University of Glasgow

Monday 10 December 2018
1:00pm – 2:00pm
(Duration: 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. Answer the following questions about Java language concepts: (20 marks total)

(a) State four differences between an **abstract class** and an **interface**. [4]

Solution:

- Abstract classes can have a constructor
- A class can implement any number of interfaces, but can only have one superclass.
- Abstract classes can have member fields while interfaces can have only static fields
- Abstract classes can have protected and private members, while interfaces can have only public members

(b) If you call a method that throws a a **checked exception**, what are the two approaches for handling that exception? [2]

Solution:

- You could **catch** the exception
- You could **re-throw** the exception by adding it to the method signature

(c) What is the difference between the **static** keyword and the **final** keyword? [2]

Solution: **final** means that a member cannot be changed – a final class cannot be subclassed, a final method cannot be overridden, a final field cannot have its value changed. **static** means that a member is associated with the class as a whole rather than with an instance of the class.

(d) What is the role of the **generic parameter** in classes such as `ArrayList`? [2]

Solution: It specifies the type that will be used in the class at run-time.

(e) Name three properties of an enumerated type (i.e., one declared with the `enum` keyword) that are not true of a normal, non-enumerated type. [3]

Solution:

- Items can be compared with `==`
- Items can be used in **`switch`** statements
- The type has a fixed set of possible values and no new values can be created.

Also give points for mentioning about using ordinal positions, `valueOf()`, etc.

(f) Name and briefly describe the two basic classes involved in **file I/O** when using the `java.nio.file` package. [4]

Solution:

- `Path` represents a location in the file system at an abstract level
- `Files` contains a number of static methods which operate on `Path` instances to manipulate actual files and directories

One point for naming each class, one point for correctly describing each.

(g) What is the difference between the types **`int`** and `Integer`? How can you convert between the two types? What is one reason to choose `Integer` over **`int`**? [3]

Solution: **`int`** is a primitive type, while `Integer` is a class. (1 mark)

In most cases, conversion between the two types happens automatically through the mechanism of **boxing** and **unboxing**. (1 mark)

You need to use `Integer` if you want to use this as the generic type in an `ArrayList` or similar. (1 mark)

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

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

```
public boolean My_Method(String[] input) {
  boolean result = false;

  for (int zzz = 0; zzz < input.length; zzz++) {
    String s = input[i];
    if (s.equals("")) {
      result = true;
    }
  }

  return result;
}
```

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

- Method name not in CamelCase
- Method name does not describe what it does
- Non-standard variable name for the loop counter variable
- Using standard **`for`** loop instead of **for-each**
- Using `.equals()` without checking for **`null`** first
- Going through whole array before returning – could just return after finding first empty string
- . . . 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)
```
import java.util.Objects;
public class B {
   public String field;

   public B (String s) {
      this.field = s;
   }

   @Override
   public boolean equals (B b) {
```

```
      return Objects.equals(this.field, b.field);
   }

   public static void main (String[] args) {
      B b1 = new B("one");
      B b2 = new B("two");
      System.out.println(b1.equals(b2));
   }
}
```

[2]

Solution: This code will not compile because the signature for `equals()` is wrong – it does not override `Object.equals()` so the `@Override` annotation will fail.

(ii)
```
public class A {
   public int i = 3;

   public A(int val) {
      int i = val;
   }
}

A a = new A(5);
System.out.println(a.i);
```

[2]

Solution:

3

(because inside the constructor, `i` refers to a local variable, not a field)

(iii)
```
int i = 5.0;
int j = 10.0;
System.out.println(i / j);
```

[2]

Solution: This code will throw a `ClassCastException` because it is not possible to assign a double value to an int variable without casting.

(iv)
```
try {
   throw new Exception("Hello!");
   System.out.println("World!");
} catch (Exception ex) {
   System.out.println(ex.getMessage());
}
```

[2]

Solution: Output:

```
Hello
```

(v)
```
// File A.java
public class A {
   public A(int value) {
      System.out.println("Value is " + value);
   }
}

// File B.java
```

```
public class B extends A {
   public static void main (String[] args) {
      A a = new A(10);
   }
}
```

[2]

Solution: File B.java will not compile: Class `B` has no constructor defined, so it will attempt to call the nonexistent no-args constructor of `A`.

3. This question concerns Java class design. (15 marks total)

First, read the following description of an employee record system.

> You are to design a class to model the employees of a company. Every employee has an employee number, a family name, and a given name.
>
> In addition, each employee has a basic hourly rate, a number of contracted hours per month, and a number of hours actually worked (in the current month). Hourly paid workers are paid 1.5 times their normal rate for any hours above their contracted number.

(a) Write a class definition for `Employee`, incorporating all of the attributes mentioned above. Be sure to use appropriate data types and access modifiers. You should also define a public constructor for `Employee`, which should initialise all fields – the initial value for "hours actually worked" should be zero. The constructor should check that the parameters are sensible, and should throw an `IllegalArgumentException` if it is given bad input. You do not need to define any instance methods. [7]

Solution:

```
// 1 mark for class signature
public class Employee {

  // 1.5 marks for correct data types
  // 0.5 marks for making them all private (or protected)
  private String familyName;
  private String givenName;
  private int employeeNum;

  private double hourlyRate;
  private int contractedHours;
  private int actualHours;

  // 1 mark for basic constructor signature
  // 0.5 mark for including throws declaration
  public Employee(String familyName, String givenName, int
     employeeNum, double hourlyRate, int contractedHours) throws
     IllegalArgumentException
  {
    // 1 mark for correct field assignments
    this.familyName = familyName;
    this.givenName = givenName;
    this.employeeNum = employeeNum;

    // .5 mark for this assignment
    this.contractedHours = 0;

    // .5 mark for this test
    if (hourlyRate <= 0) {
```

```
      throw new IllegalArgumentException("Invalid hourly rate: "
          + hourlyRate);
    }
    this.hourlyRate = hourlyRate;

    // .5 mark for this test
    if (contractedHours < 0) {
      throw new IllegalArgumentException("Invalid contracted
          hours: " + contractedHours);
    }
    this.contractedHours = contractedHours;
  }

}
```

(b) Write a method `computePay()` that computes and returns the employee's pay for the current month based on the current values of the appropriate fields. [4]

Solution:

```
// 1 mark for correct method signature
public double computePay() {
   // 1 mark for correctly determining extra hours
   // NB: Math.max() is not necessary;
   // any other correct computation is valid
   double extraHours = Math.max(actualHours - contractedHours, 0);

   // 1 mark for correctly determining regular hours
   double regularHours = Math.min(contractedHours, actualHours);

   // 1 mark for correctly computing the result
   return (regularHours * hourlyRate) + (1.5 * extraHours *
      hourlyRate);
}
```

(c) The company has decided to start dividing employees into two categories: **salaried** employees, who receive the same pay every month regardless of the number of hours worked, and **hourly** employees, whose pay is computed as above. Describe how you would modify your class design above to deal with this situation, including any possible new classes that might be introduced: you may illustrate your answer with fragments of Java code but this is not required. [4]

Solution: The students should mention the following points (1 mark each):

- The `Employee` class should be made abstract, with subclasses `SalariedEmployee` and `HourlyEmployee`

- The `computePay()` method should be made abstract in `Employee`
- The employee name and ID should stay in the `Employee` class and be made **`protected`**, while the other fields should move to `HourlyEmployee`
- The `SalariedEmployee` class should have a single field representing the monthly salary which will be used in `computePay()`

If they propose modifying the class to have a flag representing salaried/hourly, award only up to 2 marks depending on the other details.

END OF QUESTION PAPER