Object Orient Programming - Exercise II - LaTeX Objects

March 11, 2014

1 Introduction

In this exercise we will practice building a library of classes that will function as a simple object based interface for the LaTex language (more about LaTex later on this document).

1.1 Writing a Class Library

Up to now, in previous courses, we have mainly written code that "runs" or "executes". However, in real world you often need to provide other developers (colleagues or customers) an API (Application Programming interface) that encapsulates functionality for their own use. That is, a library that encapsulates functionality they can use in their own executables. For instance, consider the Java packages you used in the first exercise. When we imported the package *System.IO.MastermindUI* we actually imported functionality that allowed us, among other things, to write output into the console screen (*MastermindUI.printMessage()*). The object oriented principles you've learnt in class can make the process of providing encapsulated functionality easier and neater. In this exercise we will provide a hierarchy of classes that will allow its users to create LaTeX math terms. Keep in mind that when writing a class library there's no need to write a *main* function. The developer who uses the provided class library will write his own *main* function. You may, however, write a *main* function for your own debugging needs.

1.2 The LaTeX Language

The LaTeX language is a *typesetting* language. Or in other words, it's a language for creating documents. Using LaTeX it's really easy to compile styled documents without spending time on document layout or style. One of the biggest advantages of using LaTex is the comfort it provides when it comes to writing mathematical notations. For instance, all you have to do in order to write the following mathematical term -

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{1}$$

is to write the following expression - $X\_\{1,2\} = \backslash frac\{-b\backslash pm\backslash sqrt\{b \wedge 2 - 4ac\}\}\{2a\}\}$. Compiling the code using a latex compiler will result in pdf file with the equation above. For more information on LaTex you should look for one of the many online sources for it (Wikipedia is a nice place to start).

2 The Object Hierarchy

The classes you need to implemented are documented using Javadoc and their documentation is placed here. You are required to implement them as described in the documentation without changing the interface they expose (without removing any of the public methods, neither adding new ones). You are

encouraged, though, to add hidden fields or methods as you see fit.

In the following section we will give a partial description of the classes you should implement. The full information is found in the Javadoc documentation site.

The MathTerm class

This class is the base class for the entire class library described here. It defines the basic single-method interface the inheriting classes should adhere to: the **toLatex** method. Each of the classes in this class library should extend MathTerm and override it's *toLatex* method. MathTerm functions only as a base class, and should not be instantiated. Therefore, it's own implementation of the *toLatex* method is an empty one (return "";). Keep in mind, that this is not the Object Oriented way of defining a common interface. We will learn in the following lessons about *abstract classes*, and *interfaces* which will provide us with a much more elegant solution for this purpose. For detailed description of the MathTerm class, please refer to it's documentation here.

SimpleMathTerm (extends MathTerm)

This class represents a term which is a either a single letter variable (x,y,z,a,b,c..) or a number (3.14,0.5, 42). See documentation here.

BracketsMath (extends MathTerm)

Represents a math term inside brackets. i.e $(a + b)$ or (c). See documentation here.

SumMathTerm (extends MathTerm)

Represents a mathematical sum. (i.e expressions of this form:$\sum_{i=1}^{n} i^2$). This math term incorporates 3 MathTerms: The expression beneath the Sigma, above it, and the expression inside the summation. See documentation here.

BinaryMathTerm (extends MathTerm)

This subclass of MathTerm is the base class for all the MathTerm classes that incorporate exactly two other MathTerms. Classes that derive from BinaryMathTerm are the *ArithmeticOpMathTerm* class and the *FractionMathTerm* class. Since it functions as a base class, like its base class MathTerm, it shouldn't be instantiated, and it's *toLatex* method shouldn't be implemented (return "";). See documentation here.

SimpleBinaryOpMathTerm (extends BinaryMathTerm)

This class represents a MathTerm which is a composed of two other MathTerms and an operation. The operation can be one of the following: $+, -, *, <, >, =$. See documentation here.

FracMathTerm (extends BinaryMathTerm)

This class represents a fraction of two math terms. Fractions are terms of the following form - $\frac{a+1}{\sqrt{b}}$. See documentation here.

Provided Code

We provide you with the LatexDocument class, which will allow you to create Latex files by using your newly created latex class library. See LatexDocument class documentation here. In order to use this class you'd have to copy the LatexDocument.java file into your source folder directory (Remember not to submit it with your files). After creating the latex file, you may use "pdflatex" command line (which can be found on the computer lab environment, or can be downloaded as part of the MikTex or TexLive Latex environments) to create neat pdf documents out of it.

```
> pdflatex MyNewlyCreatedLatexFile
```

This may help you verify that the MathTerms you've created are rendered to valid latex terms. It is not necessary for the exercise submission, but you are strongly encouraged to use it though.

