TECHNISCHE UNIVERSITÄT DRESDEN

Software Technology Group

Faculty of Computer Science Institute of Software and Multimedia Technology, Software Technology Group

WS2018/19 – Design Patterns and Frameworks

Extensibility Patterns

Professor: Prof. Dr. Uwe Aßmann
Lectuer: Dr.-Ing. Sebastian Götz
Tutor: Dr. rer. nat. Marvin Triebel

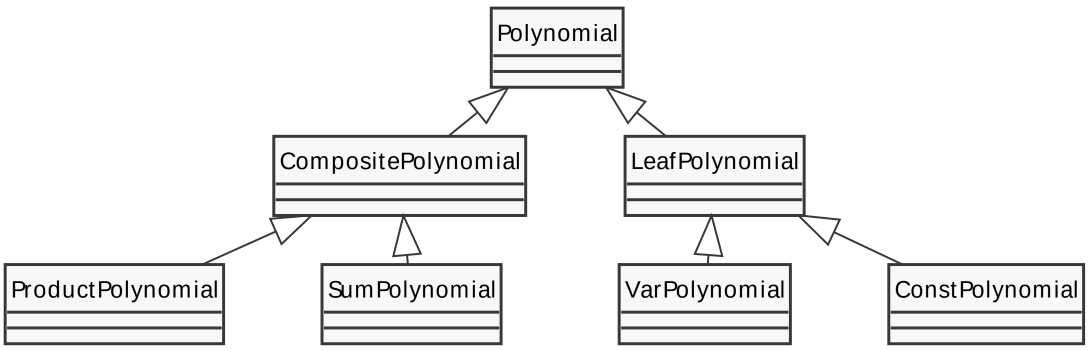
Task 1 The Degree of Polynomials

Consider the set of polynomials over one variable (x) and their degree[1]. Examples are:

$$2x^2 - 5 \text{ with degree } 2 \quad (1)$$
$$x(177x - 15x) \text{ with degree } 2 \quad (2)$$
$$\left(x - 2x^2\right)(2 + 4x) \text{ with degree } 3 \quad (3)$$

a) Which design pattern can be used for representing polynomials? Draw the class diagram!

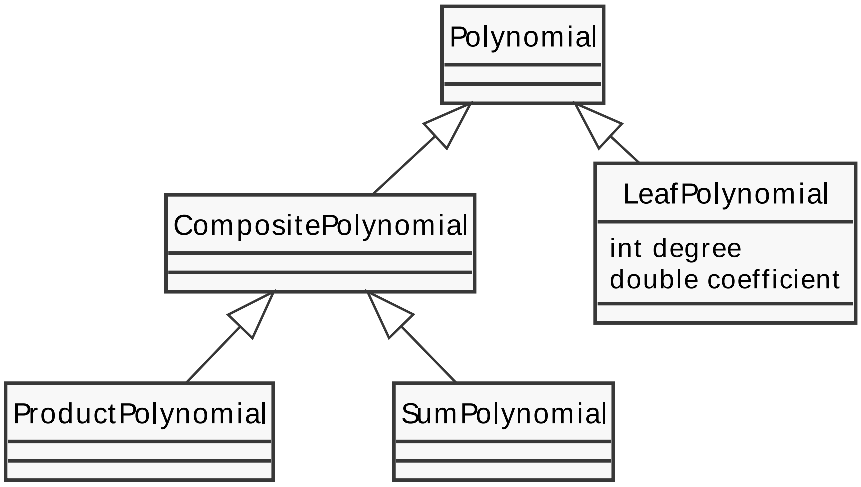
Solution: We can use the *Composite* Pattern, as shown in the following diagram.



b) What is the smallest yet reasonable amount of classes in the diagram?

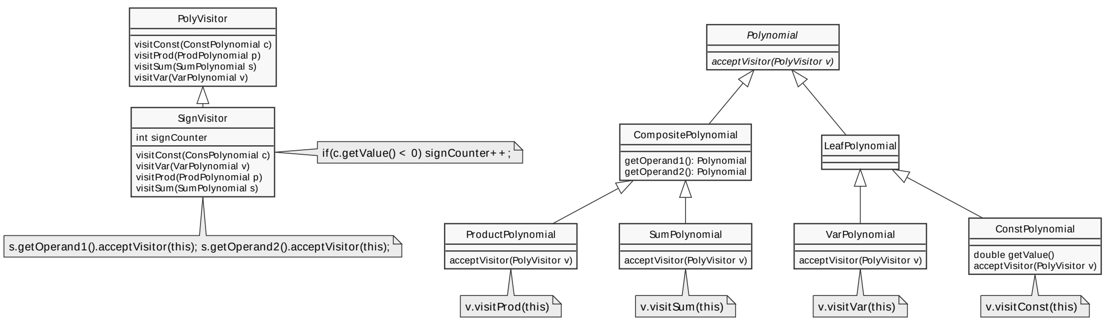
[1] https://en.wikipedia.org/wiki/Degree_of_a_polynomial

Solution: As discussed in the exercise, this depends on the context and many variations exist. One may also include a composite class for Exponentiation or for Brackets. One way to have a smaller number of classes is to merge LeafPolynomial into one class:



c) The function `int countSigns()` shall count the number of minus signs in a given polynomial. Which design patterns are suitable? Which patterns have which (dis)advantages?

Solution: The *Visitor Pattern* can be applied as follows.



