University of Glasgow

Friday 13 December 2019
4:30pm – 5:30pm
(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) Java is a **statically typed** language. Define what is meant by statically typed, and describe the difference between **implicit type conversion** and **explicit type conversion**, giving an example of each. [5]

> **Solution:** Statically typed means that the type of all variables must be specified at declaration time. (1 mark)
>
> Implicit conversion is when the compiler is able to automatically convert between data types, and generally is possibly in situations where the conversion does not lose information (1 mark). Valid examples include converting from **`int`** to **`double`** or similar, or from an instance of a subclass to the superclass (i.e., polymorphism) (1 mark).
>
> Explicit conversion occurs where information has the potential to be lost, so the conversion cannot happen automatically (1 mark). Valid examples include casting from **`double`** to **`int`** or between classes, or using methods such as `Integer.parseInt()` to convert between types (1 mark).

(b) How do constructors differ from normal methods when it comes to inheritance? How is the superclass constructor invoked from a subclass? What happens automatically if you do not call the superclass constructor, and what problems can this cause? [4]

> **Solution:** Constructors are not members, so are not inherited by subclasses. (1 mark) A superclass constructor can be invoked with the **`super`** keyword from a subclass constructor (1 mark). If you do not explicitly invoke the superclass constructor, a call is inserted to the no-args constructor (1 mark). If the superclass does not have a no-args constructor, you get a compile-time error (1 mark).

(c) Name and describe two classes or interfaces involved in writing multi-threaded code in Java. [4]

> **Solution:** The `Thread` class represents a thread of execution and provides methods such as `start()`, `stop()`, and `join()`.
>
> The `Runnable` interface is the main way of defining the behaviour of a `Thread`, by implementing `Runnable` and defining a `run()` method.
>
> (1 point for naming each class, 1 point for an accurate description of each. Also accept other classes like `Lock` or concurrent collections if description is accurate.)

(d) Describe the steps involved in creating GUI elements in Swing and defining the behaviour associated with them. [3]

> **Solution:** Three main steps (1 mark for mentioning each):
>
> - Create on-screen GUI elements
> - Create listener object and define callback behaviour

- Associate listener object with GUI elements

(e) What is the main distinguishing feature of an **immutable** class in Java? Give an example of a commonly used immutable class. List two advantages of using an immutable class. [4]

Solution: Main distinguishing feature: object state cannot change after it is constructed (1 mark). Commonly used classes: `String`, the primitive wrapper classes such as `Integer` (1 mark). Advantages: can be safely shared across threads (1 mark), can be used for lookup in dictionary-type structures (1 mark). (Also accept other advantages such as efficiency of storage.)

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

(a) This question involves information about phone calls made from a particular mobile phone over one calendar month. Details of calls are stored in a list, in which each element contains the number called and the duration of the call in minutes, represented as an instance of the `Call` class below:

```
public class Call {
  public String number;
  public int duration;
}
```

An example of a call list would be as follows:

[[“01234567890”, 5], [“09876543201”, 17], [“01234567890”, 14], . . .]

The following function is supposed to calculate the phone bill for the month, based on the following charging scheme.

- Each call is charged at a rate of 2p per minute.
- The favourite phone number has a maximum total charge of 100p; in other words, after 50 minutes of calls have been made to the favourite number, subsequent minutes are free.

The parameter `calls` is a list of `Call` objects in the format described above, and the parameter `favourite` is a string representing the favourite phone number.

