OOP Examples Sheet 2

Dr Andrew Rice
Michaelmas 2018
(Material by Dr Robert Harle)

Lecture 6: Lifecycle of an Object

6.1. **(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.

6.2. **(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);
  }
}
```

6.3. Write a Java program that takes your tick 1 Java files (ArrayLife and Pattern) and prints out all of the methods and whether they are private or public. (The online checker does exactly this to establish that you did the right thing). *Hint: you will need to make use of the Class class in Java.*

6.4. The lectures described the main mechanism to mark objects that can be deleted (or to mark objects that cannot be deleted), but not the deletion process. Compare and contrast the following approaches in terms of performance. Your answer should consider the costs associated with marking and deleting.

(a) The *mark-sweep* schemes delete the marked objects, leaving their memory available. A list of free memory chunks is maintained.
(b) The *mark-sweep-compact* schemes delete the objects and then compact the momory by moving surviving objects to be adjacent to each other (i.e. no free memory between them).
(c) The *mark-copy* schemes maintain two memory regions, one of which is active. During the marking process, surviving objects are copied into the other region. Once marking is complete, references are upated to the second region, which becomes active, and the first region is deleted.

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

Lecture 7: Java Collections and Object Comparison

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

7.2. **(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

7.3. 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.

7.4. **(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) );
```

7.5. 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.

7.6. **(*)** 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.

Lecture 8: Error Handling Revisited

8.1. **(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");
  }
}
```

8.2. 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.

8.3. 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;
  }
}
```

8.4. **(*)** 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).

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

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

Lecture 9: Copying Objects

9.1. **(W)** Adapt your OOPLinkedList class to be cloneable.

9.2. **(W)** Explain the purpose of marker interfaces.

9.3. 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

9.4. As discussed in lectures, 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 unambiguously a good solution?

9.5. 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];
}
```

Lecture 10: Language Evolution

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. Rewrite your PatternStore class from tick 3 to use lambda functions (where you have multiple criteria to sort by, you might be interested to lookup thenComparing).

10.4. Add a new method `List<String> getNumPatternsRepresentableBy(int x)` in PatternStore that produces a list of patterns requiring fewer than x bits to represent. The method body should use a single statement that uses a Java8 stream. and the string for each pattern should be "NAME H W", where NAME is the pattern name and H and W are the height and width, respectively. The list of strings should be sorted alphabetically.

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

10.6. **(*)** 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.