PatternsEditor

Release Report

"Green Tea"
Γκαβαρδίνας Όθωνας, ΑΜ: 2620
Μπουρλή Στυλιανή, ΑΜ: 2774

VERSIONS HISTORY

Date	Version	Description	Author
19/04/2018	0.0	1st Release	"Green Tea"
17/05/2018	1.0	2nd Release	"Green Tea"

1 Introduction

This document provides information concerning the **2nd** release of the project.

1.1 Purpose

A software development pattern defines a general reusable solution to a commonly occurring software development problem within a particular context. Patterns constitute a significant asset of the software engineering community. Amongst the very first approaches we have the GoF design patterns catalog that concerns best OO development practices. Then, there are also regular conferences (e.g. PLoP, EuroPLoP) that take place for more than 20 years and whose main topic is the identification of new patterns and pattern languages (the term pattern language is typically used to refer to a set of related patterns). Patterns are formally specified in terms of pattern templates. So far, several pattern templates have been proposed in the literature.

The main goal of this project is to develop a PatternsEditor, an application that makes pattern writting easier, especially for young inexperienced pattern writers. At a glance, PatternsEditor shall allow a patterns writer to prepare a new pattern based on well known templates change the structure of an existing pattern by switching between these templates, and generate actual pattern documents in well known formats (simple text, Latex), and so on.

1.2 Document Structure

The rest of this document is structured as follows. Section 2 specifies the acceptance tests that have been employed for this release of the project. Section 3 specifies the main design concepts for this release of the project.

2 Acceptance Tests

<For the user stories included in this releases specify below corresponding tests using a typical tabular form.>

2.1 Tests for User Story **<US1>**

Test ID	*T01*
Class	*PatternLanguage*
Test Class	*PatternTest*
Test Method	*testPatternLanguage()*

Description	*This test creates a new PatternLanguage object and also changes its name, using the method setName(). The purpose of this test is to check if the new Pattern Language exists and has the given name. The test uses a name as an input and compares it with the expected name, taken from getName() method. (We don't use a second constructor for default name, but we set the name to default when no name is given in our GUI.)*

2.2 Tests for User Story **<US2>**

Test ID	*T02, T03, T04, T05, T06*
Class	*TemplateFactory, Pattern*
Test Class	*PatternTest*
Test Method	*testClone(), testClone1(), testClone2(), testClone3(), testClone4(), testClone5()*
Description	*This test creates a TemplateFactory object, which calls the method createTemplate(). The return value of this method is stored in a PatternObject as a clone. The purpose of this test is to check if the clone object is the same with the original one. This is done by checking if the list of the clone includes all the PatternParts of the original's list. Every one of the above tests checks with this process one of the five templates (original Patterns).*

Test ID	*T07, T08*
Class	*PatternLanguage, Pattern*
Test Class	*PatternTest*
Test Method	*testList(), testList1()*
Description	*Method testList() creates a PatternLanguage object and a Pattern object. Then, it uses the add() method to insert the Pattern object in the list of PatternLanguage object. The purpose of this test is to check if after the call of add() method, the Pattern exists in PatternLanguage's list. The given input is the name of the Pattern, and we expect it to be the same as the object's name inside PatternLanguage's list.* Also, method testList1(), does the same process adding a PatternPart object in a Pattern's list.

2.3 Tests for User Story **<US3>**

Test ID	*T09, T10*

Class	*PatternLanguage, Pattern*
Test Class	*PatternTest*
Test Method	*testList2(), testList3()*
Description	*Method testList2() creates a PatternLanguage object and a Pattern object. Then, it uses the add() method to insert the Pattern object in the list of PatternLanguage object and remove() method to remove it. For this purpose, the method checks if the size of PatternLanguage's list is 1, after calling add() method, and 0 after calling remove() method.* *Also, method testList3(), does the same process adding and removing a PatternPart object from a Pattern's list.*

2.4 Tests for User Story **<US4>**

Test ID	*T11*
Class	*PatternPart*
Test Class	*PatternTest*
Test Method	*testContent()*
Description	*This test creates a PatternPart object and changes it's contents using setContents() method. The purpose here is to check if the contents of this PatternPart are set to the new value. The input given is the new value of the contents and the expecting output comes from the call of getContents() method for this PatternPart object.*

2.5 Tests for User Story **<US5>**

Test ID	*T12*
Class	*PatternComponent, PatternComposite, PatternPart*
Test Class	*SaveLoadTests*
Test Method	*saveLoadTextFileTest()*
Description	*This test creates a PatternLanguage object with 2 Patterns. Then it uses save methods to store the PatternLanguage in a text file. After these, it loads the contents of the PattenLanguage in a new PatternLanguage object. In the end, it checks if the two languages use the same name and contents.*

2.6 Tests for User Story **<US6>**

Test ID	*T12*
Class	*PatternComponent, PatternComposite, PatternPart*
Test Class	*SaveLoadTests*
Test Method	*saveLoadTextFileTest()*
Description	*This test creates a PatternLanguage object with 2 Patterns. Then it uses save methods to store the PatternLanguage in a text file. After these, it loads the contents of the PattenLanguage in a new PatternLanguage object. In the end, it checks if the two languages use the same name and contents.*

2.7 Tests for User Story **<US7>**

Test ID	*T13*
Class	*PatternComposite, Decorator,* *DecoratorAbstractFactory, LatexDecoratorFactory, Pattern*
Test Class	*SaveLoadTests*
Test Method	*decoratePatternTest()*
Description	*This test creates a Pattern object, decorates it with decorateComponents() and createPatternDecorator() methods and checks if the Pattern has been decorated.*

