OOP Examples Sheet 2

Dr Andrew Rice
Michaelmas 2019
(Material by Dr Robert Harle)

Practical exercises are listed on the course materials page on the department webpages.

Practical exercises associated with this supervision

- Game of Life
- Sorting

Lecture 7: Lifecycle of an Object

7.1. **(W)** 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.

7.2. **(W)** 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);
  }
}
```

7.3. It is often recommended that we make classes immutable wherever possible. How would you expect this to impact garbage collection?

7.4. **(*)** 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).

7.5. By experimentation or otherwise, work out what happens when the following method is executed.

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

Lecture 8: Java Collections and Object Comparison

8.1. **(W)** Using the Java API documentation or otherwise, compare the Java classes Vector, LinkedList, ArrayList and TreeSet.

8.2. **(W)** 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

8.3. Complete the Sorting task

(a) When implementing Merge Sort you will need some temporary space for the merging part of the algorithm. You will find that you are unable to create new arrays of type T i.e. if you try T[] temp = new T[10] you will get a compile error. You can use a plain Object array or a new ArrayList⟨T⟩ instead. Bearing in mind the concept of type-erasure why does this fail to compile? Why does Object[] or ArrayList⟨T⟩ work?

8.4. Complete the Game of Life task

8.5. 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.

8.6. Write a Java program that calculates the average (mean) for a list of integers. Provide three implmentations: 1) using a regular for-loop; 2) using a for-each loop; 3) using an iterator. What are the pros and cons of each?

Lecture 9: Error Handling Revisited

9.1. **(W)** 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) );
```

9.2. 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.

9.3. **(*)** 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.

9.4. **(W)** 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");
  }
}
```

9.5. Write a Java function that computes the square root of a double number using the Newton-Raphson method. Your function should make appropriate use of exceptions *and* assertions.

9.6. 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;
  }
}
```

Lecture 10: More questions on generics...

This section was incorrectly labelled as Lecture 10 in the published version of this examples sheet. I have left this intact in order to preserve the question numbering for those who have already started the work

10.1. **(W)** Explain in detail why Java's Generics not support the use of primitive types as the parameterised type? Why can you not instantiate objects of the template type in generics (i.e. why is new T() forbidden?)

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

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

10.4. **(*)** 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 structure. Write a new class CollectionArrayList that implements a mutable Collections-compatible Generics array-based list using this technique. Comment on any difficulties you encounter.