University of Glasgow

Tuesday, 29 April 2014
2.00 pm – 3.30 pm
(1 hour 30 minutes)

DEGREES of MSci, MEng, BEng, BSc, MA and MA (Social Sciences)

COMPUTING SCIENCE 2P: JAVA PROGRAMMING 2

Answer all 3 questions

This examination paper is worth a total of 60 marks.

The use of calculators is not permitted in this examination.

For examinations of less than 2 hours duration, no candidate will be permitted to exit during the examination.

1. For each of the following pairs of Java concepts, briefly comment on the differences between them. You may provide source code fragments if they help to strengthen your argument.

(a) an abstract class and an interface. [2]

(b) a static field and an instance field in a class. [2]

(c) the `final` and `finally` keywords. [2]

(d) the values `0x1e2f` and `1e2f`. [2]

(e) the `while` and `do ... while` constructs. [2]

(f) method overloading and method overriding. [2]

(g) the `this` and `super` keywords. [2]

(h) the `null` and `void` keywords. [2]

(i) the `byte` and `char` primitive types. [2]

(j) the `protected` and `public` visibility modifiers. [2]

2. Consider using object-oriented concepts in Java to model random events such as dice throws or coin flips. Note that you may make *reasonable assumptions* in your answer, so long as you state each assumption explicitly.

(a) Define a `RandomEventGenerator` interface which has two methods. The `nextEvent()` method returns an `int` value representing the outcome of the next random event. The `isFair()` method returns a `boolean` value to indicate whether the outcomes are uniformly distributed (i.e. all outcomes are equally likely). [3]

(b) How would you modify the `RandomEventGenerator` interface to allow event outcomes to be represented as arbitrary `Object` types, in a type-safe manner? [3]

(c) Now consider class `FairDice`, which is an implementation of the `RandomEventGenerator` interface that models an *n*-sided fair dice. Assume that *n* is encapsulated as a final instance field of type `int` in the class. Give sensible definitions for the `FairDice` constructor and the two methods that must be implemented from the `RandomEventGenerator` interface. Assume that the outcomes are boxed `Integer` objects with values between 1 and *n* inclusive. You may use Java library methods such as Math.random() in your solution. [7]

(d) Now imagine another implementation of `RandomEventGenerator` that represents coin flips. The outcomes are either Heads or Tails. Outline an appropriate way to model these outcomes in Java. [2]

(e) Suppose that the `CoinFlip` class has an instance field called `headsProbability` of type `double`. The constructor ensures that this field has a value between 0.0 and 1.0 inclusive. Also assume that the `CoinFlip` class has two methods `headEvent()` and `tailEvent()` that return head and tail outcomes with the types specified in your previous answer. Now define the two methods `isFair()` and `nextEvent()` for the `CoinFlip` class. [5]

3. Please study the following Java source code before answering the question below.

```java
public class foo
{
  public int Bar(int i) {
    if (i==0) return 1;
     if (i==1) return 1;
    if (i==2) return 2;
    return (2 * Bar(i-2) + Bar(i-3));

  }
  }
```

(a) What is the sequence of return values when the `Bar` method is invoked with the arguments 0, 1, 2, 3, 4 in five successive method calls? [5]

(b) Suggest ways in which to improve the formatting of the source code shown above. Consider identifier names and source code layout in particular. [5]

(c) Suggest ways in which to improve the efficiency of the `Bar` method without changing the computed return values. Significant code rewriting is encouraged. [5]

(d) Why might the `Bar` method and similar methods be modified to save (input, return value) pairs into a lookup table such as a `HashMap`? [5]