University of Glasgow

Sample online exam
Expected Duration: 60 minutes
Time allowed: 2 hours

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

JAVA PROGRAMMING 2
COMPSCI 2001

(Answer all 3 questions)

This examination paper is worth a total of 50 marks.

THIS IS A SAMPLE PAPER: It gives examples of the question types that can be expected on an online, open-book exam

Important note: throughout this exam, whenever you are asked to write Java code, do not worry about whether your code compiles. The markers will never test any submitted code from this exam.

1. This question asks you to understand and discuss Java programming concepts (10 marks total)

(a) Describe the difference between **implicit type conversion** and **explicit type conversion**, giving an example of each. [4]

(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]

(c) List two advantages of using an **immutable** class in Java. [2]

2. This question asks you to read and understand Java code. (20 marks total)

(a) The `Call` class is defined as follows, which is designed to store a phone call made to a given number, with a duration in minutes.

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

The following Java method makes use of the above class and is designed to compute a total phone bill (in pence) based on a list of calls. Study this method and then answer the questions that follow.

```
1  public int calculateBill (List<Call> calls, String favourite) {
2    Map<String, Integer> totals = new HashMap<>();
3    for (Call call : calls) {
4      if (!totals.containsKey(call.number)) {
5        totals.put(call.number, 0);
6      }
7      totals.put(call.number, totals.get(call.number) + call.duration);
8    }
9
10   int totalCharge = 0;
11   for (String number : totals.keySet()) {
12     if (number.equals(favourite)) {
13       totalCharge += Math.min(100, totals.get(number) * 2);
14     } else {
15       totalCharge += totals.get(number) * 2;
16     }
17   }
18
19   return totalCharge;
20 }
```

(i) Explain how a total phone bill is computed based on a list of calls. Clearly explain the role of the `favourite` parameter in this computation. [3]

(ii) Line 1 contains a parameter of type `List<Call>`. What is the role of `<Call>` in this type? What would happen if `<Call>` were not included? [3]

(iii) In line 2, a variable is declared and initalised. Clearly explain the role of the left side of the = sign and the role of the right side. What relationship must hold between the types on the left and the right for this sort of assignment to be valid? [3]

(iv) What does the **`if`** statement check for at line 4, and what does the body of the **`if`** statement (line 5) do? What potential problem could arise if that check were not there? [3]

(v) In Line 12, two values are compared with `.equals()`. Explain what this comparison does, and discuss why this was the correct choice, rather than using == for comparison. [3]

(b) Consider the following code:

```
1  // File B.java
2  public class B {
3    public int i;
4    public static int j;
5    public B(int k) {
6      i = k;
7      j = k;
8    }
9  }
10
11 // main method
12 B b1 = new B(4);
13 B b2 = new B(-3);
```

(i) After line 12 is executed, the value of `b1.i` and `b1.j` are both 4. Clearly explain why. [1]

(ii) After line 13 is executed, the value of `b1.i` is 4, while `b1.j` is 3. Clearly explain why. [2]

(iii) What is an alternative, preferred way to refer to `b1.j`, and why? [2]

3. This question concerns Java class design. (20 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]

(b) Write a line of code declaring and initialising a new `Bicycle` object with ID 5 and bike details "Details". [2]

(c) 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]

(d) Assume that `c` is a variable of type `Customer`, and `b` is a variable of type `Bicycle`. Show the Java code that can be used to rent bicycle `b` to customer `c`, and to print an error message if there is a problem. [3]

(e) 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]

END OF QUESTION PAPER