Test ID	*T14*
Class	*PatternComposite, Decorator,* *DecoratorAbstractFactory, LatexDecoratorFactory, PatternLanguage*
Test Class	*SaveLoadTests*
Test Method	*decoratePatternLanguageTest()*
Description	*This test creates a PatternLanguage object, decorates it with decorateComponents() and createPatternDecorator() methods and checks if the Pattern has been decorated.*

2.8 Tests for User Story **<US8>**

Test ID	*T15*

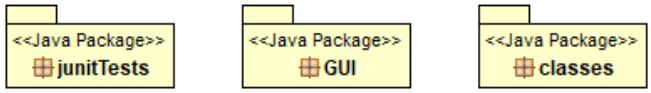
Class	*PatternComponent, PatternComposite, PatternPart,* Decorator, DecoratorAbstractFactory, LatexDecoratorFactory
Test Class	*SaveLoadLatexTest*
Test Method	*saveLoadLatexFileTest()*
Description	*This test creates a PatternLanguage object with 2 Patterns. Then it decorates the PatternLanguage and uses save methods to store the PatternLanguage in a LaTeX file. After these, it loads the contents of the PattenLanguage in a new PatternLanguage object. In the end, it checks if the two languages use the same name and contents.*

2.9 Tests for User Story **<US9>**

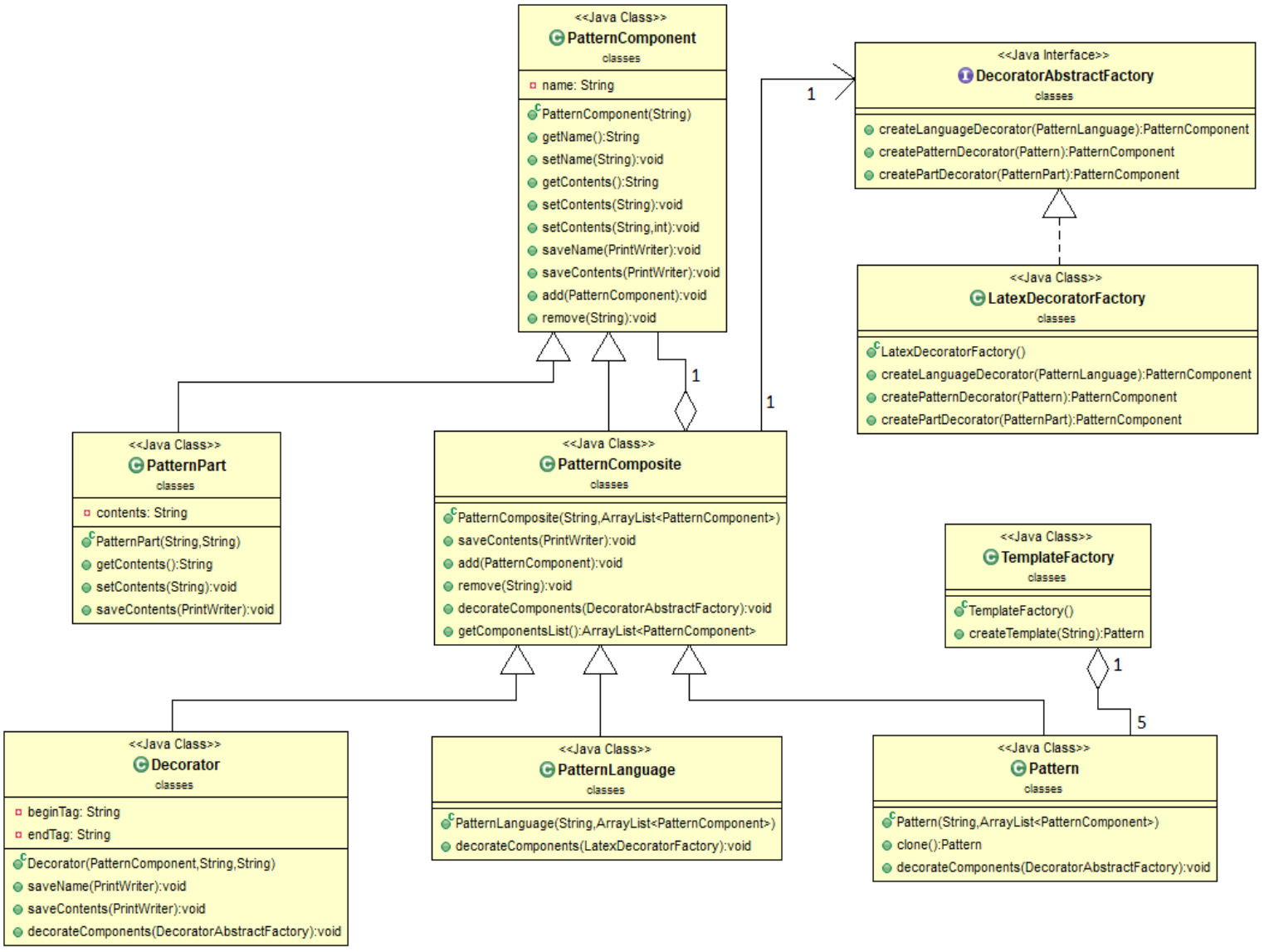
Test ID	*T15*
Class	*PatternComponent, PatternComposite, PatternPart,* Decorator, DecoratorAbstractFactory, LatexDecoratorFactory
Test Class	*SaveLoadLatexTest*
Test Method	*saveLoadLatexFileTest()*
Description	*This test creates a PatternLanguage object with 2 Patterns. Then it decorates the PatternLanguage and uses save methods to store the PatternLanguage in a LaTeX file. After these, it loads the contents of the PattenLanguage in a new PatternLanguage object. In the end, it checks if the two languages use the same name and contents.*

