Serie 6

Content:

- Design Patterns: Decorator - Iterator - Visitor

(19) Design Pattern: Decorator - Starbuzz Coffee

Read and study Chapter 3 of [1]; solve the three proposed exercices (i.e. the “Sharpen your pencil” boxes).

- Improve the solution proposed in [1] (on page 107) for handling different sizes of beverages. One would like not to use the `if...else if...else if...` block in the method `cost()`.
- Draw the class diagram of your final solution.
- Draw a sequence diagram showing the collaborations that take place when the method `cost()` is invoked on a “Decaf - Milk - Moccha”.
- Draw a sequence diagram showing the collaborations that take place when the method `getSize()` is invoked on a “Decaf - Milk - Moccha”.

This chapter is freely available at O’Reilly [2] and a copy can also be found on [3], as well as a slightly adapted version of the source code.

(16) Design Pattern: Iterator - Traversing a two-dimensional matrix

The `Matrix` class implements a simple matrix.

One would like to access this data structure sequentially, but in two different manners.

1. Develop two iterators:
   - one accessing the data structure in “Row-Column” (`iterator.RowColumnIterator`) way,
   - and the other in the “Column-Row” (`iterator.ColumnRowIterator`) way.

   Devise the iterators in a way that other iterators can be easily added using the `Iterator` interface.
2. Add the methods related to the creation of iterators to the class `Matrix`.
   Warning: apart from the creation of the iterators, no other operations of the class `Matrix` can be modified or added.
3. Complete the `TestMatrix` program so that its execution provides the following result (for a 3x2 matrix):
   - `1-1 1-2 2-1 2-2 3-1 3-2` if the matrix provides a “Row-Column” iterator, and
   - `1-1 2-1 3-1 1-2 2-2 3-2` if the matrix provides a “Column-Row” iterator.

On [3] you can find the base code for following classes (also see Figure 1):

- the class `MatrixTest`
- the class `matrix.Matrix`
- the interface `iterator.Iterator`
- the interface `iterator.Iterable`

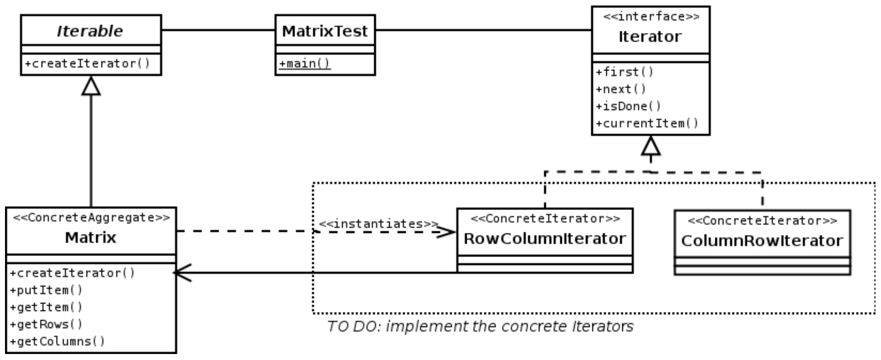

Figure 1: Class diagram for the matrix iterator.

(18) Design Pattern: Visitor - Visiting Arithmetic Expressions

In the exercise "Design Pattern: Composite - Arithmetic Expressions" we implemented the Composite design pattern. Various operations were applicable on the components (`prettyPrint()`, `prefixPrint()`, `eval()`, `depth()`).

These methods increase and "pollute" the Component interface with things that are not directly concerned with the structure. Moreover code is repeated in all classes and adding a new operation is not easy. A way to avoid this is to use the Visitor Design Pattern to encapsulate all this operations in an appropriate visitor object.

