# Object Orientation – Spring 2018 **Assignment 8**

Since there will be no lab session next week (Good Friday), we decided to shorten this assignment substantially: Only part 1 and part 2 are mandatory. Of course, to execute your test cases you need implementations of all operations. On Blackboard you can find a complete version of class `Polynomial`. *Beware of bugs in the provided code; we have only written it, not tested it thoroughly*[1].

## 1 Polynomials

Polynomials are not only very common in mathematics, but are also frequently used in other fields of science, such as physics and statistics. Such a polynomial consists of a sum of *terms*, where every term has the form $cx^n$. The *factor* $c$ is a real number unequal to zero. The *exponent* $n$ is a natural number. All exponents in the polynomial are different. The highest exponent in a polynomial determines the *degree* of that polynomial. For example, $3x^4 - 4x^3 + 2x - 8$ is of degree 4.

## 2 Learning goals

In this exercise, we ask you to design classes that implement polynomials and test those classes. After doing this exercise, you should be able to:

- Use *(list) iterators* to inspect a data type, to add new elements or to remove elements;
- make 'unit tests' and apply them to your program to test your code.

## 3 The problem

A polynomial is represented by a class called `Polynomial`. Because a polynomial can have an arbitrary number of terms, we use `Lists` to keep track of the terms of a polynomial. We do not allow terms to have a factor 0 and we do not want there to be multiple terms with the same exponent. Also, it turns out to be helpful to arrange terms in the ascending order of their exponents.

To help you get started, we put a framework on Blackboard with the class `Polynomial` and the class `Term`. The `Polynomial` class contains three constructors: one without arguments for making an empty polynomial, a copy constructor and a constructor that takes a string as an argument with the terms (as a sequence of factors alternating with exponents). Take a look at the given code to understand how this works. As you can see, the last constructor uses the *static* method `scanTerm` from the class `Term`. How this works precisely may not be clear as of now, but is not important for making the rest of the assignment. Both of the given classes can be used as a basis for the rest of your implementation.

You can also find the headers of the standard `equals` method in the framework, and the operators `+`, `-`, `*` and `/`. These operators perform addition, substraction, multiplication and division respectively. The division is only added for completeness. You are not expected to implement this, but are more than welcome to do so if you are looking for an extra challenge. The operations are examples of so called symbolic manipulations. For example, $(x^3 + 2x^2) + (x^3 - 2x^2 + x) = 2x^3 + x$. Contrary to *C++*, Java does not have *operator overloading*. This means that you cannot define your own version of `+` in Java. Therefore, you will have to give your operators a name, such as `plus` for addition.

[1] Donald Knuth (1977), slightly rephrased.

When doing this assignment, you should implement the operators to be *destructive*. For example, addition (`plus`)

```
public void plus( Polynomial p )
```

will add the given polynomial `p` to the `this` object and the result is in the `this` object.

Besides the missing implementation of the methods, the JavaDoc is also missing. Feel free to add this as well.

### Assignments

1. Make a project in NetBeans (or Eclipse) and add the given files to it. It is advisable to place `Polynomial` and `Term` in a separate package. Generate a test package for `Polynomial` (in NetBeans: **Tools** → Create/Update Tests)
2. For every operator/method, think of examples that are representative (*test cases*). Take note of the edge cases. For example, terms can disappear during addition because their factor has become zero. Use the test cases to check if your implementation takes this into account. Also implement some test cases that use multiple operators. For example: substraction is the same as multiplying by $-1$ and then addition. Also test whether your operators are *distributive*, *associative* and *commutative*
3. Implement your test cases by adding testing code to the generated testing classes
4. Implement the following operations:
   (a) The `toString` method that generates a text representation of a polynomial.
   (b) The `plus`, `minus` and `times` operators. Try to avoid the use of indices in your list by working only with (list) iterators.

## 4 Handing in

Before Sunday March 26th, 23:59, via Blackboard.