Description of IntegerArrayClass

-Developing the class with CodeContracts is quite easy, I've done that like this

```
using System;
using System.Diagnostics.Contracts;
using System.Linq;

public class IntegerArray
{
    private int[] integerArray;

    public IntegerArray()
    {
        integerArray = new int[] { };
    }

    [ContractInvariantMethod]
    private void IntegerArrayInvariant()
    {
        Contract.Invariant(this.integerArray != null);
        var ints = this.integerArray as int[];
        Contract.Invariant(ints != null);
    }

    public void addElem(int x)
    {
        Contract.Requires(integerArray != null);
        Contract.Requires(integerArray.Length >= 0);
        Contract.Assume(this.integerArray.Contains(x));
        Contract.Ensures(this.integerArray != null);
        Contract.Ensures(this.integerArray.Contains(x));

        var length = integerArray.Length;

        var newArray = new int[length + 1];

        for (var i = 0; i < length; i++)
        {
            Contract.Assert(newArray.Length > integerArray.Length);
            Contract.Assert(newArray != null);
            Contract.Assert(integerArray != null);
            Contract.Assert(i > 0);

            newArray[i] = integerArray[i];
        }

        newArray[length] = x;

        integerArray = newArray;

        Contract.Ensures(length > 0);
```

```
        Contract.Ensures(this.integerArray.Length > length);
    }

    public void delElem(int x)
    {
        Contract.Requires(integerArray != null);
        Contract.Requires(integerArray.Length > 0);
        Contract.Ensures(this.integerArray != null);
        Contract.Assume(!this.integerArray.Contains(x));
        Contract.Ensures(!this.integerArray.Contains(x));

        var length = integerArray.Length;

        var newArray = new int[length - 1];

        for (var i = 0; i < length; i++)
        {
            Contract.Assert(newArray != null);
            Contract.Assert(integerArray != null);
            Contract.Assert(newArray.Length < integerArray.Length);
            Contract.Assert(i > 0);

            if (integerArray[i] != x)
            {
                newArray[i] = integerArray[i];
            }
        }

        integerArray = newArray;

        Contract.Ensures(this.integerArray.Length < length);
    }

    public void DisplayArray()
    {
        Contract.Requires(integerArray != null);
        Contract.Requires(integerArray.Length > 0);

        foreach (var item in integerArray)
        {
            Console.WriteLine(item);
        }
    }
}
```

The class contains an invariang method called *[ContractInvariantMethod]* that represents a condition that should be true for any instances of this class. In our case we want to have an integer array that is instantiated.

There is a private integer array can can be modified via addElem and delElem methods.

In those method we can see Preconditions and Postconditions given by the commands *Contract.Requires* and *Contract.Ensures*. In case of addition of an element we want it to be an int and our array to be initialized, and

as a result our array should contain one more element and our element should be in the array. In case of deletion of an element we want it to be an int and our array to be initialized and to contain minimum 1 element, and as a result our array should contain one less element and our element should not be in the array anymore.

The loop invariants are conditions that are true for each iteration in a loop, we can check this by using *Contract.Assert* command. In our case for addition we want our array to keep being != null and our iterators to be positive. The same for deletion

Dafny

```
class IntegerArray{
  var integerArray: set<int>;

  method addElem(x: int)
  requires integerArray != {}
  requires x > 0
  ensures integerArray != {}
  {
    var length := |integerArray|;

    var newArray : set<int>;
  }

  method delElem(x: int)
  requires integerArray != {}
  {
  }
}
```

With Dafny I've tried to describe the class, describe a simple postcondition and precondition but failed to do operations on sets.