setters for its attributes. If the setter for `dayNo` is given an argument outside the acceptable range, it should simply do nothing: that is, it must not change the object, but it must not report this in any way as a mistake.

Your class must also have a method `advance` taking a positive integer `n` and returning nothing. It modifies the state of this ProjectDate object by moving it forward in time by `n` days. Again, if the argument is negative, the method should simply do nothing.

Finally the `toString` method should return "Week: 3 Day: 4" if `weekNo` is 3 and `dayNo` is 4, and so on.

In summary `ProjectDate` has this public interface, with behaviour as explained above:

```
public class ProjectDate
-------------------------
        ProjectDate()
    int getWeekNo()
   void setWeekNo(int n)
    int getDayNo()
   void setDayNo(int n)
   void advance(int n)
 String toString()
-------------------------
```

[*15 marks*]

The files that you must submit for this question are the following:

- `Engineer.java`
- `Manager.java`
- `ProjectDate.java`

Question 2

Note: in this question code that fails to compile will receive no credit.

In each of parts (a)–(c) below, you will be asked to supply the body to a method inside a class `Lens`. You will be given a skeleton file for this class. You should add your definitions of the methods at the points marked as follows:

```
// ADD CODE HERE
```

In software engineering, a lens manages the consistency between a *source* piece of data, $s \in S$, and an abstracted *view* of it, $v \in V$. It comprises three functions:

- get : $S \to V$ which, given a source, returns the view of that source (this typically involves throwing some information away);
- create : $V \to S$ which, given a view, returns a source of which this could be the view (this typically involves filling in some missing information with default values)
- put : $S \times V \to S$. Given a source and an *updated* version of the view of that source, this returns an updated version of the source.

In this question you will implement and test the three functions `get`, `create` and `put` for a lens between arrays of `String`s and arrays of `Pair`s, where each `Pair` comprises a character (type `char`) and a natural number (type `int`). You are given the class `Pair`: study it.

Note: it is expected that you may not be familiar with the Java base type `char`. You should consult the Java documentation whenever you need to. Hint: you are likely to need the `String` constructor that takes an array of `char`s, and the `charAt` method of `String`.

(a) In the class `Lens`, implement the public static method

```
Pair[] get(String[] source)
```

that returns an array of Pairs the same length as the input array of Strings. Each Pair corresponds to the String at the same position. The Pair contains the first character of the string and the length of the string. You may assume that each String has length greater than 0.

Expected behaviour:

```
get(new String[] {"foo", "bar", "froboz"})
should return the array of Pairs
{['f',3],['b',3],['f',6]}
```

[*10 marks*]

(b) In the class `Lens`, implement the public static method

```
String[] create(Pair[] view)
```

that returns an array of `String` the same length as the input array of `Pair`s. Each `String` corresponds to the `Pair` at the same position. The `String` must have the initial character and length indicated by the `Pair`. Assume that the integer in each `Pair` will always be at least 1. Extra characters must be filled in with default character 'X'.

Expected behaviour:

```
create (new Pair[]
        {new Pair('f',3), new Pair('b',3), new Pair('f',6)})
should return the array of Strings
{"fXX", "bXX", "fXXXXX"}
```

[*10 marks*]

(c) In the class `Lens`, implement the public static method

```
String[] put(String[] oldSource, Pair[] newView)
```

that expects `oldSource` and `newView` to have the same length (say n) – this check is done for you in the code provided – and returns an array of strings of length n, which is the new source.

It must truncate the `oldSource` string, or pad it with 'X's, so that it has the length indicated in the `Pair`. Assume that the integer in each `Pair` will always be at least 1.

Expected behaviour:

```
put (new String[] {"foo", "bar", "froboz"},
     new Pair[] {new Pair('f',3), new Pair('b',3), new Pair('f',3)})
should return the array of Strings
{"foo", "bar", "fro"}

put(new String[] {"foo", "bar", "froboz"},
    new Pair[] {new Pair('f',3), new Pair('b',6), new Pair('f',3)})
should return the array of Strings
{"foo", "barXXX", "fro"}
```

If the first character of a `Pair` in `newView` is not the first character of the corresponding string in `oldSource`, your code should behave the same way as if the length check failed. Use the provided length-checking code as an example, modifying the error message to "First characters don't match". [*20 marks*]

(d) One important law that some lenses obey is called GetPut. In this part you will implement a class `InvestigateLens` that tests this law on your `Lens` class.

(i) The GetPut law says: for every source s and for every view v, `get(put(`s`,`v`))=` v.

Implement a static method of `InvestigateLens` called `checkGetPut` that takes a source (type `String[]`) and a view (type `Pair[]`) and returns true if and only if the GetPut law holds for these particular values. [*5 marks*]

(ii) Implement a `main` method for `InvestigateLens` that invokes the method you implemented in parts (i) on the following inputs:

source s	view v
{"foo"}	{['f',7]}
{"foo", "bar", "froboz" }	{['f',7], ['b',3], ['f',6]}
{"foo", "bar", "froboz" }	{['f',2], ['b',5], ['f',3]}

Your method must print `true` or `false` to standard output to show whether your lens passes each test, and finally, whether it passes all three tests and therefore may (as far as these tests show) obey the law. Each result must be on a new line, and nothing else must be printed. For example, if your lens fails GetPut on all three given inputs and therefore definitely does not satisfy the GetPut law for all inputs, your output should be

```
false
false
false
false
```

[*5 marks*]

The files that you must submit for this question are the following:

- `Lens.java`
- `InvestigateLens.java`

Final Checklist

Here is a complete list of all the files that you must submit for this exam:

```
Engineer.java
Manager.java
ProjectDate.java
Lens.java
InvestigateLens.java
```