3 Design

3.1 Architecture



3.2 Design



Class Name: PatternComponent	
Responsibilities: ▪ Includes the name for the PatternPart, Pattern or PaternLanguage objects. ▪ Defines all the basic methods, that it's child classes implement. ▪ Includes set and get methods for its name.	**Collaborations:** ▪ PatternPart ▪ Pattern ▪ PatternLanguage ▪ PatternComposite

Class Name: PatternPart	
Responsibilities: ▪ Includes a name. ▪ Includes some contents. ▪ Includes set and get methods for its contents. ▪ Saves it's contents in a file.	**Collaborations:** ▪ PatternComponent ▪ Pattern ▪ PatternComposite

Class Name: PatternComposite	
Responsibilities: ▪ Includes a list for the PatternLanguage and Pattern objects. ▪ Implements add and remove methods for this list. ▪ Returns the list with getComponentsList method. ▪ Defines set and get methods for contents. ▪ Saves the contents of the list it containts.	**Collaborations:** ▪ PatternComponent ▪ PatternPart ▪ Pattern ▪ PatternLanguage ▪ DecoratorAbstractFactory

Class Name: PatternLanguage	
Responsibilities: ▪ Has a name.	**Collaborations:** ▪ Pattern

▪ Is a PatternComposite object. ▪ Includes a list of Pattern objects. ▪ Can decorate it's name. ▪ For each Pattern it contains, calls the decorate methods of it.	▪ PatternComposite ▪ PatternComponent ▪ DecoratorAbstractFactory ▪ LatexDecoratorFactory

Class Name: Pattern	
Responsibilities: ▪ Has a name. ▪ Is a PatternComposite object. ▪ Includes a list of PatternParts. ▪ Creates a deep copy of a Pattern template with clone method. ▪ Can decorate it's name. ▪ For each PatternPart it contains, calls the decorate methods of it.	**Collaborations:** ▪ PatternComposite ▪ PatternComponent ▪ PatternPart ▪ PatternLanguage ▪ TemplateFactory ▪ DecoratorAbstractFactory ▪ LatexDecoratorFactory

Class Name: TemplateFactory	
Responsibilities: ▪ Creates five templates in its constructor. ▪ Includes a struct (HashMap) in witch it stores the five templates. ▪ Creates a clone of a template.	**Collaborations:** ▪ Pattern ▪ PatternPart

Class Name: Decorator	
Responsibilities: ▪ Has a list with a single decorated object. ▪ Saves name and contents of a decorated object.	**Collaborations:** ▪ PatternComposite ▪ PatternLanguage ▪ Pattern

Class Name: DecoratorAbstractFactory	
Responsibilities:	**Collaborations:**

▪ Contains the methods for the DecoratorFactory	▪ LatexDecoratorFactoy ▪ PatternComposite

Class Name: LatexDecoratorFactory	
Responsibilities: ▪ Creates a Decorator object. ▪ Has methods for decorating a PatternLanguage, a Pattern or a PatternPart object.	**Collaborations:** ▪ DecoratorAbstractFactory ▪ PatternLanguage ▪ PatternsDecorator

Class Name: Menu	
Responsibilities: ▪ Contains all the GUI of the program. ▪ Gives the user some options. ▪ User can create, load or save a PatternLanguage here. ▪ User can save and load a PatternLanguage in LaTeX format here. ▪ User can add, edit or remove a Pattern in a PatternLanguage here. ▪ Displays messages if no name is given, and uses default names. ▪ Does the actual load and save processes.	**Collaborations:** ▪ PatternComponent ▪ PatternComposite ▪ PatternPart ▪ Decorator ▪ PatternLanguage ▪ Pattern ▪ TemplateFactory ▪ DecoratorAbstractFactory ▪ LatexDecoratorFactory

4 Implementation

PatternComponent.java

```java
package classes;

import java.io.FileOutputStream;

public class PatternComponent {

    private String name;

    public PatternComponent(String name) {
        this.name = name;
    }

    public String getName(){
        return name;
    }

    public void setName(String name){
        this.name = name;
    }

    public String getContents(){
        //only define here
        return "";
    }

    public void setContents(String contents){
        //only define here
    }

    public void setContents(String contents, int index) {
        //only define here
    }

    public void saveName(PrintWriter outputStream){
        outputStream.println(getName());
    }
```

....

```
38
39      public void saveContents(PrintWriter outputstream){
40          //only define here
41      }
42
43      public void add(PatternComponent patternComponent){
44          //only define here
45      }
46
47      public void remove(String patternComponentTitle){
48          //only define here
49      }
50  }
```

