Prof. Dr. Eicke Godehardt
Frankfurt University of Applied Sciences

Exercise Sheet 3

November 2nd: Composite

Exercise 1

a) Implementing the Composite Pattern:
We want to evaluate arithmetic expressions of the form $3 + 4 * 5$ etc. In order to do this, define an interfac as follows:

```
public interface ArithmeticExpr {
    Const eval();
}
```

To model constants we implement the following class:

```
class Const implements ArithmeticExpr {
    /* fields */
    private int value;

    /* constructor */
    Const(int v) {
        value = v;
    }

    /* getters */
    int getValue() {
        return value;
    }

    /* toString */
    public String toString() {
        return Integer.toString(value);
    }

    public Const eval() {
        return this;
    }
}
```

1. Implement classes `Sum` and `Prod` representing sums and products, resp. The classes should have fields `arithExpr left` and `arithExpr right` (of type `ArithmeticExpr`) to accommodate the fact that both are binary operators
2. Implement the `eval()` method as follows

   ```
   Const eval() {
       return new Const((left.eval().getValue()) + (right.eval().getValue()));
   }
   ```

   or

   ```
   Const eval() {
       return new Const((left.eval().getValue()) * (right.eval().getValue()));
   }
   ```

resp.

3. Define a few test cases

b) Implement an unary operator represented by the class `Neg` with

$$Neg(ArithExpr) := -1 * ArithExpr.$$

c) Describe how the Composite Pattern is used!

d) Extend the "Calculator" with variables:

1. The Environment `Environment` is a Hashmap which we use to lookup and put names (`env.lookup(name)` or `env.put(name, value)`. This can be used to assign variables: `ArithmeticExpr x = new Var("x")` and `env.put("x", new Const(4))` would assign x to 4.
2. Implement a class `Var` as follows

```
class Var implements ArithmeticExpr {
    /* fields */
    String name;

    /* constructor */
    Var(String n) {
        name = n;
    }

    /* toString */
    public String toString() {
        return name;
    }

    public Const eval(Environment env) {
        return env.lookup(name);
    }

}
```

(You have to change the signature of the `ArithmeticExpr` interface in order to do so!)

3. Evaluate the expression $(x + -(4 * 7))$ with $x = 4$ and $x = -34$!

Exercise 2

If you liked the previous exercise, implement an interpreter for Lisp in python! Note, that you do need to know neither Lisp nor Python as you can treat Lisp as a glorified RPN calculator and Python is pretty regular…

1. Look into `http://norvig.com/lispy.html` and try to understand it!
2. Where is the Composite Pattern?
3. Have a look at the size of the Lisp interpreter written in Java! (We will analyze the striking difference between 90 lines in Python and > 1000 lines in Java in a later unit…)

Warning: This is hard but interesting stuff!

Exercise 3

Have a look at the `headfirst.composite` packages from Head First, i.e.

1. run (and understand!) `headfirst.composite.menu.MenuTestDrive`
2. run (and understand!) `headfirst.composite.menuiterator.MenuTestDrive`
3. explain the differences (hint draw some UML diagrams)

Exercise 4

Take the previous example code and add indentation levels to menus so that they pretty-print!

Exercise 5

Have a look at the Composite Iterator in Head First, p 369. The code tries to implement a DFS (Depth First Search), however, the code does not work – find the flaw in the logic! Supply better code! (Hint: `http://www.coderanch.com/t/100049/patterns/Head-First-Design-Patterns-Composite`)

Hints

- Consult the literature!
- You can work in pairs, if you want!
- If you want to learn a Java API, look into the java docs!
- Always use the same familiar IDE (suggestion Eclipse)!