Programming Methods & Java Examples

Dr Robert Harle, 2009

The following questions may be useful to work through for supervisions and/or revision. They are separated broadly by handout, but borders are necessarily blurred. Since there is no Java lecture course, I have included some Java questions that pick up on points in the ticks. There are no past exam questions for the "Programming Methods & Java" question on CST Paper 1 so the course website will also feature some sample tripos-esque questions for you to download and try.

Some of these questions are more involved than others - roughly speaking they start quick and easy and finish tougher in each section. An asterix by any question or part of a question indicates that the answer may need more than you've been explicitly taught. Consult your supervisor for a sensible subset to attempt.

Handout 1: Bouncing Ball

1. Imagine you have been given software that predicts the weather next year by simulating all the physical processes that the software authors felt relevant. The software predicts that it will rain next week. What could you do to decide whether or not to trust the simulation?

2. A student wishes to have an equivalent class to Vector2D for a 3D vector. She argues that because a 3D vector is a 2D vector with a bit extra (z), it should derive from Vector2D. Comment on this.

3. In Vector2D.java, the member variables are all private. This means that any derived classes cannot reference them directly. Is this a problem?

4. Explain what a marker interface is and find an example in Java that is not the interface Cloneable.

5. A student forgets to use super.clone() in their clone() method:

```
public class SomeClass extends SomeOtherClass implements Cloneable {
   private int[] mData;

   ...
   public Object clone() {
      SomeClass sc = new SomeClass();
      sc.mData = mData.clone();
   }

}
```

What could go wrong?

6. Vector2D provides methods to add two vectors. Contrast the following approaches to providing an add method assuming Vector2D is (i) mutable, and (ii) immutable.

(a) `public void add(Vector2D v)`

(b) `public Vector2D add(Vector2D v)`

(c) `public Vector2D add(Vector2D v1, Vector2D v2)`

(d) `public static Vector2D add(Vector2D v1, Vector2D v2)`

7. A student proposes to modify the bouncing ball simulation to use a time step that can vary in length dynamically. They argue that this will allow the simulation to continue rather than die abruptly when the bounces become too small for the current time step. Comment on this approach.

8. An alternative strategy to clone()-ing an object is to provide a *copy constructor*. This is a constructor that takes the enclosing class as an argument and copies everything manually:

```
public class MyClass {
  private String mName;
  private int[] mData;

  // Copy constructor
  public MyClass(MyClass toCopy) {
    this.mName = toCopy.mName;
    // TODO
  }
  ...
}
```

(a) Complete the copy constructor.

(b) Make MyClass clone()-able (you should do a deep copy).

(c) * Why might the Java designers have disliked copy constructors? [Hint: What happens if you want to copy an object that is being referenced using its parent type?].

(d) * Under what circumstances is a copy constructor a good solution?

9. Consider the class below. What difficulty is there in providing a deep clone() method for it?

```
public class CloneTest {
  private final int[] mData = new int[100];
}
```

10. (*) Rewrite the basic bouncing ball simulator (where the floor is static) to work with a-priori event detection.

11. Modify the bouncing ball simulation so that the ball is given some horizontal velocity intially. With each bounce the horizontal speed should be reduced in a similar fashion to the vertical speed.

Handout 2: Design Patterns

12. Explain the difference between the two types of polymorphism in programming.

13. Imagine you have two abstract classes: Employee (which implements everything associated with being an employee) and Ninja (which implements everything needed to be a ninja). If you have employees who are also ninjas, you would presumably want to write:

```
public class NinjaEmployee extends Ninja, Employee {
   ...
}
```

This is called multiple inheritance. Suggest why it isn't supported in Java (Hint: multiple inheritance of interfaces *is* supported so what's the difference?)

14. Suppose you have an abstract class TeachingStaff with two concrete subclasses: Lecturer and Professor. Problems arise when a lecturer gets promoted because we cannot convert a Lecturer object to a Professor object. Using the State pattern, show how you would redesign the classes to permit promotion.

15. Describe how Java I/O makes use of the Decorator pattern (e.g. Look at java.io.Reader and java.io.InputStreamReader).