PatternComposite.java

```java
package classes;
import java.io.PrintWriter;

public class PatternComposite extends PatternComponent{

    private ArrayList<PatternComponent> componentsList;

    public PatternComposite(String name, ArrayList<PatternComponent> componentsList) {
        super(name);
        this.componentsList = new ArrayList<PatternComponent>();
    }

    public void saveContents(PrintWriter outputStream) {
        for(PatternComponent c : getComponentsList()) {
            c.saveName(outputStream);
            c.saveContents(outputStream);
        }
    }

    public void add(PatternComponent patternComponent) {
        componentsList.add(patternComponent);
    }

    public void remove(String patternComponentTitle) {
        for(int i = 0; i< componentsList.size(); i++) {
            if(componentsList.get(i).getName().equals(patternComponentTitle)) {
                componentsList.remove(componentsList.get(i));
            }
        }
    }

    public void decorateComponents(DecoratorAbstractFactory decoratorFactory) {
        //only define here
    }
```

....

```java
    public void decorateComponents(DecoratorAbstractFactory decoratorFactory) {
        //only define here
    }

    public ArrayList<PatternComponent> getComponentsList() {
        return componentsList;
    }
}
```

PatternPart.java

```java
package classes;

import java.io.FileOutputStream;

public class PatternPart extends PatternComponent {

    private String contents;

    public PatternPart(String name, String contents) {
        super(name);
        this.contents = contents;
    }

    public String getContents() {
        return contents;
    }

    public void setContents(String contents) {
        this.contents = contents;
    }

    public void saveContents(PrintWriter outputStream) {
        outputStream.println("\t"+getContents());
    }
}
```

PatternLanguage.java

```java
package classes;
import java.util.ArrayList;

public class PatternLanguage extends PatternComposite {

    public PatternLanguage(String name, ArrayList<PatternComponent> componentsList) {
        super(name, componentsList);
    }


    public void decorateComponents(LatexDecoratorFactory dFactory){
        for(PatternComponent c : getComponentsList()) {
            ((Pattern)c).decorateComponents(dFactory);
            dFactory.createPatternDecorator((Pattern)c);
        }
    }
}
```

Pattern.java

```java
package classes;
import java.util.ArrayList;

public class Pattern extends PatternComposite {

    public Pattern(String name, ArrayList<PatternComponent> componentsList){
        super(name,componentsList);
    }


    public Pattern clone(){
        Pattern newPattern = new Pattern("",new ArrayList<PatternComponent>());
        for(PatternComponent c : getComponentsList()) {
            String newName = c.getName();
            String newContents = c.getContents();
            newPattern.add(new PatternPart(newName, newContents));
        }
        return newPattern;
    }

    public void decorateComponents(DecoratorAbstractFactory dFactory){
        for(PatternComponent c : getComponentsList()) {
            dFactory.createPartDecorator((PatternPart)c);
        }
    }

}
```

Decorator.java

```java
package classes;

import java.util.ArrayList;

public class Decorator extends PatternComposite{

    private String beginTag;
    private String endTag;

    public Decorator(PatternComponent p, String beginTag, String endTag){
        super(p.getName(), new ArrayList<PatternComponent>());
        this.beginTag = beginTag;
        this.endTag = endTag;
        p.setName(beginTag+p.getName()+endTag);
        getComponentsList().add(p);
    }

    public void saveName(PrintWriter outputStream) {
        getComponentsList().get(0).saveName(outputStream);
    }

    public void saveContents(PrintWriter outputStream) {
        getComponentsList().get(0).saveContents(outputStream);
    }

    public void decorateComponents(DecoratorAbstractFactory decoratorFactory) {
        //only define here
    }
}
```

DecoratorAbstractFactory.java

```java
package classes;

public interface DecoratorAbstractFactory {
    public PatternComponent createLanguageDecorator(PatternLanguage language);
    public PatternComponent createPatternDecorator(Pattern pattern);
    public PatternComponent createPartDecorator(PatternPart part);
}
```

LatexDecoratorFactory.java

```java
package classes;
import java.util.ArrayList;

public class LatexDecoratorFactory implements DecoratorAbstractFactory{

    public PatternComponent createLanguageDecorator(PatternLanguage language) {
        String beginTag = "\\title{";
        String endTag = "}\n\\maketitle";
        Decorator decoratedLanguage = new Decorator(language, beginTag, endTag);
        return decoratedLanguage;
    }
    public PatternComponent createPatternDecorator(Pattern pattern) {
        String beginTag = "\\section{";
        String endTag = "}";
        Decorator decoratedPattern = new Decorator(pattern, beginTag, endTag);
        return decoratedPattern;
    }
    public PatternComponent createPartDecorator(PatternPart part) {
        String beginTag = "\t\\subsection{";
        String endTag = "}";
        Decorator decoratedPart = new Decorator(part, beginTag, endTag);
        return decoratedPart;
    }

}
```

TemplateFactory.java

```java
package classes;
import java.util.HashMap;

public class TemplateFactory {

    private HashMap<String,Pattern> templatesList;

    public TemplateFactory() {
        //create HashMap for templates
        templatesList = new HashMap<String,Pattern>();

        //define PatternParts
        PatternPart name = new PatternPart("Name","");
        PatternPart template = new PatternPart("Template","");
        PatternPart problem = new PatternPart("Problem","");
        PatternPart solution = new PatternPart("Solution","");
        PatternPart context = new PatternPart("Context","");
        PatternPart forces = new PatternPart("Forces","");
        PatternPart benefits = new PatternPart("Benefits","");
        PatternPart consequences = new PatternPart("Consequences","");
        PatternPart patternClassification = new PatternPart("Pattern Classification","");
        PatternPart intent = new PatternPart("Intent","");
        PatternPart alsoKnownAs = new PatternPart("Also known As","");
        PatternPart motivation = new PatternPart("Motivation","");
        PatternPart applicability = new PatternPart("Applicability","");
        PatternPart structure = new PatternPart("Structure","");
        PatternPart participants = new PatternPart("Participants","");
        PatternPart collaborations = new PatternPart("Collaborations","");
        PatternPart implementation = new PatternPart("Implementation","");
        PatternPart sampleCode = new PatternPart("Sample Code","");
        PatternPart knownUses = new PatternPart("Known Uses","");
        PatternPart relatedPatterns = new PatternPart("Related Patterns","");
        PatternPart example = new PatternPart("Example","");
        PatternPart dynamics = new PatternPart("Dynamics","");
        PatternPart exampleResolved = new PatternPart("Example Resolved","");
        PatternPart variants = new PatternPart("Variants","");
```

