Exercise 3

Behavioral Subtyping

1. Let `sortedArray` be a Java class, which supports a single private field `content`. The field `content` must be an array of integers with no duplicates, which is sorted in increasing order. The following is a method for the insertion of a value in the array:

```
void insert (int newElement)
{
       int[] newContent = new int[content.length+1];
       int i = 0;
       while (i<content.length && content[i]< newElement)
       {
              newContent[i]=content[i];
              i++;
       }
       newContent[i]= newElement;
       while (i<content.length)
       {
              newContent[i+1]= content[i];
              i++;
       }
       content=newContent;
}
```

   Write an appropriate invariant for the class, as well as a pre- and postcondition for the method `insert`. Make the specification as precise as possible. In these conditions, you may use the logical quantifiers ∀ and ∃.

2. Consider the following code:

```
class A
{
  int x;
  public int getX () {return x;}
  public void setX (int x) {this.x = x;}
};

class B
{
  //requires a1 != a2 && x1 > x2;
  //ensures result > 0;
  int m (A a1, A a2, int x1, int x2)
  {
    a1.setX (x1);
    a2.setX (x2);
    return a1.getX() - a2.getX();
  }
}
```

Add pre/post conditions to the methods of class `A` in a way that satisfies the postcondition of method `m`.

3. Let `C` be a class with an integer field `x` and a method `m`. Let `m` have
   - Precondition `x>0`
   - Postcondition `x<1`

   Suppose now that there is a class `D` with an integer field `x` and a method `m`. In which of the following cases does the displayed specification of `m` in `D` permit `D` to be a behavioral subtype of `C`?

   (a) Pre `x > 0` Post `x < -1`
   (b) Pre `x > 0` Post `x < 2`
   (c) Pre `x > -1` Post `x < 1`
   (d) Pre `x > 2` Post `x < 1`
   (e) Pre `x > -4` Post `x < -old(x)*old(x)`
   (f) Pre `true` Post `false`

4. Consider the following Java code annotated with specifications:

```
class A
{
        int f;
        //invariant f>0;

        //requires p > f;
        //ensures result > 0;
        int m (int p) { … }
};

class B
{
        int f;
        //invariant f>0;

        //requires p > 0;
        //ensures result == old(p) + f;
        int m (int p) { … }
};
```

Can you prove that `B` is a behavioral subtype of `A`? If not, then provide a generalization of the behavioral subtyping rules that:

- Has the substitution property.
- Is flexible enough to prove that `B` is a behavioral subtype of `A`.

5. You are provided with three classes `ArrayNonDecreasing`, `ArrayIncreasing`, and `ArrayNoDuplicates`. Each of them has a private field `content` of type array of integers. The classes have the following properties:
   - `ArrayNonDecreasing` – elements of `content` must be in non-decreasing order
   - `ArrayIncreasing` - elements of `content` must be in increasing order
   - `ArrayNoDuplicates` – elements of `content` must be unique

   (a) Write invariants that express these properties
   (b) Suppose that there are no mutator methods. Could there be any behavioral subtype relations between these classes? If yes, then describe them.
   (c) Can you find mutator methods that break the potential for behavioral subtyping relations between the classes?

6. Consider two classes `Stack` and `Queue`, implementing the obvious data structures, both of which have methods with the following signatures:

```
void push(Object o);
Object pop();
bool isEmpty();
int size();
void reverse();
```

   - Despite having identical signatures, these two classes cannot be behavioral subtypes of one another. Why not?
   - When implementing these two classes, is there any possibility of code re-use? If so, give details.
   - What programming languages/features could support such code re-use without subtyping? Which of these do you think would be most suitable here?