— 1 —

GUI — Problems

1.1 Simple classes

• **Problem 1.1.**
Create a class objects of which represent real polynomials

$$P(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n$$

Polynomial can be internally represented by a list (e.g., **ArrayList**) of coefficients a_i. The default constructor creates the polynomial $P(x) = 0$. Another constructor creates a polynomial based on a list or an array (of any length) of coefficients a_i passed to it. Yet another one takes an object of our class and creates a new polynomial idendical to the one passed by argument, but independent of the original (something like the copy constructor in C++).

Define methods and static functions allowing the user to perform the following operations on our polynomials:

- add a polynomial to *this* polynomial (modifying *this* polynomial); the method should return **this** to allow cascading invocations;
- subtract a polynomial from *this* polynomial;
- add two polynomials getting the third, independent of the two argument polynomials (static function);
- subtract two polynomials getting the third, independent of the two argument polynomials (static function);
- multiply *this* polynomial by another one (modifying *this* polynomial); the method should return **this** to allow cascading invocations;
- multiply two polynomials getting the third, independent of the two argument polynomials (static function);
- multiply this polynomial by a number (modifying *this* polynomial); the method should return **this** to allow cascading invocations;
- multiply a polynomial by a number getting new polynomial (static function);
- calculate the value of *this* polynomial for a given real argumenti x;
- modify any coefficient of *this* polynomial (possibly a coefficient which corresponds to the power of x higher than the highest existing before modification);

- get a concise string representation of polynomial (overriding the **toString** method); e.g., in the form `1+x^3-1.25x^7`.

Write a program testing all these operations.

• **Problem 1.2.**
Write a class **Rectagle** which represents rectangles on a Cartesian plane (assuming that their sides are always parallel to the axes) and a class **Point** representing points on the plane. Rectangle may be internally represented by two points – lower-left and upper-right vertices. [Another possible representation could be, e.g., upper-left vertex, width and height]. Constructors of the class can take two points or four numbers – coordinates of the upper-right vertex, width and hight.

Implement the following fuctionality:

- method **contains** invoked on a rectangle checks if a point passed by argument belongs to the rectangle;
- method **isIn** invoked on a point check if the point belongs to the rectangle passed by argument;
- static function which takes two rectangles and returns new rectangle which is the intersection of the two given rectangles (or **null** if the intersection is empty).
- static function which takes two rectangles and returns new rectangle which is the smallest rectangle possible containg both rectangles passed to the function;
- methods which return the area and circumference of a given rectangle;
- methods returning (as **Point**s) all four vertices of a given rectangle, e.g., **getBottomLeft**, **getBottomRight**, **getTopLeft**, **getTopRight**;
- overriden **toString** methods in both classes.

Write a program which tests all these functions and methods.