UNIVERSITY OF EDINBURGH

COLLEGE OF SCIENCE AND ENGINEERING

SCHOOL OF INFORMATICS

INFORMATICS 1 - OBJECT-ORIENTED PROGRAMMING

Wednesday 20 May 2009

09:30 to 11:30

Convener: M O'Boyle
External Examiner: R Irving

INSTRUCTIONS TO CANDIDATES

1. **Note that ALL QUESTIONS ARE COMPULSORY.**
2. **DIFFERENT QUESTIONS MAY HAVE DIFFERENT NUMBERS OF TOTAL MARKS. Take note of this in allocating time to questions.**

THIS EXAMINATION WILL BE MARKED ANONYMOUSLY

1. In each of parts (a)–(c) below, you will be asked to supply the body to a method inside a class. There will be a separate class for each part, named `OneA`, `OneB` and `OneC` respectively. You will be given a skeleton file for each of these classes, and the skeleton will contain the appropriate method declaration. You should add your definitions of the methods at the points marked as follows:

```
// ADD YOUR CODE HERE
```

(a) Implement the method `boolean sharedEnd(int[] a, int[] b)` in the class `OneA` (skeleton file supplied). Given two arrays of `int`s, this should return `true` if they have the same first element **or** they have the same last element. The arrays can be of different lengths but should contain at least one element.

Expected behaviour:

```
sharedEnd(new int[]{1, 2, 3}, new int[]{7, 3}) -> true
sharedEnd(new int[]{1, 2, 3}, new int[]{1, 3, 2}) -> true
sharedEnd(new int[]{1, 2, 3}, new int[]{7, 3, 2}) -> false
sharedEnd(new int[]{1, 2, 3}, new int[]{2, 3}) -> true
```

[14 marks]

(b) Implement the method `int[] sumNext(int[] nums)` in the class `OneB` (skeleton file supplied). Given an array `nums` of `int`s, this returns a modified array, using the following transformation: for each adjacent pair of elements (starting at the beginning of the array), sum the two elements, then replace the first element with that sum and replace the second element with 0. Thus, the first element is replaced with the sum of the first and second elements, the second is replaced with 0, the third element is replaced by the sum of the third and fourth elements, and so on. You can assume that the array is of length two or greater, and is always of even length. Return the modified array.

Expected behaviour:

```
sumNext(new int[]{2, 3, 3, 0}) -> {5, 0, 3, 0}
sumNext(new int[]{5, 6}) -> {11, 0}
sumNext(new int[]{2, 3, 3, 0, 7, 1}) -> {5, 0, 3, 0, 8, 0}
```

[15 marks]

(c) Implement the method `boolean sumAndDivide(int one, int two, int three)` in the class `OneC` (skeleton file supplied). This returns the value `true` if and only if the following holds: if the sum of `one` and `three` is greater than `two`, then `two` is divisible by 3 without remainder.

Expected behaviour:

```
sumAndDivide(3, 3, 8) -> true
sumAndDivide(3, 9, 9) -> true
sumAndDivide(3, 8, 9) -> false
sumAndDivide(3, 6, 0) -> true
sumAndDivide(3, 5, 0) -> true
```

[*15 marks*]

(d) The class `QuestionOneLauncher` (skeleton file supplied) has a single `main()` method. Inside `main()`, add calls to each of the methods you have defined in parts (a)–(c) above to test that your implementations produce the correct results. You can write your tests in any way you think appropriate, but you should have at least two tests for each method. You are *not* required to write a test for every input shown in the 'expected behaviour' boxes. [*6 marks*]

