OOP Exercise Sheet 2010/11

Dr Robert Harle

These exercises follow the notes and are intended to provide material for supervisions. There are four types of question, indicated by a letter in brackets after the question number:

(U) Straightforward question to test understanding.

(P) Programming practice question. May take more time to complete and not likely to appear in a tripos exam.

(T) Questions in a tripos exam style that could form a significant part of a full tripos question.

There are probably too many exercises here for the small number of supervisions likely to be allocated to the course. Supervisors may wish to set appropriate subsets. At a minimum, I suggest everyone should attempt the **(U)** questions and most of the **(T)** questions. There are also some sample tripos questions on the course website.

Mapping Code to Hardware

Q1. **(U)** How much system memory can we use if we have 8, 32 or 64 bit registers?

Q2. **(U)** Why do you think CPU manufacturers haven't all agreed on a single set of instructions?

Q3. **(U)** Apple recently started using Intel processors that support x86 instructions. This means Apple machines can now run Microsoft Windows. However, off-the-shelf PC software (which is compiled for x86) does not run on a Mac that is using the Apple operating system compiled for the Intel processor. Why not?

Pointers and References

Q4. **(U)** Pointers are problematic because they might not point to anything useful. A `null` reference doesn't point to anything useful. So what is the advantage of using references over pointers?

Q5. **(U)** Draw box diagrams to illustrate what happens in each step of the following Java code in memory:

```
Person p = null;
Person p2 = new Person();
p = p2;
p2 = new Person();
p=null;
```

Q6. **(T)** A programmer proposes a new imperative language whereby all variables are passed by reference (even those of primitive type). Discuss the advantages and disadvantages of this design.

Q7. **(T)** The following code is a failed attempt to write a function that doubles the value of an `int`. Make the code work *without changing the return type of the function.*

```
public void double(int x) {
  x=2*x;
}

int a = 7;
double(a);
```

Q8. **(P)** The notes mentions the confusion over passing by value and reference. Write Java code to demonstrates that the variable declared by the code `int[] test` can be viewed as a reference that gets copied when passed as an argument.

OOP Concepts

Q9. **(U)** Identify the primitives, references, classes and objects in the following Java code:

```
double d = 5.0;
int i[] = {1,2,3,4};
List l = new List();
```

```
Double k = new Double();
Tree t;
float f;
Computer c = null;
```

Q10. **(U)** A book is composed of a cover, a table of contents, and a number of pages, each of which can contain text or diagrams. Draw a simple UML diagram to represent this information.

Encapsulation

Q11. **(T)** Consider the following snippets of code:

```
public class Wheel {
  public float radius;
};

public class Bike {
  public Wheel[] wheels = null;

  public create(float wheelRadius) {
    wheels = new Wheel[2];
    wheels[0].rimSize=wheelRadius;
    wheels[1].rimSize=wheelRadius;
  }

  public float getSpeed(float pedalFrequency) {
    return 2.0*Math.PI*wheels[0].radius*pedalFrequency;
  }
};
```

(a) Why are these two classes said to be tightly coupled? Give three examples of problems that this tight coupling might cause.
(b) Rewrite the classes to loosen the coupling and address the concerns you highlighted in your examples

Q12. **(T)** The Vector2D class developed in lectures provides a method to add two Vector2D objects. Contrast the following approaches[1] to providing such 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)`

Inheritance

Q13. **(U)** A student wishes to create a class for a 3D vector and chooses to derive from the Vector2D class (i.e. public void Vector3D extends Vector2D). The argument is that a 3D vector is a "2D vector with some stuff added". Explain the conceptual misunderstanding here.

