OOP Exercise Sheet 2014/15

Dr Robert Harle

These exercises follow the notes and are intended to provide material for supervisions. For the majority of students this course has two challenges: the first is understanding the core OOP concepts; the second is applying them correctly in Java. The former can be addressed through traditional academic study techniques; the latter requires a combination of knowledge and experience—i.e. *practice*. To get the most out of this course you will need to put in effort programming *from scratch*. Many of these questions prod you to do so!

They are graded A–C to indicate the effort expected (An A would be worth a few marks, a C worth much more). Any of the questions could be a component of a Tripos exam question: the rating is indicative of how substantial that component might be. Your supervisor may not wish you to try all questions—please check with them. Questions with an asterisk are meant to stretch students finding the course straightforward. For specific exam practice, there are also some sample tripos questions on the course website.

Types, Objects and Classes

1. (A) Give three differences between a typical functional and a typical imperative programming language.

2. (A) Demonstrate the use of all of Java's control flow statements (as given in the lectures) by writing the following functions:

- int sum(int[] a), which produces the sum of the elements in a.
- int[] cumsum(int[] a), which produces the cumulative sum of a e.g. [1,2,3] becomes [1,3,6].
- int[] positives(int[] a) which produces the array of positives in a (note your returned array should not have any empty slots).

3. (A) Identify the primitives, references, classes and objects in the following Java code:

```
double d = 5.0;
int i[] = {1,2,3,4};
LinkedList<Double> l = new LinkedList<Double>();
Double k = new Double();
Tree t;
float f;
Computer c = null;
```

4. (B)

(a) Write a Java function that uses an array-of-arrays to represent an n×n unit matrix of floats.
(b) Write another Java function that transposes an n×n matrix represented using an array-of-arrays. Your function should be *in-place* i.e. use only $O(1)$ space.

5. (A*) Write Java code to test whether your Java environment performs tail-recursion optimisations or not.

Pointers, References and Memory

6. (A) 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?

7. (A) Draw some simple diagrams to illustrate what happens with each step of the following Java code in memory:

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

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

8. (B) The following code is a failed attempt to write a function that can be used to add increments to a 2D vector. Explain why it doesn't work and adapt the code to work (without using a custom class for those that know how to do this).

```
public void vectorAdd(int x,int y,int dx, int dy) {
  x=x+dx;
  y=y+dy;
}

public static void main(String[] args) {
  int a=0;
  int b=2;
  vectorAdd(a,b,1,1);
  System.out.println(a+'' ''+b);
  // (a,b) is still (0,2)
}
```

9. (B) The notes mention 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.

10. (C) 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.

Creating Classes

11. (B)

(a) Develop a mutable class to embody the notion of a 2D vector based on floats (do not use Generics). At a minimum your class should support addition of two vectors; scalar product; normalisation and magnitude.

(b) What changes would you be needed to make it immutable?

(c) Contrast the following approaches to providing the addition method for both the (i) mutable, and (ii) immutable versions.

- `public void add(Vector2D v)`
- `public Vector2D add(Vector2D v)`
- `public Vector2D add(Vector2D v1, Vector2D v2)`
- `public static Vector2D add(Vector2D v1, Vector2D v2)`

(d) How can you convey to a user of your class that it is immutable?

12. (B) Write a class OOPLinkedList that encapsulates the linked list of integers you know from FoCS. Your class should support the addition and removal of elements from the head, querying of the head element, obtaining the n^{th} element and computing the length of the list. You may find it useful to first define a class OOPLinkedListElement to represent a single list element. Do not use Generics.

13. (C*) Write a class to represent a binary tree and adapt it to represent a functional array.

Inheritance

14. (A) 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.

15. (A) If you don't specify an access modifier when you declare a member field of a class, what does Java assign it? Demonstrate your answer by providing minimal Java examples that will and will not compile, as appropriate.

16. (A) 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.

17. (B) Consider the Java class below:

```
package questions;

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

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.

18. (B) Create a class OOPSortedLinkedList that derives from OOPLinkedList but keeps the list elements in ascending order.

19. (C*) Create a class OOPLazySortedLinkedList that derives from OOPSortedLinkedList but avoids performing any sorting until data are expressly requested from it, whereupon it first sorts its contents and then returns the result.

Polymorphism

20. (A) Explain the differences between a class, an abstract class and an interface in Java.

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

22. (A) 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.

23. (A) A Computer Science department keeps track of its CS students using some custom software. Each student is represented by a Student object that features a pass() method that returns true if and only if the student has all six ticks to pass the year. The department suddenly starts teaching NS students, who only need four 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.

24. (C) An alternative implementation of a list uses an array as the underlying data structure rather than a linked list

(a) Write down the asymptotic complexities of the array-based list methods.
(b) Abstract your implementation of OOPLinkedList to extract an appropriate OOPList interface.
(c) Implement OOPArrayList (which should make use of your interface).
(d) When adding items to an array-based list, rather than expanding the array by one each time, the array size is often doubled whenever expansion is required. Analyse this approach to get the asymptotic complexities associated with an insertion.

25. (C*)

(a) Write an interface to represent a queue.

(b) Implement OOPListQueue, which should use two OOPLinkedList objects as per the queues you constructed in your FoCS course. You may need to implement a method to reverse lists.
(c) Implement OOPArrayQueue. Use integer indices to keep track of the head and the tail position.
(d) State the asymptotic complexities of the two approaches.

26. (C) 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 (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

27. (A) Write a small Java program that demonstrates constructor chaining using a hierarchy of three classes as follows: A is the parent of B which 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.

28. (A) Explain why this code prints 0 rather than 7.

