Assignment 2 - Specifications and Modeling

Exercise 1

A way to document the code is to use *contracts*. For this exercise, you will have to consider:

- *preconditions*: the conditions the caller of a method has to fulfill to be able to perform the call (e.g., to call the `sqrt(x)` method, which computes the square root of its argument, the caller has to provide a value `x>=0`).
- *postconditions*: the guarantees the caller gets after the method is executed (e.g., the method `sqrt(x)` will return a `result` such that `result*result == x`).
- *invariants*: the conditions that all the instances of a class have to satisfy while they can be observed by the clients (e.g., for a `BankAccount` the `balance >= 0`, after every `deposit` and `withdraw` operation).

Contracts are part of the *specification*, not of the implementation, so they cannot modify the program state. Therefore, only pure expressions can be used in preconditions, postconditions and invariants.

Consider a simple game, in which a player can move left, right, up and down on a `n*n` board and a partial Java implementation:

```
public class Player {
    private int x, y;
    public final int n;

    public Player(int x, int y, int n) {
       this.x = x;
       this.y = y;
       this.n = n;
    }

    public int getX() {
      return x;
    }
```

```
    public int getY() {
      return y;
    }

    public void moveLeft() {
       x = x - 1;
       if (x < 0) {
           x = n - 1;
       }
    }
    ...
}
```

Write a suitable invariant for the class `Player`, a precondition for the constructor and a postcondition for the method `moveLeft`.

Exercise 2

Consider the following C# code:

```
public class Bag
{
    private int[] elems;
    private int count;

    public Bag(int[] initialElements) {
        this.count = initialElements.Length;
        int[] e = new int[initialElements.Length];
        initialElements.CopyTo(e, 0);
        this.elems = e;
    }

    public Bag(int[] initialElements, int start, int howMany) {
        this.count = howMany;
        int[] e = new int[howMany];
        Array.Copy(initialElements, start, e, 0, howMany);
        this.elems = e;
    }

    public int Count() {
        return count;
    }

    public int[] GetElements() {
        return elems;
    }

    public int RemoveMin() {
        int m = System.Int32.MaxValue;
```

```
        int mindex = 0;
        for (int i = 0; i < count; i++) {
            if (elems[i] < m) {
                mindex = i;
                m = elems[i];
            }
        }
        count--;
        elems[mindex] = elems[count];
        return m;
    }

    public void Add(int x) {
        if (count == elems.Length) {
            int[] b = new int[2*elems.Length];
            Array.Copy(elems, 0, b, 0, elems.Length);
            elems = b;
        }
        elems[count] = x;
        count++;
    }

}
```

Find class invariants and preconditions for all the methods of the class `Bag`, and postconditions for the method `Add`. Express them using the syntax of C#'s Code Contracts:

- At the beginning of each method, `Contract.Requires(expr);` can be used to denote a precondition. Here, `expr` should be a pure boolean C# expression referring only to fields and pure methods with greater or equal visibility than the method. For example, in the precondition of a `public` method, we are not allowed to mention `private` fields.
- Similarly, `Contract.Ensures(expr);` can be used to denote a postcondition.
- If needed, `Contract.ForAll(lower, upper, pred)` can be used to express that the predicate `pred` holds for all integers from `lower` (inclusive) to `upper` (exclusive). For example, the predicate `is_digit` holds for all integers from 0 to 10.
- `Contract.OldValue(expr)` can be used in a postcondition to refer to the value of the `expr` before the execution of the method.
- Class invariants are denoted in a special contract invariant method. Again, the visibility of all referred fields must be greater or equal to the visibility of the contract invariant method.

```
[ContractInvariantMethod]
private void ObjectInvariant() {
        Contract.Invariant(expr);
        ...
}
```

- Methods which have no side effects can be marked with the `[Pure]` attribute (in the line before the method declaration). Only pure methods can be used in contract expressions. For example, getters are pure methods, because they do not modify the state.

Exercise 3

1. Draw a UML class diagram for the system described below:
   (a) every student is either undergraduate or graduate (no student can be in both categories at the same time);
   (b) a student should register at a university, and only registered students are legal students;
   (c) every student has a unique student ID, and he or she has only one major;
   (d) students with the same major are regarded as classmates; students can have several classmates.
2. Which properties of the system above cannot be captured using UML class diagrams?

Exercise 4

UML allows some properties of a generalization (such as relations) to be *redefined*, instead of being inherited. The purpose of redefinition is to add more specific constraints, which are particular to the specialized instances, but which *do not contradict* the existing constraints (the new constraints must be *stronger* than the old ones).

1. Use redefinition (marked in UML as `{redefines property_name}`) to create the UML class diagram for the following system:
   (a) a company has one or more employees and each employee works for exactly one company.
   (b) each employee can work on one or more projects at the same time.
   (c) junior employees cannot be assigned to more than two projects.

(d) summer interns (which are employed for 1 or 2 months) can work on only one development project.

2. Discuss different ways of implementing the UML diagram in Java.