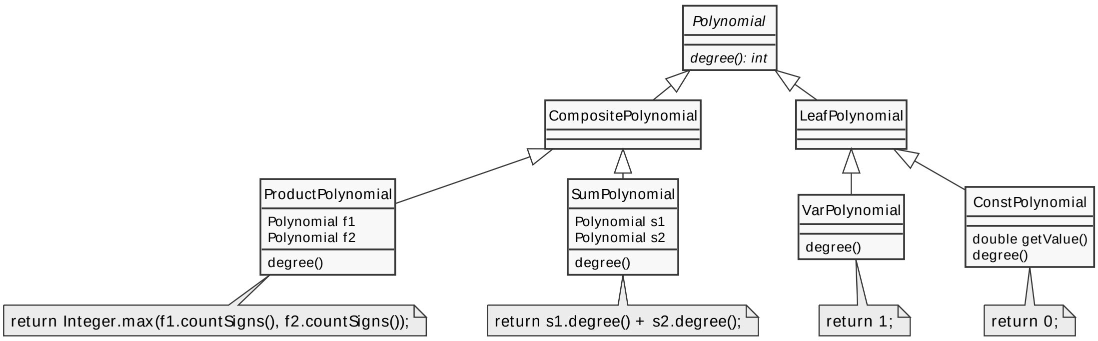
Alternatively, it is also possible to apply *Object Recursion*.

d) Which design pattern can be used to compute the degree of a polynomial?

Solution: We may apply *Object Recursion* as shown in the following figure. Alternatively, it is also possible to apply again the *Visitor* pattern.

e) Implement the function `int degree()` in the created class of a polynomial.

Solution: We implement the *Object Recursion* as follows:



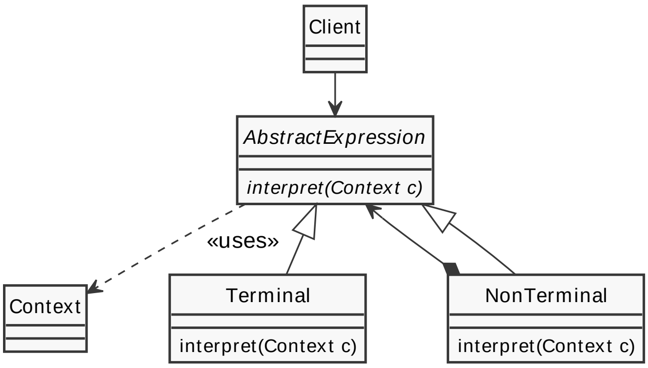
Note: In the shown data structure x^2 is encoded as $x \cdot x$, thus every VarPolynomial has degree 1. Alternatively, the degree may be an integer property of VarPolynomial.

Task 2 Secant Method of polynomials

The secant method[2] is a simple way to find a zero of a polynomial numerically. A pseudcode version can be found on the Wikipedia article. The secant method evaluates the polynomial for each iteration at a new x.

a) What is the `Interpreter` pattern? What is its structure?

Solution: The structure is as follows:



b) Implement the method `evaluate(double x)` in the class `Polynomial` of the previous task. Use the `Interpreter` pattern.

Listing 1: Implementation of Interpreter in pseudo code.

```
class Assignment {
  Assignment(double x) {
```

[2] https://en.wikipedia.org/wiki/Secant_method

```
    this.x=x;
  }
  double getX() {
    return this.x;
  }
}

class SumPolynomial {
  /* ... */
  double interpret(Assignment a) {
    return this.operand1.interpret(a) + this.operand2.interpret(a);
  }
}

class ProdPolynomial {
  /* ... */
  double interpret(Assignment a) {
    return this.operand1.interpret(a) * this.operand2.interpret(a);
  }
}

class VarPolynomial {
  /* ... */
  double interpret(Assignment a) {
    return a.getX();
  }
}

class ConstPolynomial {
  /* ... */
  double interpret(Assignment a) {
    return this.getValue();
  }
}
```

c) *optional:* Implement the secand method and test the Polynomial class and evaluation function.

d) You want to extend your class `Polynomial` to geometric functions (sinus, cosinus, tangens) of polynimals. Which design pattern can you use for the extension?

Solution: We can use the *Decorator* design pattern.

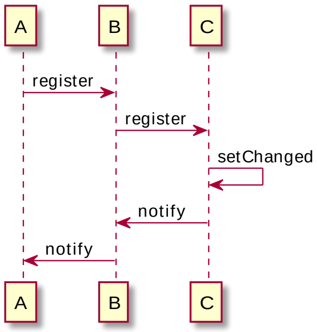
Listing 2: Implementation of Interpreter in pseudo code.

```
class SinusPolynomial extends Polynomial {
  Polynomial p;
  SinusPolynomial(Polynomial p) {
    this.p=p;
  }
  double interpret(Assignment a) {
    return Math.Sinus(this.p.interpret(a));
  }
}
```

Task 3 Chained Observer

Consider the chained variant of the *observer* design pattern with three agents A, B, and C. Consider the following case: A observers B and B observes C.

a) Draw a sequence diagram of the given scenario, where C notifies its observers.

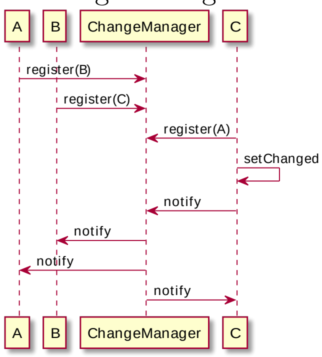
Solution: 

b) Now assume that also C observes A. Which problem occurs? How can we fix that problem?

Solution: We have cyclic observers and subjects. Thus, notification will never terminate. We can fix the problem using a *ChangeManager*. The solution is described in slide set 4, slide 49 from the lecture.

c) Draw a sequence diagram of your solution.

Solution: The change managet informas every interested participant and assure



termination.

d) In your solution, did you apply the *Mediator* design pattern?

Solution: Yes, the *ChangeManager* is a special case of the *Mediator* pattern.