16. A drawing program has an abstract Shape class. Each Shape object supports a draw() method that draws the relevant shape on the screen. There are a series of concrete subclasses of Shape, including Circle and Rectangle. The drawing program keeps a list of all shapes in a List<Shape> object.

(a) Should draw() be an abstract method?

(b) Write Java code for the function in the main application that draws all the shapes on each screen refresh.

(c) Show how to use the Composite pattern to allow sets of shapes to be grouped together and treated a a single entity.

(d) Which design pattern would you use if you wanted to extend the program to draw frames around some of the shapes?

17. In the Java GUI Libraries, each component has a LayoutManager that determines how any child components it has will be laid out on screen. For example, GridLayout is a concrete LayoutManager that sets things out in a uniformly-sized grid, whilst BorderLayout is also a concrete LayoutManager that lays out one main component and four others along its sides. Identify the design pattern in use here.

18. Explain using diagrams how to the Abstract Factory pattern would help in writing an application that must support different languages (english, french, german, etc).

19. One technique to break a Singleton object is to extend it and implement Cloneable. This allows Singleton objects can be cloned, breaking the fundamental goal of a Singleton! Write a Java Singleton class that is not final but still prevents subclasses from being cloned.

20. Assume there is a machine somewhere on the internet that can supply the latest stock price for a given stock. The software it runs is written in Java and implements the interface:

```
public interface StockReporter {
  public double getStockPrice(String stockid);
}
```

You are given a Java class MyStockReporter that implements this interface for you. When you use a MyStockReporter object, your request is passed onto the real machine.

(a) Identify the design pattern

(b) Why is this design inefficient?

(c) Draw a UML class diagram *and* a sequence diagram to explain how the Observer pattern could improve efficiency. Give the changes to the interface that would be required.

Handout 3: Photo Organiser

21. Why are interfaces useful in decoupling classes?

22. Explain why it is rarely possible to fully decouple the View from the Controller in the MVC pattern.

23. Describe how Java swing makes use of the MVC, Composite and Observer patterns.

24. The Photo class in the lectures does not behave sensibly if we provide a bad filename for the photo. Alter the program so that when a bad filename is given, a message will alert the user to the mistake rather than continuing regardless.

25. * Write a view component for the Photo Organiser that lists all of the photos in the organiser by filename, sorted alphabetically.

Handout 4: Web Server

26. The sendHeader() method in HTTPConnection.java uses a boolean return value to indicate success or failure. Discuss the advantages and disadvantages of using exceptions instead.

27. Explain the notion of a thread. Why do threads make programming harder?

28. HTTPConnection needs to read in a requested file from the disk and send it out to the browser. The code supplied does so byte-by-byte:

```
DataOutputStream output =
   new DataOutputStream(mSocket.getOutputStream());
while (true) {
   int b = requestedfile.read();
   if (b == -1) {
      break; //end of file
   }
   output.write(b);
}
```

It is also possible to read chunks of data in one go using the read(byte[] b) method of InputStream. Investigate the use of this method. Test your code to determine whether reading and sending chunks is more efficient than doing so byte-by-byte.

29. * Look again at the code in the previous question

(a) Why does a Java InputStream not throw an exception for the end of file, instead of relying on a return value of -1?

(b) Why does read() return an int when it only holds a byte (0–255)?

30. Write a simple program that creates two threads. Each thread should loop forever, printing out "Thread X" where X is either 1 or 2, depending on the thread which wrote it. What do you notice about the order of printing when you run it?

31. * Write a client that connects to the WebServer in lectures and acts like a browser requesting an HTML file.

32. * Write a server/client pair to handle a multi-user chat. Clients should connect to the server, and anything typed locally should be relayed to the server, which should use the observer pattern to relay it to all other connected clients.

Handout 5: Simple Computer Vision

33. Why are the member variables in Pixel.java declared final?

34. Explain what caused the StackOverflowException seen in the lectures. Write a toy program that results in a StackOverflowException.

35. The user of the class Car below wishes to maintain a collection of Car objects such that they can be iterated over in some specific order.

```
public class Car {
   private String manufacturer;
   private int age;
}
```

(a) Show how to keep the collection sorted alphabetically by the manufacturer without writing a Comparator.

(b) Show how to keep the collection sorted by {manufacturer, age}. i.e. sort first by manufacturer, and sub-sort by age.