The code below contains two errors. Give a clear explanation of each error (including how it would affect the execution of the function) and say how to correct it. You may illustrate your answer with fragments of Java code, but this is not required. [6]

```
1  public int calculateBill (List<Call> calls, String favourite) {
2    Map<String, Integer> totals = new HashMap<>();
3    for (Call call : calls) {
4      totals.put(call.number, totals.get(call.number) + call.duration);
5    }
6
7    int totalCharge = 0;
8    for (String number : totals.keySet()) {
9      if (number.equals(favourite)) {
10       totalCharge += 100;
11     } else {
12       totalCharge += totals.get(number) * 2;
13     }
14   }
15
16   return totalCharge;
17 }
```

Solution: Error 1: in the loop at lines 3–5, it does not check whether the number has already been seen before adding to the HashMap (1 mark). This will result in a NullPointerException at run-time the first time through the loop (1 mark). The fix

is to add a check using `totals.containsKey()` and to add calls to unseen phone numbers to the map directly (1 mark).

Error 2: the if statement at lines 9–10 does not work correctly: it should check the total number of minutes made to the favourite number and should only charge the actual amount if it is less than 50 (1 mark). The result will be that the bill will be too high for anyone who called their favourite number for less than 50 minutes (1 mark). The fix is to use `Math.min()` or similar to compute the correct value (1 mark).

(b) For each of the following Java code fragments, indicate **exactly** what will happen when it is 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(s) are. Assume that all necessary classes have been imported.

(i)
```
double d = 5;
int i = 2;
System.out.println(d / i);
```

[1]

Solution:
2.5

(ii)
```
String s1 = "hello";
String s2 = new String("hello");
System.out.println(s1 == s2);
System.out.println(s1.equals(s2));
```

[1]

Solution:
false
true

(iii)
```
List<String> days = Arrays.asList ("MONDAY", "TUESDAY",
    "WEDNESDAY", "THURSDAY", "FRIDAY", "SATURDAY", "SUNDAY");

System.out.println(days.stream()
  .filter(s -> s.contains("T"))
  .count());
```

[1]

Solution:
3

(iv)
```
for (int i = 0; i < 10; i++) {
   System.out.println(i--);
}
```

[1]

Solution:
It prints `0` in an infinite loop until stopped

(v)
```
Scanner s = new Scanner(System.in);
while (true) {
```

```
   System.out.print("> ");
   try {
      int i = s.nextInt();
      System.out.println("You entered: " + i);
      break;
   } catch (Exception e) {
      s.next();
   }
}
```

[1]

Solution: It continues to prompt the user with a `>` until they enter a valid integer value, and then prints out `You entered:` followed by that value.

(vi)
```
ArrayList<String> l = new List<>();
l.add("one");
l.add("two");
Collections.reverse(l);
System.out.println(l);
```

[1]

Solution: Two problems here, both of which prevent compilation: `List` is not a subclass of `ArrayList`, and also `List` is abstract and cannot be instantiated. 0.5 points for mentioning each issue.

(vii)
```
// File A.java
public abstract final class A {
   public String toString() {
      return "Hello";
   }
}

// main method
System.out.println(new A());
```

[1]

Solution: This will not compile as a class cannot be both abstract and final (and also an abstract class cannot be instantiated).

(viii)
```
// File B.java
public class B {
   public static int i;
   public int j;

   public B(int k) {
      i = k;
      j = k;
   }
}

// main method
B b1 = new B(4);
B b2 = new B(-3);
System.out.println(b1.i + " " + b2.i + " " + b1.j + " " + b2.j);
```

[1]

Solution:
`-3 -3 4 -3`

(ix)
```
// File C.java
public class C {
   public String s;
   public C(String s) {
      s = s;
   }
}

// main method
C c = new C("five");
System.out.println(c.s);
```

[1]

Solution:
null

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

First, read the following description of a bicycle sharing system.

> You are to design a set of classes to model a simplified bike-share system similar to Glasgow's NextBike system or to the Santander Cycles in London.
>
> Each **bicycle** has the following properties: an identifier (a positive integer), a bicycle type (a string), as well as a Boolean flag indicating whether the bicycle is available for rental.
>
> A **customer** can rent only one bicycle at a time: when a bicycle is returned, the total cost of that rental is computed by multiplying the rental time by the base rental rate, which is currently £2 per hour. If a customer attempts to rent a second bicycle when they already have one rented, or to rent a bicycle that is already rented to another customer, an error is returned and the bicycle is not rented.