....

```
39          //create the Patterns/templates with their PatternParts
40          //micro pattern
41          Pattern template0 = new Pattern("template0", new ArrayList<PatternComponent>());
42          template0.add(name);
43          template0.add(template);
44          template0.add(problem);
45          template0.add(solution);
46          //inductive mini pattern
47          Pattern template1 = new Pattern("template1", new ArrayList<PatternComponent>());
48          template1.add(name);
49          template1.add(template);
50          template1.add(context);
51          template1.add(forces);
52          template1.add(solution);
53          //deductive mini pattern
54          Pattern template2 = new Pattern("template2", new ArrayList<PatternComponent>());
55          template2.add(name);
56          template2.add(template);
57          template2.add(problem);
58          template2.add(solution);
59          template2.add(benefits);
60          template2.add(consequences);
61          //gang of four pattern
62          Pattern template3 = new Pattern("template3", new ArrayList<PatternComponent>());
63          template3.add(name);
64          template3.add(template);
65          template3.add(patternClassification);
66          template3.add(intent);
67          template3.add(alsoKnownAs);
68          template3.add(motivation);
69          template3.add(applicability);
70          template3.add(structure);
71          template3.add(participants);
72          template3.add(collaborations);
73          template3.add(consequences);
74          template3.add(implementation);
```

....

```
        template3.add(sampleCode);
        template3.add(knownUses);
        template3.add(relatedPatterns);
        //system of patterns pattern
        Pattern template4 = new Pattern("template4", new ArrayList<PatternComponent>());
        template4.add(name);
        template4.add(template);
        template4.add(alsoKnownAs);
        template4.add(example);
        template4.add(context);
        template4.add(problem);
        template4.add(solution);
        template4.add(structure);
        template4.add(dynamics);
        template4.add(implementation);
        template4.add(exampleResolved);
        template4.add(variants);
        template4.add(knownUses);
        template4.add(consequences);

        //put the templates in HashMap
        templatesList.put("Micro-Pattern Template", template0);
        templatesList.put("Inductive Mini-Pattern", template1);
        templatesList.put("Deductive Mini-Pattern", template2);
        templatesList.put("Gang-of-Four Pattern", template3);
        templatesList.put("System of Patterns Template", template4);
    }

    public Pattern createTemplate(String templateName){

        Pattern template = templatesList.get(templateName);
        return template.clone();
    }

}
```

TESTS

PatternTest.java

```java
package junitTests;

import static org.junit.jupiter.api.Assertions.*;


class PatternTest {

    @Test //T01, testing getName and setName in PatternLanguage object
    void testPatternLanguage() {
        PatternLanguage newPatternLanguage = new PatternLanguage("", new ArrayList<PatternComponent>());
        newPatternLanguage.setName("NewName");
        assertEquals("NewName", newPatternLanguage.getName());
    }

    @Test //T02, testing createTemplate and clone methods for Micro-Pattern Template
    void testClone() {
        Pattern newPattern = new Pattern("", new ArrayList<PatternComponent>());
        TemplateFactory temp = new TemplateFactory();
        newPattern = temp.createTemplate("Micro-Pattern Template");

        assertEquals(4, newPattern.getComponentsList().size());

        assertEquals("Name", newPattern.getComponentsList().get(0).getName());
        assertEquals("Template", newPattern.getComponentsList().get(1).getName());
        assertEquals("Problem", newPattern.getComponentsList().get(2).getName());
        assertEquals("Solution", newPattern.getComponentsList().get(3).getName());
    }

    @Test //T03, testing createTemplate and clone methods for Inductive Mini-Pattern
    void testClone2() {
        Pattern newPattern = new Pattern("", new ArrayList<PatternComponent>());
        TemplateFactory temp = new TemplateFactory();
        newPattern = temp.createTemplate("Inductive Mini-Pattern");

        assertEquals(5, newPattern.getComponentsList().size());
```