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

(a) `OneA.java`

(b) `OneB.java`

(c) `OneC.java`

(d) `QuestionOneLauncher.java`

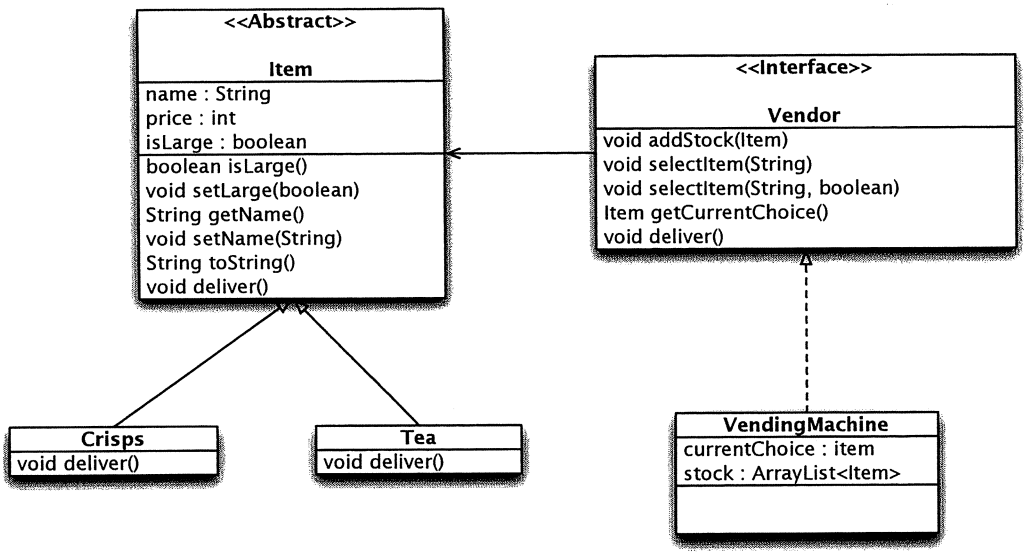

Figure 1: Diagram of the Vending Machine

2. This question involves the construction of a simulated Vending Machine. In the simulation, the machine sells only crisps and hot tea; however the design is intended to accommodate a wider variety of items, belonging to the abstract class `Item`. The design of the simulation is shown in Figure 1.

   You are given the classes `Item` and `Crisps`, and you are also given the interface `Vendor`, shown here:

```
public interface Vendor {

     void addStock(Item item);
     void selectItem(String name);
     void selectItem(String name, boolean isLarge);
     Item getCurrentChoice();
     void deliver();
 }
```

   In addition, you are given a launcher class `VendingMachineLauncher`

   You are required to write two new classes: `VendingMachine`, which implements `Vendor`, and the class `Tea`. This task is broken down in more detail below, starting with `VendingMachine`.

   (a) Declare two instance variables: `currentChoice` of type `Item` and `stock`, which is an `ArrayList` of `Items`. In addition, implement the method `public void addStock(Item item)` which adds an `Item` to the `stock`. [*5 marks*]

(b) Implement the method `public void selectItem(String name)`. Given the arguments `"crisps"` or `"tea"`, this searches through `stock` to find the first matching object `o` of type `Item`. If it finds one, then `o` becomes the value of the variable `currentChoice`, otherwise the latter is set to be `null`. [*15 marks*]

(c) Implement an overloaded method `public void selectItem(String name, boolean isLarge)`. This should behave in the same way as `public void selectItem(String name)`, but in addition, it should call the `setLarge()` method of `currentChoice` with the argument `isLarge`.

Also implement the method `public Item getCurrentChoice()` which just returns `currentCoice`. [*10 marks*]

(d) Implement the method `public void deliver()`. This checks whether the value of `currentChoice` is `null`. If it is, it prints out the following string to the console: `"Sorry, out of stock!"`. Otherwise, it calls the `Item`'s `deliver()` method, removes it from `stock`, and sets `currentChoice` to be `null`. [*10 marks*]

(e) Implement the class `Tea`, following the pattern of `Crisps`. The `deliver()` method of an instance of `Tea` should print out different strings to the console, depending on the value of `isLarge()`. If the latter returns `false`, the string should be `"Pouring into a small cup"`; otherwise, it should be `"Pouring into a large cup"`. [*10 marks*]

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

- `VendingMachine.java`
- `Tea.java`