Q14. **(U)** Suggest UML class diagrams that could be used to represent the following:

(a) A shop is composed of a series of departments, each with its own manager. There is also a store manager and many shop assistants. Each item sold has a price and a tax rate.
(b) Vehicles are either motor-driven (cars, trucks, motorbikes) or human-powered (bikes, skateboards). All cars have 3 or 4 wheels and all bikes have two wheels. Every vehicle has an owner and a tax disc

Q15. **(U)** Consider the Java class below:

```
package questions;

public class X {
  MODIFIER int value = 3;
};
```

[1] Workbook 3 discusses the static keyword

Another class Y attempts to access the field value in an object of type X. Describe what happens at compilation and/or runtime for the range of MODIFIER possibilities (i.e. public, protected, private and unspecified) under the following circumstances:

(a) Y subclasses X and is in the same package;
(b) Y subclasses X and is in a different package;
(c) Y does not subclass X and is in the same package;
(d) Y does not subclass X and is in a different package.

Q16. **(T)** Explain what is meant by (dynamic) polymorphism in OOP and explain why it is useful, illustrating your answer with an example.

Q17. **(P)** A CS department keeps tracks of its CS students using some custom software. Each student is represented by a Student object that features a pass() method that returns true iff the student has all seven ticks to pass the year. The department suddenly starts teaching NS students, who only need five ticks to pass. Using inheritance and polymorphism, show how the software can continue to keep all Student objects in one list in code without having to change any classes other than Student.

Q18. **(U)** Explain the differences between a class, an abstract class and an interface in Java.

Q19. **(U)** A programming language designer proposes adding 'selective inheritance' whereby a programmer manually specifies which methods or fields are inherited by any subclasses. Comment on this idea.

Q20. **(T)** Imagine you have two classes: Employee (which represents being an employee) and Ninja (which represents being a Ninja). An Employee has both state and behaviour; a Ninja has only behaviour.

You need to represent an employee who is also a ninja in Java (a common problem in the real world). By creating only one interface and only one class(NinjaEmployee), show how you can do this without having to copy method implementation code from either of the original classes.

Lifecycle of an Object

Q21. **(U)** Write a small Java program that demonstrates constructor chaining using a hierarchy of three classes as follows: A is the parent of B is the parent of C. Modify your definition of A so that it has exactly one constructor that takes an argument, and show how B and/or C must be changed to work with it.

Q22. **(T)** 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?

Q23. **(T)** 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?

Q24. **(T)** 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];
}
```

Java Class Libraries

Q25. **(U)** Using the Java API or otherwise, compare the Java classes Vector, LinkedList, ArrayList and TreeList.

Q26. **(U)** The problem described for the slide "Generics and SubTyping" can be solved in Java generics through the use of *wildcards*. After researching wildcards, summarise how they solve the problem.

Q27. **(P)** Write a Java class that can store a series of student names and their corresponding marks (percentages) for the year. Your class should use at least one Map and should be able to output a List of all students (sorted alphabetically); a List containing the names of the top P% of the year as well; and the median mark.

Object Comparison

Q28. **(U)** Write a class that represents a 3D point (x,y,z). Give it a natural order such that values are sorted in ascending order by z, then y, then x.

Q29. **(T)** 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.

I/O

Q30. **(P)** Write a Java program that reads in a text file that contains a single floating point number on each line and prints the mean and standard deviation of all the values.

Q31. **(P)** Write a Java program that reads in a text file that contains two integers on each line, separated by a comma (i.e. two columns in a comma-separated file). Your program should print out the same set of numbers, but sorted by the first column and subsorted by the second.

Design Patterns

Q32. **(T)** 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.

Q33. **(T)** A drawing program has an abstract Shape class. Each Shape object supports a draw() method that draws the relevant shape on the screen (as per the example in lectures). 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 as 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? Show how this would work.

Q34. **(T)** One technique to break a Singleton object is to extend it and implement Cloneable. This allows Singleton objects to 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.

Q35. **(T)** 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 automatically passed onto the real machine.

(a) Identify the design pattern in use here
(b) Why is this design inefficient?
(c) Draw a UML class diagram to explain how the Observer pattern could improve efficiency. Give the changes to the interface that would be required.

Q36. **(U)** Explain using diagrams how to the Abstract Factory pattern would help in writing an application that must support different languages (english, french, german, etc).