....

```
44
45          assertEquals("Name", newPattern.getComponentsList().get(0).getName());
46          assertEquals("Template", newPattern.getComponentsList().get(1).getName());
47          assertEquals("Context", newPattern.getComponentsList().get(2).getName());
48          assertEquals("Forces", newPattern.getComponentsList().get(3).getName());
49          assertEquals("Solution", newPattern.getComponentsList().get(4).getName());
50      }
51
52      @Test //T04, testing createTemplate and clone methods for Deductive Mini-Pattern
53      void testClone3() {
54          Pattern newPattern = new Pattern("", new ArrayList<PatternComponent>());
55          TemplateFactory temp = new TemplateFactory();
56          newPattern = temp.createTemplate("Deductive Mini-Pattern");
57
58          assertEquals(6, newPattern.getComponentsList().size());
59
60          assertEquals("Name", newPattern.getComponentsList().get(0).getName());
61          assertEquals("Template", newPattern.getComponentsList().get(1).getName());
62          assertEquals("Problem", newPattern.getComponentsList().get(2).getName());
63          assertEquals("Solution", newPattern.getComponentsList().get(3).getName());
64          assertEquals("Benefits", newPattern.getComponentsList().get(4).getName());
65          assertEquals("Consequences", newPattern.getComponentsList().get(5).getName());
66      }
67
68      @Test //T05, testing createTemplate and clone methods for Gang-of-Four Pattern
69      void testClone4() {
70          Pattern newPattern = new Pattern("", new ArrayList<PatternComponent>());
71          TemplateFactory temp = new TemplateFactory();
72          newPattern = temp.createTemplate("Gang-of-Four Pattern");
73
74          assertEquals(15, newPattern.getComponentsList().size());
75
76          assertEquals("Name", newPattern.getComponentsList().get(0).getName());
77          assertEquals("Template", newPattern.getComponentsList().get(1).getName());
78          assertEquals("Pattern Classification", newPattern.getComponentsList().get(2).getName());
79          assertEquals("Intent", newPattern.getComponentsList().get(3).getName());
```

....

```
80          assertEquals("Also known As", newPattern.getComponentsList().get(4).getName());
81          assertEquals("Motivation", newPattern.getComponentsList().get(5).getName());
82          assertEquals("Applicability", newPattern.getComponentsList().get(6).getName());
83          assertEquals("Structure", newPattern.getComponentsList().get(7).getName());
84          assertEquals("Participants", newPattern.getComponentsList().get(8).getName());
85          assertEquals("Collaborations", newPattern.getComponentsList().get(9).getName());
86          assertEquals("Consequences", newPattern.getComponentsList().get(10).getName());
87          assertEquals("Implementation", newPattern.getComponentsList().get(11).getName());
88          assertEquals("Sample Code", newPattern.getComponentsList().get(12).getName());
89          assertEquals("Known Uses", newPattern.getComponentsList().get(13).getName());
90          assertEquals("Related Patterns", newPattern.getComponentsList().get(14).getName());
91      }
92
93      @Test //T06, testing createTemplate and clone methods for System of Patterns Template
94      void testClone5() {
95          Pattern newPattern = new Pattern("", new ArrayList<PatternComponent>());
96          TemplateFactory temp = new TemplateFactory();
97          newPattern = temp.createTemplate("System of Patterns Template");
98
99          assertEquals(14, newPattern.getComponentsList().size());
100
101         assertEquals("Name", newPattern.getComponentsList().get(0).getName());
102         assertEquals("Template", newPattern.getComponentsList().get(1).getName());
103         assertEquals("Also known As", newPattern.getComponentsList().get(2).getName());
104         assertEquals("Example", newPattern.getComponentsList().get(3).getName());
105         assertEquals("Context", newPattern.getComponentsList().get(4).getName());
106         assertEquals("Problem", newPattern.getComponentsList().get(5).getName());
107         assertEquals("Solution", newPattern.getComponentsList().get(6).getName());
108         assertEquals("Structure", newPattern.getComponentsList().get(7).getName());
109         assertEquals("Dynamics", newPattern.getComponentsList().get(8).getName());
110         assertEquals("Implementation", newPattern.getComponentsList().get(9).getName());
111         assertEquals("Example Resolved", newPattern.getComponentsList().get(10).getName());
112         assertEquals("Variants", newPattern.getComponentsList().get(11).getName());
113         assertEquals("Known Uses", newPattern.getComponentsList().get(12).getName());
114         assertEquals("Consequences", newPattern.getComponentsList().get(13).getName());
```

....

```
115     }
116
117     @Test //T07, testing list, add method in Pattern object
118     void testList() {
119         PatternLanguage newPatternLanguage = new PatternLanguage("", new ArrayList<PatternComponent>());
120         Pattern newPattern = new Pattern("name", new ArrayList<PatternComponent>());
121         newPatternLanguage.add(newPattern);
122         assertEquals("name", newPatternLanguage.getComponentsList().get(0).getName());
123     }
124
125     @Test //T08, testing list, add method in Pattern object
126     void testList1() {
127         Pattern newPattern = new Pattern("", new ArrayList<PatternComponent>());
128         PatternPart newPatternPart = new PatternPart("name", "contents");
129         newPattern.add(newPatternPart);
130         assertEquals("name", newPattern.getComponentsList().get(0).getName());
131         assertEquals("contents", newPattern.getComponentsList().get(0).getContents());
132     }
133
134     @Test //T09, testing list, remove method in PatternLanguage object
135     void testList2() {
136         PatternLanguage newPatternLanguage = new PatternLanguage("", new ArrayList<PatternComponent>());
137         Pattern newPattern = new Pattern("name", new ArrayList<PatternComponent>());
138         newPatternLanguage.add(newPattern);
139         assertEquals(1, newPatternLanguage.getComponentsList().size());
140         newPatternLanguage.remove("name");
141         assertEquals(0, newPatternLanguage.getComponentsList().size());
142     }
143
144     @Test //T10, testing list, remove method Pattern object
145     void testList3() {
146         Pattern newPattern = new Pattern("", new ArrayList<PatternComponent>());
147         PatternPart newPatternPart = new PatternPart("name", "contents");
148         newPattern.add(newPatternPart);
149         assertEquals(1, newPattern.getComponentsList().size());
150         newPattern.remove("name");
```