Improve your design of the preceding exercise by using the Visitor Pattern:

- On [3] you can find the base code for this exercise (see Figure 2), containing:
  - the package `visitor.syntax` containing the composite classes.
  - the interface `visitor.syntax.Expression` has changed: it now has only one `accept()` method.
  - the package `visitor.tools` with the `ExpressionVisitor` interface and one concrete visitor `PrettyPrintVisitor`.
- Write the code of the `accept()` method of `AdditionExpression`, `SubstractionExpression`, `Constant`, `DivisionExpression` and `MultiplicationExpression`.
- Implement the other visitors: `PrefixPrintVisitor`, `CalculationVisitor`, `DepthVisitor` which are equivalent to the corresponding operations of the previous exercise.
- Draw a sequence diagram illustrating what happens when the `accept()` method of an `Expression` is invoked.
- **Optional**:
  Implement a new visitor: `BuildTreeVisitor` which displays an expression as a `javax.swing.JTree`.

References

[1] Elisabeth Freeman, Eric Freeman, Bert Bates, and Kathy Sierra. *Head First Design Patterns*. O' Reilly & Associates, Inc., 2004.

[2] Elisabeth Freeman, Eric Freeman, Bert Bates, and Kathy Sierra. Head First Design Patterns, 2004. http://cdn.oreillystatic.com/oreilly/booksamplers/9780596007126_sampler.pdf (accessed Apr 26, 2016).

[3] Jacques Pasquier. Génie logiciel I, 2024. https://moodle.unifr.ch/course/view.php?id=280345 (accessed Apr 19, 2024).

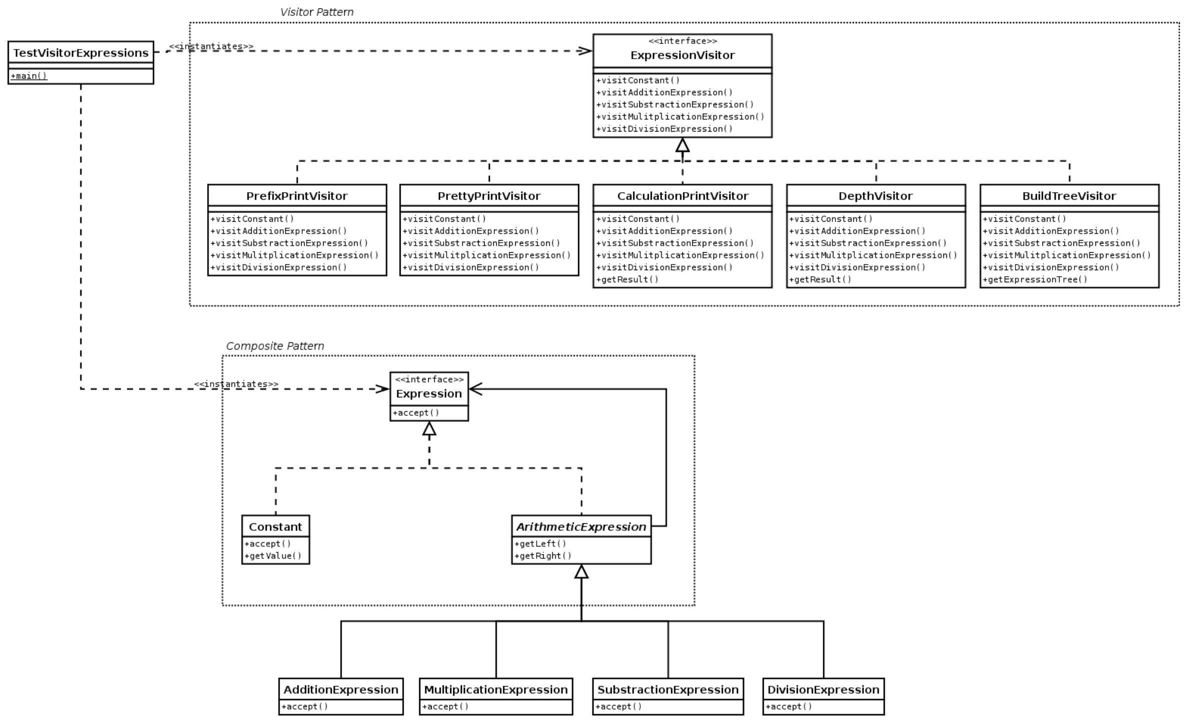

Figure 2: Class diagram for the “visitor enhanced” arithmetic expressions.