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 along with sample solutions.

The solutions here are generally more detailed than I would expect you to produce under exam conditions. As long as you include the same points as in the solutions, you would get full marks.

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]

Solution: Implicit conversion is when the compiler is able to automatically convert between data types, and generally is possible 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 (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, your code will not compile (1 mark).

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

Solution: 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 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]

> **Solution:** Overall, the phone bill is computed by charging 2p for each minute of calls, across all of the calls in the list. (1 mark) The `favourite` parameter appears to represent a "favourite" phone number where there is an upper limit on the cost of calls to that number. (1 mark) Specifically, only the first 50 minutes of calls to that number are charged. (1 mark)

(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?

> **Solution:** The type in <> represents the generic parameter for the list – that is, the type `List<Call>` represents a list of `Call`s (1 mark). If the types were omitted, the code would still compile (1 mark), but the type-safe properties

provided by the use of generics would not be available (specifically, the **for** loop starting at line 3 would need to be rewritten). (1 mark)

(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]

Solution: The left-hand side declares the type of the variable – in this case, it is the interface type `Map` (1 mark). The right-hand side is the specific value assigned to the variable, in this case an object of type `HashMap`. (1 mark). This line is valid because the type on the right-hand side is a more specific version of the type on the left-hand side (in this case it is an interface relationship, but this would also work with a subclass) (1 mark).

(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]

Solution: This statement checks whether the call number has already been seen (1 mark), and makes sure to add the number to the map if it isn't there already. (1 mark) If that check were not there, the method would throw a `NullPointerException` the first time through the loop, at line 7. (1 mark)

(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]

Solution: Comparing with `.equals()` compares the objects for equality using the class-specific, overridden version of the method (1 mark). Comparing with == compares the object locations in memory (1 mark). In this case, the important fact is whether the objects are equal, not identical, so `.equals()` is the correct choice (1 mark).

(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]

> **Solution:** The constructor of the `B` class sets both fields to the given parameter, which in this case is 4 (1 mark).

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

> **Solution:** The second call to the constructor does not change the value of `b1.i`, because it is an instance field (1 mark). But when the second call changes the value of `b2.j`, it also changes the value of `b1.j` because the field is static and therefore shared between all instances of the class.

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

> **Solution:** A better way to refer to it is `B.j` (1 mark), since it is a static field and static fields are associated with the class, not with objects of the class (1 mark).

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]

Solution: PROPER JAVA SYNTAX IS NOT REQUIRED HERE

```
// 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 line of code declaring and initialising a new `Bicycle` object with ID 5 and bike details "Details".

Solution: PROPER JAVA SYNTAX IS NOT REQUIRED HERE

```
Bicycle b = new Bicycle(5, "Details");
```

(1 mark for correct declaration, 1 mark for correct use of constructor)

(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]

Solution: PROPER JAVA SYNTAX IS NOT REQUIRED HERE

```
// 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();
  }
}
```

(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]

Solution: PROPER JAVA SYNTAX IS NOT REQUIRED HERE

```
// 1 mark for using try-catch syntax
try {
   // 1 mark for correct method call
   c.rentBike(b);
} catch (IllegalArgumentException ex) {
   // 1 mark for doing something sensible with the exception
   System.out.println("Error renting bike: " + ex.getMessage());
}
```

(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]

Solution: Answer 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.