Exercise #6

If you haven't already, read the relevant parts of the course materials and try to answer the questions at the end of the chapter. This exercise sheet is meant as a preparatory exercise for the second exam on the 3^{rd} December.

scriptum up until page 267.

Exercise group meeting

Aufgabe 1: Think of a question

Prepare a question concerning programming for the next round of voluntary exercise group meetings (3^{rd} - 5^{th} December). The question does not have to pertain to a subject covered in the exam.

Cycle

`Cycle` is a data structure that works similarly to a single-linked list. The only difference between the two is that the last element of `Cycle` points to the first element of `Cycle` instead of `null`.

Listing 1: Cycle

```
public class Cycle {

    private int elem;
    private Cycle next;

    public Cycle(int value, Cycle cycle) {
        elem = value;
        if (cycle == null) {
            next = this;
        } else {
            next = cycle.next;
            cycle.next = this;
        }
    }
}
```

Aufgabe 2: output

Add the method `String toString()` to `Cycle`. The method should return the supplied element itself and all elements reachable from it. Also, add a method `int count()` that returns the number of elements within the `Cycle`. Note that you will have to detect the end of the `Cycle`, to avoid an infinite loop.

Aufgabe 3: equals

Add the method `boolean equals(Object o)` to `Cycle`. The method should check whether both Cycles contain the same elements. Pay attention to the implementation guidelines for the `equals` method mentioned in the Java-API and handle all special cases appropriately . Argue why the substitution principle would be violated if these special cases were not explicitely considered. Also, give an example of a method that would no longer work if these special cases were not considered.

The method may *never* throw a NullPointerException or ClassCastException, no matter which object is supplied in the parameter o

Aufgabe 4: return the elements

Add a method `int getFirst()` to `Cycle`. The method should simply return `this.elem`. Also, add a method `int getIndex(int n)` that returns the n^{th} element of the `Cycle`. Finally, add a method `int getLast()` which returns the final element of the `Cycle` (the one that points back to the first one).

Aufgabe 5: fold

Add a method `int fold()` that calculates the sum of all elements and returns it.

Aufgabe 6: map

Add a method `Cycle map()` that returns a new `Cycle` object. This new object should have the same structure as `this`, but the elements should be `%5` (remainder of integer division by 5) of the original ones.

Aufgabe 7: reduce

Add a method `Cycle reduce()` that returns a new `Cycle` object. The new `Cycle` should not contain any negative elements, but all non-negative elements should be in their original order.

Aufgabe 8: Testing

Add a static `main` method that creates at least two `Cycle` objects, each with at least five elements. Call `map`, `reduce` and `fold` for each of these. Now, compare two instances of `Cycle` with `equals` where that method returns `true`. Finally, print all the objects to the console, using `toString()`.

Aufgabe 9: Visualize

Sketch your testing program's behavior. Consider the constructor's behavior especially. Also, sketch the links between the individual objects.

cf. figure 4.5, scriptum page 252

Aufgabe 10: Runtime estimation

Estimate the algorithmic costs of `int getFirst()` and `int fold()`. Consider the average and maximum runtime of these methods, as well as their memory usage. Use Big-O notation.