(a) Write a full class definition for `Bicycle` following the specification above. Be sure to use appropriate data types and access modifiers. Include a constructor that initialises all fields to appropriate values. The initial value for **available** should be **`true`**. Also implement a getter for all fields, and a setter method for the **available** flag. [5]

Solution:

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

  // 0.5 mark for correct data types
  // 0.5 marks for making them all private (or protected)
  private int id;
  private String bikeDetails;
  private boolean available;

  // 0.5 mark for constructor signature
  public Bicycle(int id, String bikeDetails) {
    // 0.5 marks for these field assignments
    this.id = id;
    this.bikeDetails = bikeDetails;

    // 0.5 marks for this assignment (could also be at
       declaration time)
    this.available = true;
  }

  // 1 mark for correct getters and setters
  // 0.5 mark for correct access modifiers
  public boolean isAvailable() {
    return this.available;
  }
```

```
  public int getId() {
    return this.id;
  }

  public String getDetails() {
    return this.bikeDetails;
  }

  public void setAvailable(boolean available) {
    this.available = available;
  }
}
```

(b) Write a class definition for the `Customer` class. Your class definition should include implementations of the following two public methods:

- **`void`** `rentBike(Bicycle bike)` – rents the given bike to the customer, or else throws an `IllegalArgumentException` if one of the above error conditions is encountered (i.e., customer already has a bike rental, or specified bike is unavailable).
- **`double`** `endRental()` – ends the rental of the current bike and returns the total cost of the rental. If the customer is not currently renting a bike, this method should instead throw an `IllegalArgumentException`.

As part of your answer, you may want to make use of the `java.time.Instant` class, which represents a single instantaneous point in time. You can obtain an `Instant` object corresponding to the current time as follows:

```
Instant currentTime = Instant.now();
```

You can also compute the difference in hours between two `Instant` objects as follows (NB: this is a slight simplification to the real behavour of the `Instant` class):

```
long difference = Duration.between(instant1, instant2).toHours();
```

Your `Customer` class only needs to include fields that are required to support the above behaviour, and only the above methods are required. There is no need to add extra fields or to write constructors, getters, or setters unless they are required as part of your implementation.

[7]

Solution:

```
// Import statements are not required
import java.time.Duration;
import java.time.Instant;

// 0.5 marks for correct class header
public class Customer {

  // 1 mark for correct (plausible) data types and correct access
     modifiers
  private Bicycle rentedBike;
  private Instant rentalTime;

  // 0.5 marks for correct header
  public void rentBike(Bicycle bike) {
    // 0.5 marks for this check
    if (rentedBike != null) {
      throw new IllegalArgumentException("You already have a
         rented bike!");
    }
    // 0.5 marks for this check
    if (!bike.isAvailable()) {
      throw new IllegalArgumentException("Bike is not available
         for rental!");
    }
    // 0.5 marks for each of the actions below (1.5 total)
    this.rentedBike = bike;
    bike.setAvailable(false);
    this.rentalTime = Instant.now();
  }

  // 0.5 marks for correct header
  public double endRental() {
    // 0.5 marks for this check
    if (rentedBike == null) {
      throw new IllegalArgumentException("No current rental to
         end!");
    }

    // 0.5 marks for each of the actions below (1.5 total)
    rentedBike.setAvailable(true);
    rentedBike = null;
    return 2 * Duration.between(rentalTime,
       Instant.now()).toHours();
  }
}
```

(c) The bike provider now wants to allow customers to rent more than one bike at a time. How would you modify your `Customer` class to meet this requirement? You may illustrate your answer with fragments of Java code, but it is not required. [3]

Solution: The student should mention at least the following points (1 mark each):

- The internal representation of rented bikes will need to change.
- The signature of `endRental()` will need to change.
- The behaviour of both `Customer` methods will need to change – in particular, the error conditions are now different than before.