Übungsblatt 5

1. Was ist die Ausgabe nachfolgenden Programms? Welche Methoden werden statisch und welche werden dynamisch gebunden? Wieviele verschiedene Methoden hat ein Objekt der Klasse Test4 / Test5?

```
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. Selbe Fragen, anderes Programm:

```
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. Gegeben ist folgende doppelt verkettete 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. Ändere die Implementierung so, dass ein Array zur Speicherung der Daten verwendet wird. Das extern sichtbare Verhalten der Klasse soll sich dabei allerdings nicht ändern.
b. Die Methode `getFirstNode` wird in der Array-Implementierung nicht mehr benötigt. Welche Klassen müssen möglicherweise angepasst werden, wenn man die Methode komplett entfernen will?
c. Gibt es mit dem Code in der `main` Methode ein unterschiedliches Verhalten zwischen der verketteten Liste und dem Array?

4. Erläutere für jede der folgenden Klassen, ob deren Objekte vollständig gekapselt sind. Erhalten alle Konstruktoren und Methoden die Invarianten? Ist sichergestellt, dass andere Klassen die Konsistenz der Objekte nicht verletzen können?

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

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

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

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

5. **Klausurbeispiel vom letzten Jahr!** Typisierung
   Es gebe einen Operator `\type(x)`, der den dynamischen Typ der Referenz `x` liefert. Weiters gebe es den Operator `<:` um die Subtyp-Beziehung auszudrücken. `S <: T` genau dann, wenn `S=T` oder `S` ist ein Subtyp von `T`.
   Man kann diese Operatoren verwenden, um die Typisierung eines Programms in der Schnittstellen-Spezifikation (durch Vorbedingungen, Nachbedingungen und Invarianten) auszudrücken.

```
class Exa {
   Object a;

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

class ExaTyped {
   A a;

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

   a. Spezifizieren Sie die Klasse `Exa` mit Hilfe der Operatoren `\type(x)` und `<:` so, dass nur Objekte vom Typ `C` (einschliesslich Subtypen) als Parameter an `m` übergeben werden können. Lassen Sie die Typisierung von `Exa` unverändert.

   b. Spezifizieren Sie die Klasse `Exa` mit Hilfe der Operatoren `\type(x)` und `<:` so, dass nur Objekte vom Typ `B` (einschliesslich Subtypen) als Rückgabewert von `m` zurückgegeben werden können. Lassen Sie die Typisierung von `Exa` unverändert.

c. Spezifizieren Sie die Klasse `Exa` mit Hilfe der Operatoren `\type(x)` und `<:` so, dass nur Objekte vom Type `A` (einschliesslich Subtypen) im Feld `a` gespeichert sein können. Lassen Sie die Typisierung von `Exa` unverändert.

d. Ist die spezifizierte Version der Klasse `Exa` (die Lösungen der vorhergehenden Teilaufgaben) äquivalent zur stärker typisierten Klasse `ExaTyped`? Geben Sie eine Implementierung der Methode `m` an, die die Unterschiede aufzeigt.

6. **Klausurbeispiel vom letzten Jahr!** Co-, Contra- und Invarianz
Gegeben sei das folgende Programmfragment in einer Java-ähnlichen Programmiersprache:

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

In dieser Programmiersprache ist es erlaubt, die Parametertypen und den Rückgabetyp von Methoden in Subklassen anzupassen. Der Programmiersprachen-Designer kann sich allerdings nicht entscheiden, welche Regeln er dafür einführen muss, um statische Typsicherheit zu garantieren.

a. In welcher Beziehung müssen die Parametertypen `C` und `F` stehen? Bezeichnet man diese Regel als Co-, Contra- oder Invarianz?

b. In welcher Beziehung müssen die Rückgabetypen `B` und `E` stehen? Bezeichnet man diese Regel als Co-, Contra- oder Invarianz?

c. Geben Sie jeweils ein kurzes Codefragment an, an dem man sieht, warum es ohne die jeweilige Restriktion zu Problemen kommt.