University of Glasgow

Thisday, 1 Thismonth Thisyear
XX.XX am/pm XX.XX am/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.

You must not leave the examination room within the first half hour or the last fifteen minutes of the examination.

INSTRUCTIONS TO INVIGILATORS

Please collect all exam question papers and return to School together with exam answer scripts

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]

Solution: Any 2 points from::

- An abstract class defines or inherits abstract methods. [1]
- An interface only provides method signatures, i.e. all methods are abstract. [1]
- An abstract class may contain fields and non-abstract methods. [1].
- A class can implement multiple interfaces but may only extend one abstract class. [1].

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

Solution: Any 2 points from:

- A static field is per-class whereas an instance field is per-object. [1]
- A static field is declared with the `static` keyword.[1]
- A static field should be initialized in a `static` initializer whereas an instance field should be initialized in a constructor. [1]
- Static fields can be referenced via a class reference or an object reference (including `this`), instance fields can only be referenced via an object reference. [1]

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

Solution: Any 2 points from:

- final is a modifier for classes, methods and fields. [1]
- final prevents inheritance (classes), overriding (methods) or updating (fields). [1]
- `finally` is a control-flow keyword, linked to a `try` statement [1]
- Code in the finally block is guaranteed to execute when the try block completes, or after any exceptions thrown in the try block have been caught, or before any exceptions thrown in the try block have been propagated to the calling scope. [1]

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

Solution: Any 2 points from:

- 0x1e2f has hexadecimal 32-bit integer type [1]
- 1e2f has 32-bit floating point type [1]
- value of hex number is 7727 [1]
- value of FP number is 100.0 [1]

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

Solution: Any 2 points from:

- while loops execute 0 or more times, depending on associated condition [1]
- do...while loops execute 1 or more times, depending on associated condition [1]
- while evaluates condition before loop body, do/while evaluates condition after loop body [1]

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

Solution: Any 2 points from:

- an overloaded method has multiple methods with the same names but different signatures (i.e. different numbers and/or parameter types) [1]
- an overriding method is a method in a subclass that has the same signature as a method in a superclass. [1]
- overriding is supported by object polymorphism. [1]
- in JVM, can overload on return type, but can only overload on parameter types in Java language [1]

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

Solution: Any 2 points from:

- this refers to object in its current class context [1]

CONTINUED OVERLEAF

- super refers to object in its superclass context [1]
- can use both as pseudo-references for instance members. [1]
- can use both as constructor calls e.g. `this()` [1]

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

Solution: Any 2 points from:

- null is an empty reference value, may be assigned to any reference type [1]
- void indicates an absence of type - used only in method signatures for methods that do not return any value [1]
- (would give [1] mark for) null is a value, void is a type.

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

Solution: Any 2 points from:

- char is a 16-bit unicode character type
- byte is an 8-bit integer type
- char is unsigned, byte is signed

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

Solution: Any 2 points from:

- protected members are visible anywhere in the current package scope, and in any subclass. [1]
- public members are visible anywhere [1]
- public is more visible than protected [1].

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]

Solution:

```java
interface RandomEventGenerator {   // 1 mark
   int nextEvent(); // 1 mark
   boolean isFair(); // 1 mark
}
```

(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]

Solution: Accept the best suggestion from:

- suggest using `Object` instead of `int` for return type of `nextEvent()`. [1].
- suggest defining another interface `RandomEvent` which event outcome classes would have to implement. Then `nextEvent()` will have this return type. [2].
- Suggest using Java generics. i.e. RandomEventGenerator<T> with return type of `nextEvent()` set to T. [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]

Solution:

```java
public FairDice(int n) {   // 1 mark
  this.n = n;   // 1 mark
}

@Override // 0.5 marks
public boolean isFair() {
  return true; // 1 mark
```

```
        }

        @Override // 0.5 marks
        public Integer nextEvent() {
          /* 1 mark for new Integer construction
           * 1 mark for randomness
           * 1 mark for sensible arithmetic
           */
          return new Integer((int)(Math.random()*n) + 1);
        }
```

(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]

Solution: Best solution of:

- String values “heads” and “tails” [0.5]
- static final integer values in the CoinFlip class [1]
- enumeration values, either in CoinFlip class or separate CoinFlipOutcomes enum [2]
- static final instances of a CoinFlipOutcome class [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]

Solution:

```
        @Override
        public boolean isFair() {
          return (this.headsProbability==0.5);   // 1 mark
        }

        @Override
        public CoinFlip nextEvent() {
          double d = Math.random();       // 1 mark
          if (d<this.headsProbability) { // 1 mark
            return headEvent();           // 0.5 marks
          } else {                     // 1 mark
```

```
            return tailEvent();          // 0.5 marks
        }
    }
```

CONTINUED OVERLEAF

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

```
    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]

Solution: This is the Fibonacci sequence, so the first five calls will yield 1, 1, 2, 3, 5 respectively. [1] mark per correct answer.

(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]

Solution: Any five from:

- Consistent layout of braces [1].
- Consistent indentation of statements [1]
- Class names should be camel case, with initial caps [1]
- method names should camel case, with initial lower case [1]
- Class name and method name could be more informative [1]
- Method name should really be `fib` [1]
- brackets for compound arithmetic not required or could be better placed around multiplication to show precedence [1]

(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]

Solution: Max 5 marks from the points below:

- make method `static` [1], to avoid virtual method call overhead [1]

- change computation to f(n-1) + f(n-2) [2]
- handle bad input sensibly, when parameter is negative [1]
- transform recursive computation into equivalent iterative computation, e.g. using a `for` loop [3]
- use a closed form solution to avoid recursion, involves golden ratio, floating point arithmetic... [3]

(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]

Solution: Stretch question! Any 5 marks from:

- pure function [1]
- stateless class - output depends only on input [1]
- expensive function to compute, involves recursion [1]
- lookup at second invocation to prevent repeat [1]
- trade off space (to save precomputed values) for time (to recompute values) [2]
- this is a form of caching for repeated computation [2]
- key,value store is common optimization [1]
- this is function memo-ization [1]

END OF QUESTION PAPER