....

```
151         assertEquals(0, newPattern.getComponentsList().size());
152     }
153
154     @Test //T11, testing getContents and setContents methods in PatternPart object
155     void testContent() {
156         PatternPart newPatternPart = new PatternPart("name", "contents");
157         assertEquals("contents", newPatternPart.getContents());
158         newPatternPart.setContents("newContents");
159         assertEquals("newContents", newPatternPart.getContents());
160     }
161 }
```

SaveLoadTests.java

```java
package junitTests;

import static org.junit.jupiter.api.Assertions.*;

class SaveLoadTests {

    @Test //T12, testing saving PatternLanguage in text File and load it
    void saveLoadTextFileTest() {
        //create PatternLanguage
        PatternLanguage TextLanguage = new PatternLanguage("TextLanguage", new ArrayList<PatternComponent>());
        //add a micro Pattern to it
        TemplateFactory micro = new TemplateFactory();
        Pattern firstPattern = micro.createTemplate("Micro-Pattern Template");
        for(int i = 0; i < firstPattern.getComponentsList().size(); i++) {
            if (firstPattern.getComponentsList().get(i).getName().equals("Name")) {
                firstPattern.getComponentsList().get(i).setContents("A");
            }
            else if(firstPattern.getComponentsList().get(i).getName().equals("Template")) {
                firstPattern.getComponentsList().get(i).setContents("b");
            }
            else if(firstPattern.getComponentsList().get(i).getName().equals("Problem")) {
                firstPattern.getComponentsList().get(i).setContents("c");
            }
            else if(firstPattern.getComponentsList().get(i).getName().equals("Solution")) {
                firstPattern.getComponentsList().get(i).setContents("d");
            }
        }

        firstPattern.setName("A");
        TextLanguage.add(firstPattern);

        //add a mini Pattern to it
        TemplateFactory mini = new TemplateFactory();
        Pattern secondPattern = mini.createTemplate("Inductive Mini-Pattern");
        for(int i = 0; i < secondPattern.getComponentsList().size(); i++) {
            if (secondPattern.getComponentsList().get(i).getName().equals("Name")) {
```

....

```
                secondPattern.getComponentsList().get(i).setContents("B");
            }
            else if(secondPattern.getComponentsList().get(i).getName().equals("Template")) {
                secondPattern.getComponentsList().get(i).setContents("g");
            }
            else if(secondPattern.getComponentsList().get(i).getName().equals("Context")) {
                secondPattern.getComponentsList().get(i).setContents("h");
            }
            else if(secondPattern.getComponentsList().get(i).getName().equals("Forces")) {
                secondPattern.getComponentsList().get(i).setContents("f");
            }
            else if(secondPattern.getComponentsList().get(i).getName().equals("Solution")) {
                secondPattern.getComponentsList().get(i).setContents("u");
            }
        }
        secondPattern.setName("B");
        TextLanguage.add(secondPattern);

        //save to file
        FileOutputStream outputStream = null;
        try
        {
            outputStream = new FileOutputStream(TextLanguage.getName()+".txt");
        }
        catch(FileNotFoundException e)
        {
            System.out.println("Error opening the file "+TextLanguage.getName()+".txt."); //open file wi
            System.exit(0);
        }
        PrintWriter outputWriter = new PrintWriter(outputStream);
        TextLanguage.saveName(outputWriter); //call saveName method for the PatternLanguage
        TextLanguage.saveContents(outputWriter); //call saveContents method for the PatternLanguage
        outputWriter.close( ); //close file

        //load from file
        PatternLanguage myPatternLanguage = new PatternLanguage("", new ArrayList<PatternComponent>());
```

....

```
91          Scanner inputStream = null;
92          String fileName =  "TextLanguage";
93          try
94          {
95              inputStream = new Scanner(new FileInputStream(fileName+".txt")); //open File to load PatternLan
96          }
97          catch(FileNotFoundException ex)
98          {
99              System.out.println("File "+ fileName+".txt was not found");
100             System.out.println("or could not be opened.");
101             System.exit(0);
102         }
103
104         HashSet<String> mySet = new HashSet<String>(); //HashSet with all PatternPart's Names
105         mySet.add("Template");
106         mySet.add("Problem");
107         mySet.add("Solution");
108         mySet.add("Context");
109         mySet.add("Forces");
110         mySet.add("Benefits");
111         mySet.add("Consequences");
112         mySet.add("Pattern Classification");
113         mySet.add("Intent");
114         mySet.add("Also known As");
115         mySet.add("Motivation");
116         mySet.add("Applicability");
117         mySet.add("Structure");
118         mySet.add("Participants");
119         mySet.add("Collaborations");
120         mySet.add("Implementation");
121         mySet.add("Sample Code");
122         mySet.add("Known Uses");
123         mySet.add("Related Patterns");
124         mySet.add("Example");
125         mySet.add("Dynamics");
126         mySet.add("Example Resolved");
```

