University of Glasgow

Thursday 7 December 2017
16:30 – 17:30
(1 hour)

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

COMPUTER SCIENCE 2P: JAVA PROGRAMMING 2

Answer all 3 questions

This examination paper is worth a total of 50 marks.

The use of calculators is not permitted in this examination.

INSTRUCTIONS TO INVIGILATORS: Please collect all exam question papers and exam answer scripts and retain for school to collect. Candidates must not remove exam question papers.

1. This question deals with the Java programming language. (20 marks total)

(a) Consider the following Java method:

```
public static void main (String[] args) {
  System.out.println("Hello world");
}
```

Answer the following questions in the context of the above method:

(i) What is special about a method with the given signature? [1]

> **Solution:** If a class has a method with this precise signature, then the class can be run as a Java application.

The remaining questions deal with the above method signature in general, and do not refer to any special properties of the exact method above.

(ii) What effect does the **`public`** keyword have on this method? What would be the difference if it were removed? [2]

> **Solution:** **`public`** means that the method is visible to any other Java code in any class or package. (1 mark) No modifier (i.e., "default" or "package-private" visibility) means that it is visible only to other classes in the same package. (1 mark)

(iii) What effect does the **`static`** keyword have on this method? What would be the difference if it were removed? [2]

> **Solution:** **`static`** means that this method can be called without needing to create an instance of the enclosing class. (1 mark) With no such modifier, the method could only be called on an instance. (1 mark)

(iv) What is the meaning of **`void`** in the method signature? What would be the effect if it were removed? [2]

> **Solution:** **`void`** represents the return value of the method – in this case, it indicates that the method does not return a value. (1 mark) If it were removed the code would not compile because all methods need a return type. (1 mark)

(v) What data type is `String[] args`? What is another way that same data type could have been written? [2]

> **Solution:** It is an array of strings. (1 mark) Alternative syntax options include the C-style `String args[]` and the varargs version `String... args`. (1 mark)

(b) These questions all deal with the Java-language concept of **reflection**.

(i) What is reflection? [2]

Solution: Reflection refers to the built-in ability to examine and modify a Java application at run-time.

(ii) State two benefits of using reflection. [2]

Solution: Possible answers:

- You can extend external classes
- You can use it to implement a class browser or IDE
- You can use it to support debugging and write test cases

(iii) State two potential costs of using reflection. [2]

Solution: Possible answers:

- It imposes a performance overhead
- It can be used to expose internal code that was not meant to be public

(c) In the context of **multi-threaded programming**, define the following terms:

(i) Deadlock. [1]

Solution: When two or more threads are blocked waiting for each other.

(ii) Livelock [1]

Solution: When multiple threads are constantly acting in response to each other's actions.

(iii) Starvation [1]

Solution: When a thread is unable to gain access to shared resources and does not progress.

(iv) Memory consistency error [1]

Solution: When different threads have inconsistent views of what should be the same data.

(v) Atomic action [1]

Solution: An action that either happens completely, or does not happen at all.

2. This question concerns the Java programming language. (15 marks total)

(a) The following Java class has no programming errors and will run correctly, but is badly written. Identify **five stylistic problems** with the class. [5]

```
import java.util.ArrayList;

public class MyClass {

  public static void main(String[] args) {
    ArrayList Values = new ArrayList();
    Values.add(1);
    Values.add(2);
    Values.add(3);
    Values.add(4);
    Values.add(5);

    int i = 1;
    while (true) {
      try {
        System.out.println(Values.get(i - 1));
        i++;
      } catch (IndexOutOfBoundsException ex) {
        break;
      }
    }
  }
}
```

Solution: Any five of the following (1 mark each)

- Variable name not starting with lower case letter
- Not using ArrayList in a generic way
- Repetitive code at the start where `Values` is filled in – should use a loop for this sort of thing
- Using an exception as a core part of the control flow
- Starting a counter at 1 and having to subtract 1 from it to use it as an index
- . . . any other valid criticism

(b) For each of the following Java code fragments, indicate **exactly** what will happen when it is compiled and executed. If it produces output, show the exact output; if it runs but produces an error, specify the error precisely; if it will not compile, describe what the problem is.

