Exercise sheet 8

1. Install the MultiJava compiler and the JML tools. There are two options to do this:
   - If you use Eclipse you can add the update site:

     `http://www.sct.inf.ethz.ch/research/universes/tools/eclipse/`

     and install the JML plug-in from there.
     Be sure to use at least a Java 5 virtual machine.
     After that, a “JML Tools” menu item is available for Java files.
     In the project properties you can enable the Universe type system support.
     This was tested on Linux and Windows XP, with a Sun Java 5 VM, and Eclipse 3.2.

   - Otherwise, install the command-line tools available at:

     `http://www.cs.iastate.edu/~leavens/JML/download.shtml`

     After successfully setting the environment variables, you can verify a program with universe-annotations with the following program call:

     ```
     $ java org.jmlspecs.checker.Main --universes
           -w3 -forg.multijava.mjc.UniverseFilter XXX.java
     ```

2. Use the Universe Type System to annotate the ArrayList from the exercise lesson.

3. Given are the following classes for a linked list:

```
class Entry {
     Object element;
     Entry previous, next;

     Entry( Object o, Entry p, Entry n ) {
          element = o; previous = p; next = n; }
}

public class LinkedList {
     private Entry header;

     public LinkedList() {
          header = new Entry(null, null, null);
          header.next = header;
          header.previous = header;
     }

     public void add( Object o ) {
          Entry newE = new Entry(o,header,header.next);
          header.next.previous = newE;
          header.next = newE;
     }
```

```
public Object get( int idx ) {
     Entry e = header.next;

     for( int i=0; i<idx; ++i ) { e = e.next; }

     return e.element;
}

public ReadIterator getReadIterator() {
     return new ReadIterator( header );
}

public DeleteIterator getDeleteIterator() {
     return new DeleteIterator( header );
}

public static void main( String[] args ) {
     LinkedList ll = new LinkedList();
     ll.add( new Integer(20) );
     ll.add( "xyz" );
     ll.add( new Float(2.2f) );
     ll.add( "Hello World" );
     ll.add( new Object() );

     ReadIterator itr = ll.getReadIterator();

     int i = 0;
     while( itr.hasNext() ) {
          itr.moveNext();
          System.out.println("Element[" + i + "]: "
                             + itr.element() );
          ++i;
     }

     DeleteIterator itd = ll.getDeleteIterator();
     itd.moveNext();
     itd.moveNext();
     itd.delete();
     System.out.println("Deleted second element.");

     itr = ll.getReadIterator();

     i = 0;
     while( itr.hasNext() ) {
          itr.moveNext();
          System.out.println("Element[" + i + "]: "
                             + itr.element() );
          ++i;
     }
}
```

```
}
```

The iterators are implemented in the following way:

```
public class ReadIterator {

     public ReadIterator( Entry h ) {
          // the header is a dummy that is never null
          current = h;
          header = h;
     }

     public boolean hasNext() {
          return current.next != header;
     }

     public void moveNext() {
          current = current.next;
     }

     public Object element() {
          return current.element;
     }

     protected Entry current;
     protected Entry header;
}

public class DeleteIterator extends ReadIterator {

     public DeleteIterator( Entry h ) {
          super( h );
     }

     public void delete() {
          if( current.previous != null )
               current.previous.next = current.next;

          if( current.next != null )
               current.next.previous = current.previous;

          current = current.next;
     }
}
```

The source code for the exercise is available on the web page!

a) Use the Universe Type system to guarantee the encapsulation of LinkedList.
b) Annotate the ReadIterator.
c) Annotate the DeleteIterator.
d) Can both of the iterators be typed with the Universe Type system? Provide reasoning for your answer. If needed, change the implementation in a way such that universe typing is possible.

4. Complete exercise 3 from exercise sheet 6 and exercise 4 from exercise sheet 7.

5. **Competition:** Find a program which

a) uses the Universe Type system,
b) can be compiled with the JML compiler without warnings,
c) and still violates the ownership invariants.

6. Question of **Non-Null Type system** from last year's exam:

The appearance of **NullPointerExceptions** in Java-Programs signals a programming error, which can be only found during runtime. A Variable **x** is used for a method invocation or a field access, although the variable does not reference a valid object.

In Java this error could only be found at runtime, because **null** is a valid value for every reference type and thus an assignment of **null** to a variable of a reference type is type correct.

In this exercise you have to extend the type system of a Java like programming language with the following properties:

NullPointerExceptions are avoided with the type system.

The type system offers two types for every class **T** of a program: An expression of Type **nullable T** can be evaluated to **null** or a reference of a **T**-Object. An expression of Type **non_null T** can not be evaluated to **null**. This additional type information is used to check statically that every action that can raise a **NullPointerException** is forbidden. The type system should guarantee that the following holds in all execution states:

All variables and expressions with **non_null** types contain references to non-null objects.

In the following exercises, only the checks and rules which are needed beside the regular checks of Java have to be written down. You can ignore object initialization, arrays and generic types.

a) Which subtype relation has **nullable T** and **non_null T**? What is the relation between those types and the Null Type (the Type with the constant **null**)?

b) Which additional type checks have to be done in method calls `x.m(a)`?

c) Given is the following Code segment:

```
class Example {
  nullable Field field;

  void m() {
    nullable Local local = ...;

    if( local != null ) { someop(); local.opl(); ... }
    if( field != null ) { someop(); field.opf(); ... }
  }

  void someop() { /* unknown */ }
}
```

Will the method call on **local** be forbidden with the rules from b)? Would it be safe to allow the call? Give reasons for your answer!

Will the method call on **field** be forbidden with the rules from b)? Would it be safe to allow the call? Give reasons for your answer!