Exercise Sheet 13

1. Given is the following interface, which is supposed to represent a set of integers.

```
interface IntSet {

  // requires !has(i)
  // ensures  has(i)
  void add(int i);

  // requires has(i)
  // ensures  !has(i)
  void remove(int i);

  boolean isEmpty();
  boolean has(int i);
}
```

a. Implement class `ArrayIntSet` that realizes the interface by storing the integers in an array. Specify the class by invariants, pre- and postconditions, and modifies clauses.

b. Argue that all assertions and preconditions of called methods hold in the client code below. Strengthen your specification in class `ArrayIntSet` in case it proved to be too weak.

```
// requires iset != null
// requires iset.has(0) && iset.has(5) && !iset.has(10)
void foo(ArrayIntSet iset) {
  assert iset.has(5);
  iset.remove(0);
  assert !iset.has(10);
  assert iset.has(5);
  iset.add(10);
  assert iset.has(5);
}
```

c. Does your specification in `ArrayIntSet` preserve information hiding? If not, rewrite the specification accordingly.

2. Given is the following class:

```
class List {
   int i;
   List next;
   // invariant  next != null => i == next.i

   // increment i in the whole list
   void inc() { … }
}
```

a. Write a formal specification for method `inc()`.
b. Write down what can be assumed and what needs to be proven for `inc()` in the standard invariant semantics.
c. Write a recursive implementation of the method `inc()`.
d. Verify if the implementation fulfills the assertions defined in b.
e. If not, write an alternative solution that fulfills the assertions.
f. Reconsider your previous solutions if the refined invariant semantics is applied.

3. Homework.

The solution in Java to implement the observer pattern is to have the observers extend and implement the `java.util.Observer` interface. Program an observer pattern that uses reflection and removes the necessity of using a pre-defined interface.