CS 61B Discussion 5: Inheritance II Fall 2014

1 WeirdList

Below is a partial solution to the WeirdList problem from homework 3 showing only the most important lines.

Part A. Complete the implementation of `WeirdList` by adding a field. Then complete the lines in `UseWeirdList` (on the next page) such that `wlOne` is a WeirdList containing only the number 1, and `wlTwoOne` contains the numbers 2 and 1. Ignore map for now.

Throughout this problem, do *n*ot use any `if`, `switch`, `while`, `for`, `do`, or `try` statements, and do not use the '`?:`' operator.

```
public class WeirdList {
    private int head;
    private WeirdList tail;

    /** The empty sequence of integers. */
    public static final WeirdListEmpty EMPTY = new WeirdListEmpty();

    /** Returns the number of elements in the sequence that
     *  starts with THIS. */
    public int length() {  return 1 + tail.length();   }

    public WeirdList(int h, WeirdList t) { head = h; tail = t; }

    /** Apply FUNC.apply to every element of THIS WeirdList in
     *  sequence, and return a WeirdList of the resulting values. */
    public WeirdList map(IntUnaryFunction func) {
        return new WeirdList(func.apply(head), tail.map(func));
    }
}

public class WeirdListEmpty extends WeirdList{
    public int length() {
        return 0;
    }

    public WeirdListEmpty() {
        super(0, null);
    }

    public WeirdList map(IntUnaryFunction func) {
        return this;  /* Return yourself, a WeirdListEmpty. */
    }
}

public interface IntUnaryFunction {
    /** Return the result of applying this function to X. */
    int apply(int x);
}
```

```
public class UseWeirdList {
    /** Sets wlOne to be a WeirdList containing one.
      * and wlTwoOne to be a WeirdList containing two, then one. */
    public static void main(String[] args) {
        WeirdList wlOne = new WeirdList(1, WeirdList.EMPTY);
        WeirdList wlTwoOne = new WeirdList(2, wlOne);
    }

    /** Returns the maximum non-negative integer in W. If
      * no non-negative integers exist, return -1 instead.
      * Do not use recursion in the code for this method. */
    public static int maxPos(WeirdList w) {
        Maximizer maximizer = new Maximizer();
        L.map(maximizer);
        return maximizer.max;
    }
}

public class Maximizer implements IntUnaryFunction {
    public int max;

    public Maximizer() {
        max = -1;
    }

    public int apply(int x) {
        max = Math.max(x, max);
        return max;
    }
}
```

Part B. Complete the WeirdListEmpty implementation by filling in the return statement for the map function. If you do not know how to proceed, ask your neighbors. If you know the answer, help out your neighbors. Do it quick, the next part is the intersting part.

Part C. Implement the maxPos function. Hint: You'll need to create another class! Do not make recursive calls in your maxPos function or your new class. As throughout this entire problem, do *n*ot use any `if`, `switch`, `while`, `for`, `do`, or `try` statements, and do not use the '`?:`' operator.

Part D. Follow-up: In your solution, did your return value for the apply method matter?

The return value of the `apply` method doesn't matter, because we just end up taking the `max` member from `Maximizer` anyway.

2 TrReader

A complete solution to the TrReader problem from Homework 3 is given below. The `Reader` class is abstract, requires that subclasses implement at least the `close()` and `read(char[], int, int)` abstract methods, and provides a default implementation for a no-argument `read()` method that uses the abstract `read(char[], int, int)` method to do the real work.

```
public class TrReader extends Reader {
    private final Reader subReader;
    private final String from, to;

    public TrReader(Reader str, String f, String t) {
        subReader = str;      from = f;         to = t;
    }

    public void close() throws IOException { subReader.close();  }

    /** Return translation of single char IN using _FROM and _TO.
     *  e.g. if _FROM="abcdefg", _TO="1234567", IN='c', returns '3' */
    private char translateOneChar(char in) { /* omitted for brevity */ }

    public int read(char[] cbuf, int off, int len) throws IOException {
        int numRead = _subReader.read(cbuf, off, len);
        for (int i = off; i < off + numRead; i += 1) {
            cbuf[i] = translateOneChar(cbuf[i]);
        }
        return numRead;
    } } /* bad style but hey what can you do */
```

Consider the code below:

```
    Reader r = new FileReader("TrReaderTest.java");
    Reader trR = new TrReader(r, "import jav.", "josh hug___");
    System.out.println(trR.read());
```

For each of the following, state whether the statement is True, False, or Unknown given the information provided.

1. ____ The `read(char[], int, int)` method is inherited from `FileReader` by `TrReader`.
   F: No TrReader does not extend FileReader

2. ____ Any call to the `read(char[], int, int)` method of a TrReader object results in a call to the `read(char[], int, int)` of `FileReader`.
   F: No, only if the reader happens to be a FileReader.

3. ____ The call to `trR.read()` causes a compilation error since we did not define a `read()` method in `TrReader`.
   F: No, a read() method is inherited from Reader.

4. ____ The call to `trR.read()` results in a call to the `read(char[], int, int)` method of trR.
   T: Yes, that's what the default implementation of read() does.

5. ____ The call to `trR.read()` results in a call to the `read(char[], int, int)` method of some instance of the `FileReader` class.
   T: Yes, since trR was instantiated with a FileReader.

6. ____ The call to `trR.read()` results in a call to the `read()` method of the `FileReader` class.
   U: We don't know if FileReader overrides the default read() method.

3 Extra Questions (not covered in section)

These questions might not make sense until you get around to understanding the basics covered in lectures 12 and 13. If this is the case, make sure to go and understand these lectures, and come back to these questions later.

1. The class RuntimeException has many subclasses. Why do they exist? Why don't we always throw a RuntimeException?

2. As we learned in class (if we were there), there are two types of exceptions: Checked Exceptions and Unchecked Exceptions. Any exception which is a subclass of Error or RuntimeException is unchecked. All other exceptions are considered checked. What are the two ways to keep the compiler happy if a method might throw a checked exception?

3. It might seem strange at first that exceptions that are subtypes of Exception or Error are treated differently, and are allowed to pass silently without being caught or specified. Why are these particular exceptions allowed to go unchecked?

4. The protected keyword specifies that a member can be accessed by other components of a package AND by subclasses. If we omit a keyword, a member is considered package protected, and may only be accessed by other package components (i.e. not by any subclasses). Why do you think the Java designers did this?

5. Suppose we have a class BundleOfDogs that stores objects of type Dog, which can be added using a put() method. Suppose that BundleOfDogs also has a method randomDogGenerator() that returns an object of type RandomDog. Suppose that the RandomDog class is implemented as a nested class (i.e. a class inside BundleOfDogs.java). Would it make more sense for RandomDog to be an inner class or a static class?