Exercise Sheet 5

1. What is the output of the subsequent program? Which methods are statically bound and which dynamically bound? How many methods does an object of the class Test4 have?

```
class C1 {
    private void m() { System.out.println("C1.m"); }

    protected void l() { m(); }
}

public class Test4 extends C1 {

    private void m() { System.out.println("C2.m"); }

    public static void main( String[] args ) {
        C1 c1 = new Test4();
        c1.l();
    }
}
```

2. Same question, different program:

```
class C1 {
    private void m() { System.out.println("C1.m"); }

    protected void l() { m(); }
}

public class Test5 extends C1 {

    private void m() { System.out.println("C2.m"); }

    protected void l() { m(); }

    public static void main( String[] args ) {
        C1 c1 = new Test5();
        c1.l();
    }
}
```

3. Given the following double linked list:

```
public class DList<T> {

      protected class Node<T> {
        Node<T> prev, next;
        T data;
      }

      private Node<T> first, last;

      public DList() {
        first = new Node<T>();
        last = new Node<T>();
        first.next = last;
        last.prev = first;
      }

      public DList( T[] d ) {
        this();
        for( T x : d ) {
             add( x );
        }
      }

      public void add( T d ) {
        Node<T> n = new Node<T>();
        n.data = d;
        n.prev = last.prev;
        n.next = last;
        last.prev.next = n;
        last.prev = n;
      }

      public T getFirst() {
        return getFirstNode().data;
      }

      protected Node<T> getFirstNode() {
        return first.next;
      }

      public static void main( String[] args ) {
        Integer[] ia = new Integer[] {1, 2, 3};
        DList<Integer> dl = new DList<Integer>( ia );

        ia[0] = 5;

        System.out.println("First element: " + dl.getFirst());
      }
 }
```

a. Change the implementation, such that an array is used for storing the data. The external observable behavior of the class should not change.
b. The method `getFirstNode` is not needed anymore. Which classes need to be adapted if we remove the method?
c. Is there a different behavior between linked-lists and arrays in the main method?

4. Argue for each of the following classes that they preserve encapsulation. Do all the constructors and methods preserve the invariants? Is it guaranteed that other classes do not violate the consistency of the objects?

```
public class FourA {
    private int a;
    private int b;

    /*@ invariant a >= b; @*/

    /*@ requires ta >= 0; @*/
    public FourA(int ta) {
         a = ta;
         b = 0;
    }

    public void increment() {
         ++a;
         ++b;
    }
}

public class FourB {
    public int a;
    public int b;

    /*@ invariant a >= b; @*/

    /*@ requires ta >= 0; @*/
    public FourB(int ta) {
         a = ta;
         b = 0;
    }

    public void increment() {
         ++a;
         ++b;
    }
}
```

```
public class FourC {
    private int a;
    private int b;
    public int c;

    /*@ invariant a >= b; @*/

    /*@ requires ta >= 0; @*/
    public FourC(int ta) {
         a = ta;
         b = 0;
    }

    public void increment() {
         ++a;
         ++b;
    }
}
```

5. **Typing example from last years exam!**
   Assume we have an operator `\type(x)`, which returns the dynamic type of the reference `x`. The operator `<:` expresses the subtype-relation. `S <: T` hold, if `S=T` or `S` is a subtype of `T`. You can use these operators to express the typing of a program in the interface-definition, vis. precondition, postcondition, and invariant.

```
class Exa {
   Object a;

   Object m(Object c) { ... }
}

class ExaTyped {
   A a;

   B m(C c) { ... }
}
```

   a. Specify the class Exa with the help of the operators `\type(x)` and `<:` in a way, such that only objects of type `C` (with subtypes) can be used as parameter for `m`. Do not change the typing of `Exa`.

   b. Specify the class Exa with the help of the operators `\type(x)` and `<:` in a way, such that only objects of type `B` (with subtypes) will be given back as return value for `m`. Do not change the typing of `Exa`.

   c. Specify the class Exa with the help of the operators `\type(x)` and `<:` in a way, such that only objects of type `A` (with subtypes) can be stored in the field `a`. Do not change the typing of `Exa`.

d. Is the specified version of the class `Exa` (the solution of the previous question) equivalent to the stronger typed class `ExaTyped`? Give an implementation of the method `m` showing the differences.

6. Co-, Contra-, and In-variant example from last years exam

Given is the following program-fragment of a Java-like programming language:

```
class Super {
   B m(C c) { ... }
}
class Sub extends Super {
   E m(F c) { ... }
}
```

This programming language allows you to adopt the parameter and return type of methods in subclasses. The program language designer can not determine which rules he has to enforce to guarantee static type safety.

a. Which relationship has to be between the parameter types `C` and `F`? Is this rule co-, contra- or in-variant?

b. Which relation ship has to be between the return types `B` and `E`? Is this rule co-, contra- or in-variant?

c. For each of them give a code-fragment showing the problems, which arise without such restrictions.