```
public class Test {
  public int x=0;
  public void Test() {
    x=7;
  }
  public static void main(String[] args)  {
    Test t = new Test();
    System.out.println(t.x);
  }
}
```

29. (B) Describe how garbage collection works in Java and the issue with finalizers.

Error Handling

30. (B) The following code captures errors using return values. Rewrite it to use exceptions.

```
public class RetValTest {
  public static String sEmail = "";

  public static int extractCamEmail(String sentence) {
    if (sentence==null || sentence.length()==0)
       return -1;  // Error - sentence empty
    String tokens[] = sentence.split(" "); // split into tokens
    for (int i=0; i< tokens.length; i++) {
      if (tokens[i].endsWith("@cam.ac.uk")) {
sEmail=tokens[i];
return 0; // success
      }
    }
    return -2; // Error - no cam email found
  }

  public static void main(String[] args) {
    int ret=RetValTest.extractCamEmail("My email is rkh23@cam.ac.uk");
    if (ret==0) System.out.println("Success: "+RetValTest.sEmail);
    else if (ret==-1) System.out.println("Supplied string empty");
    else System.out.println("No @cam address in supplied string");
  }
}
```

31. (B) Write a Java function that computes the square root of a double number using the Newton-Raphson method. Your function should throw an exception of your own creation if the supplied double is negative.

32. (B*) Comment on the following implementation of pow, which computes the power of a number:

```
public class Answer extends Exception {
  private int mAns;
  public Answer(int a) { mAns=a; }
  public int getAns() {return mAns;}
}

public class ExceptionTest {
  private void powaux(int x, int v, int n) throws Answer {
    if (n==0) throw new Answer(v);
    else powaux(x,v*x,n-1);
  }

  public int pow(int x, int n) {
    try { powaux(x,1,n); }
    catch(Answer a) { return a.getAns(); }
    return 0;
  }
}
```

33. (B*) In Java try...finally blocks can be applied to any code—no catch is needed. The code in the finally block is *guaranteed* to run after that in the try block. Suggest how you could make use of this to emulate the behaviour of a destructor (which is called immediately when indicate we are finished with the object, not at some indeterminate time later).

34. (B*) By experimentation or otherwise, work out what happens when the following method is executed.

```
public static int x() {
   try {return 6;}
   finally { ... }
}
```

Copying Objects

35. (C*) 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?

36. (C) 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();
      return sc;
   }

}
```

Explain what could go wrong, illustrating your answer with an example

37. (B) 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 Collections

38. (A) Using the Java API documentation or otherwise, compare the Java classes Vector, LinkedList, ArrayList and TreeSet.

39. (B) Rewrite your OOPList interface and OOPLinkedList class to support lists of types other than integers using Generics. e.g. OOPLinkedList<Double>.

40. (B) 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.

41. (B*) Research the notion of *wildcards* in Java Generics. Using examples, explain the problem they solve.

42. (B) Why does Java's Generics not support the use of primitive types as the parameterised type?

43. (C*) Java provides the List interface and an abstract class that implements much of it called AbstractList. The intention is that you can extend AbstractList and just fill in a few implementation details to have a Collections-compatible strudture. Write a new class CollectionArrayList that implements a mutable Collections-compatible Generic array-based list using this technique. Comment on any difficulties you encounter.

Object Comparison

44. (A) Write an immutable 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.

45. (A*) Explain why the following code excerpts behave differently when compiled and run (may need some research):

```
String s1 = new String("Hi");
String s2 = new String("Hi");
System.out.println( (s1==s2) );

String s3 = "Hi";
String s4 = "Hi";
System.out.println( (s3==s4) );
```

46. (C) 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) Using a Comparator, show how to keep the collection sorted by {manufacturer, age}. i.e. sort first by manufacturer, and sub-sort by age.

47. (B*) 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

48. (A) Explain the difference between the State pattern and the Strategy pattern.

49. (A) In lectures the examples for the State pattern used academic rank. Explain the problems with the first solution of using direct inheritance of Lecturer and Professor from Academic rather than the State pattern.

50. (C) 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.

51. (B*) Explain how Java uses the Decorator pattern with Reader (yes, research will be required).