3 Examples

Consider the following examples:

Example 1

The following code builds the term $(a+2)^2$:

```
SimpleBinaryOpMathTerm internalTerm =
        new SimpleBinaryOpMathTerm(new  SimpleMathTerm("a"), new SimpleMathTerm("2"), '+');

BracketsMathTerm brTerm = new BracketsMathTerm(internalTerm);
brTerm.setExponentTerm(new SimpleMathTerm("2"));
System.out.println(brTerm.toLatex()); // Outputs: \right( a+2 \left)^{ 2 }.
```

Example 2

The following code builds the term $\frac{b}{c \cdot d}$:

```
SimpleMathTerm numerator = new SimpleMathTerm("b");
SimpleBinaryOpMathTerm denominator =
        new SimpleBinaryOpMathTerm(new SimpleMathTerm("c"), new SimpleMathTerm("d"), '*');
FractionMathTerm fracTerm = new FractionMathTerm(numerator, denominator);

System.out.println(fracTerm.toLatex()); // Outputs: \frac{ b }{ c \cdot d }
```

4 Important Remarks

- Pay attention to the fact that when a method receives a argument of type MathTerm, it means it can receive any of its subclasses as argument (remember, we defined inheritance in class as a "is-a" relation). This notion is called *Polymorphism*, and will be explained in depth in the following weeks.
- As mentioned before, more advanced notions such as *abstract classes* and *interfaces*, which will also be taught in the following weeks, will allow you to implement this library in a more elegant, object oriented way.

- In the implementation you might want to use the basckslash character "\" (like in $\backslash frac$ or $\backslash sum$). However, you should instead use a double backslash ("\\"). The character \" has a special role in the java environment (and in many others) and by using the double backslash we indicate that we're only interested in the character, and not in its functionality. Thus, instead of writing:

  `String cmd = "\frac{ " + var + " };`

  You should write:

  `String cmd = "\\frac{ " + var + " };`

- You might need to nest MathTerms inside other MathTerms. Keep in mind that every time you open a curly bracket { there should be one space afterwards. Same goes for closing curly bracket } - there must be one space preceding it.

Guidelines

- Start early.
- Read the lectures and citations material, and make sure you feel comfortable with it.
- Read a bit about LaTex. Get the general idea behind it.
- Start with the implementation of MathTerm and BinaryMathTerm base classes and only then implement their derivatives.
- Create a main method to check your code as you progress. Use the LatexDocument class to print your results into a file, and use pdflatex command line (in computer lab) to check that your latex code is valid and can rendered into pdf.
- As soon as you're done writing a class, refer to the supplied testers, and the the Unit Testing test associated with it.
- Try to use the concepts learned in class. A good sign that you should be using a different approach is when you notice code repetition.

5 School Solution

No school solution will be supplied for this exercise. However an elaborate set of testers (JUnit based) are supplied so you can test each of your classes independently. Use the testers, and the supplied code to test your code as you progress.

6 README

Please answer the following questions in the README file:

- Say you want to add a new term: an integral term. Describe in few words how you would implement it. Which class will it derive from? You can propose modifications to the class hierarchy.
- If there was any implementation difficulty, or challenge. Please mention and discuss it.

7 Testers

The testers for this exercise are given in the form of JUnit unit tests. A quick reminder of what JUnit is can be found here. This section we'll explain how to run the testers on your code.

7.1 Configuring Eclipse to work with JUnit 4.0

First you have to configure your IDE to import the JUnit library. This can be done easily by following the following steps:

- Go to Project menu.
- Go to properties, Java Build Path, Libraries, Add Library, JUnit and choose Junit 4.

7.2 Run Tests

The unit testers are found in $\sim oop/student\_testers/ex2$. Copy the tests files into your src directory. Now all you have to do to run a test is to open the test file and execute it (Ctrl+F11 in Eclipse). A windows with the test results will appear in your IDE environment.
There's a test file for every class, and tests for scenarios using the entire class library. Use the class-specific tests to test each class as you're done writing it, and use the TestScenarios.java to run full scenarios test. The LatexObjectsTest.java (a test suite) runs both specific class tests and general scenarios.

For non-Eclipse users: Either check your IDE specific details on how to run JUnit, or run the tests in an IDE-independent manner using command line.

8 Submission

Copy the class library files and the README into a new, empty directory (make sure the testers or the supplied class are not there). Create a JAR file named ex2.jar containing only the files by invoking the shell command: **jar cvf ex2.jar *.java README** The jar file should include the following files only:

- README
- BinaryMathTerm.java
- BracketsMathTerm.java
- FractionMathTerm.java
- MathTerm.java
- SimpleBinaryOpMathTerm.java
- SimpleMathTerm.java
- SumMathTerm.java

Goodluck!!