(i)
```
System.out.println (1 + 2 + " three " + 4);
```

[2]

Solution:

```
3 three 4
```

(ii)
```
double d = Double.MAX_VALUE;
System.out.println (d + d);
```

[2]

Solution:

```
Infinity
```

(iii)
```
int i = 6;
switch (i % 2) {
case 0:
   System.out.println("even");
case 1:
   System.out.println("odd");
default:
   System.out.println("something weird?");
}
```

[2]

Solution: Output:

```
even
odd
something weird?
```

(iv)
```
int[] numbers = { 1, 2, 3, 4, 5, 6, 7 };
int count = 0;
for (int i = 0; i <= numbers.length; i++) {
   count += numbers[i];
}
System.out.println("Count is " + count);
```

[2]

Solution: This code throws an `ArrayIndexOutOfBoundsException` and produces no output.

(v)
```
// File A.java
public class A {
}

// File B.java
public class B extends A {
}

// File C.java
public class C {
   public static void main (String[] args) {
      B b = new A();
   }
}
```

[2]

Solution: File C.java will not compile because an instance of type A cannot be assigned to a variable of its subclass B.

3. Java programming and object-oriented modelling. (15 marks total)

This question is about a hypothetical single-player interactive game called *SlimeCraft*. The game is played on a two-dimensional grid of squares. Each square has a unique (x, y) co-ordinate, where x and y are non-negative integer values. Squares are *adjacent* if they have an absolute difference of at most 1 in both their x and y co-ordinates.

The player occupies one square at any point in the game, but may move around between squares.

A square is associated with a substance, which indicates what that square contains. Substances are air and slime. A square is also associated with a player if the player occupies that square.

(a) Assume a class `Substance` with subclasses `Air` and `Slime`. Also assume a class `Player`. Now give the Java source code for the `Square` class. Include full definitions for instance fields. You should also provide a public constructor for `Square` objects. The constructor should automatically fill the square with a new `Air` object, and does not need to associate the square with any player object. You **do not** need to define getters and setters explicitly, or any other public methods. [4]

Solution:

```
public class Square { // 0.5 marks for class
   private Substance s; // 0.5 marks for substance
   private Player p; // 0.5 marks for player
   private int x;  // 0.5 marks for 2 int coordinates
   private int y;

   public Square(int x, int y) { // 0.5 marks for constructor
     this.{x,y} = {x,y};  // 0.5 marks for coord setting
     this.s = new Air();  // 0.5 marks for new Air
     this.p = null;       // 0.5 marks for explicit null assignment
   }

}
```

(b) Assume the SlimeCraft game board is of fixed size $N \times N$, where x and y coordinates range from 0 to $N-1$. Describe an appropriate data structure to represent the game board. You may use Java source code to declare the data structure, although this is not required. [2]

Solution: Either a two-dimensional array of Square references, i.e. `Square[][] board`. This may be encapsulated inside a `Board` class (optional). Equal credit for a two-dimensional `ArrayList<Square>`.

(c) Suppose the `Player` class implements the `Locatable` interface, which has `getX()` and `getY()` methods that return the integer coordinates of the current position of the player.

Write a method `surroundWithSlime()` that takes two parameters, a player and a board, and returns `void`. The method sets the contents of all squares that are *adjacent* to the current square so that they contain slime. Pay particular attention to the 'corner' cases if a player is near the edge of the board. You may assume the existence of appropriate constructors and getter/setter methods for any class without defining them explicitly. [8]

Solution: This is going to be a doubly nested for loop **(1 mark)**, over x and y in the appropriate ranges **(2 marks)**. To handle corner cases, the best option is to use special logic with loop control flow keywords – catching `ArrayIndexOutOfBoundsException` should not receive full marks **(3 marks)**. At each adjacent square, set the contents to be a new slime object **(2 marks)**. Example solution:

```
for (int i=x-1; i<=x+1; i++) {
  if (i<0 || i>=N) continue; // or break if >= N
  for (int j=y-1; j<=y+1; j++) {
    if (j<0 || j>=N) continue; // or break if >=N
    if (i==x && j==y) continue;
    board[i][j].setContents(new Slime());
    // or board.set(i,j, new Slime());
  }
}
```

(d) If there is only one player involved in the SlimeCraft game, why is it poor engineering practice for every `Square` object to have a `Player` reference? [1]

Solution: In an $N \times N$ board, there will be N^2 `Player` references. All but one of these will be `null`. This wastes memory—it's an extra word field per `Square` instance.