....

```
        mySet.add("Variants");

        String PatternLanguageName = inputStream.nextLine( ); //load PatternLanguage's Name
        myPatternLanguage.setName(PatternLanguageName); //the new PatternLanguage has the loaded Name
        int store = 0; //flag
        ArrayList<PatternComponent> Parts =  new ArrayList<PatternComponent>();  //store PatternParts from loading in this ArrayList
        ArrayList<PatternComponent> Patterns =  new ArrayList<PatternComponent>(); //store Patterns from loading in this ArrayList
        while (inputStream.hasNextLine( )){ //while it is not end of file
            String line = inputStream.nextLine( ); //read line from file
            if(line.equals("Name")) //if line has Name as content
            {
                if(store == 0) //if it's the first line with Name as content
                {
                    Pattern loadPattern = new Pattern("", new ArrayList<PatternComponent>()); //create new Pattern
                    Patterns.add(loadPattern); //add it to Pattern's ArrayList
                    PatternPart loadPart = new PatternPart("",""); //create new PatternPart
                    String loadName = inputStream.nextLine( ); //next line has the name value
                    Patterns.get(0).setName(""+loadName.charAt(1)); //the Pattern has the loaded Name
                    loadPart.setName("Name"); //the PatternPart with the Name "Name" has also this name
                    loadPart.setContents(""+loadName.charAt(1)); //So, the PatternPart with Name "Name" has as contents the loaded Name
                    Parts.add(loadPart); //add it to PatternPart's ArrayList
                    store = 1; //change flag's value
                }
                else //if it's not the first line with Name as content
                {
                    for(PatternComponent p : Parts) { //read now a New Pattern so add all the PatternPart's we had to the previous Pattern
                        Patterns.get(0).add(p);
                    }
                    Parts.clear(); //delete the previous PatternParts, so Parts ArrayList is empty
                    myPatternLanguage.add(Patterns.get(0)); //add the previous Pattern to PatternLanguage
                    Patterns.remove(0); //delete it
                    Pattern loadPattern = new Pattern("", new ArrayList<PatternComponent>()); //create new Pattern
                    Patterns.add(loadPattern); //add it to Pattern's ArrayList
                    PatternPart loadPart = new PatternPart("",""); //create new PatternPart
                    String loadName = inputStream.nextLine( ); //next line has the name value
                    Patterns.get(0).setName(""+loadName.charAt(1));  //the Pattern has the loaded Name
```

....

```
                loadPart.setName("Name"); //the PatternPart with the Name "Name" has also this name
                loadPart.setContents(""+loadName.charAt(1)); //So, the PatternPart with Name "Name" has as contents the loaded Name
                Parts.add(loadPart); //add it to PatternPart's ArrayList
            }

        }
        else if(mySet.contains(line)) //if line has as content a PatternPart's Name
        {
            PatternPart loadPart = new PatternPart("",""); //create new PatternPart
            loadPart.setName(line); //PatternPart has the loaded Name
            String loadContent = inputStream.nextLine( ); //next line has the contents for the PatternPart
            loadPart.setContents(""+loadContent.charAt(1)); //PatternPart has the loaded contents
            Parts.add(loadPart); //add the PatternPart to PatternPart's ArrayList
        }
        else
        {
            continue;
        }
    }
    //for the last Pattern
    for(PatternComponent p : Parts) { //add all the PatternPart's we had to the last Pattern
        Patterns.get(0).add(p);
    }
    Parts.clear(); //delete the last PatternParts, so Parts ArrayList is empty
    myPatternLanguage.add(Patterns.get(0)); //add the last Pattern to PatternLanguage
    Patterns.remove(0); //delete it
    inputStream.close(); //close file

    assertEquals(TextLanguage.getName(), myPatternLanguage.getName()); //check if PatternLanguages have same name

    for(int i=0;i<TextLanguage.getComponentsList().size();i++)
    {
        assertEquals(TextLanguage.getComponentsList().get(i).getName(), myPatternLanguage.getComponentsList().get(i).getName()); //check if p

        Pattern p  = (Pattern)TextLanguage.getComponentsList().get(i);
        Pattern p2 = (Pattern)myPatternLanguage.getComponentsList().get(i);
```

....

```
            int size = p.getComponentsList().size();
            for(int j=0;j<size;j++)
            {
                assertEquals(p.getComponentsList().get(j).getName(), p2.getComponentsList().get(j).getName()); //check if parts have same name
                assertEquals(p.getComponentsList().get(j).getContents(), p2.getComponentsList().get(j).getContents()); //check if parts have same
            }
        }
    }

    @Test //T13, testing decorateComponents and createPatternDecorator for Pattern object
    void decoratePatternTest() {
        Pattern myPattern = new Pattern("Name", new ArrayList<PatternComponent>());
        LatexDecoratorFactory dFactory = new LatexDecoratorFactory();
        myPattern.decorateComponents(dFactory);
        Decorator decoratedPattern = (Decorator)(dFactory.createPatternDecorator(myPattern));
        assertEquals("\\section{Name}", decoratedPattern.getComponentsList().get(0).getName());
    }

    @Test //T14, testing decorateComponents and createLanguageDecorator for PatternLanguage object
    void decoratePatternLanguageTest() {
        PatternLanguage myPatternLanguage = new PatternLanguage("Name", new ArrayList<PatternComponent>());
        LatexDecoratorFactory dFactory = new LatexDecoratorFactory();
        myPatternLanguage.decorateComponents(dFactory);
        Decorator decoratedPatternLanguage = (Decorator)(dFactory.createLanguageDecorator(myPatternLanguage));
        assertEquals("\\title{Name}\n\\maketitle", decoratedPatternLanguage.getComponentsList().get(0).getName());
    }
}
```

SaveLoadLatexTest.java

```
package junitTests;
import static org.junit.jupiter.api.Assertions.*;

class SaveLoadLatexTest {

    @Test //T15, testing saving PatternLanguage in LaTeX File and load it
    void saveLoadLatexFileTest(){
        //create new PatternLanguage
        PatternLanguage LatexLanguage = new PatternLanguage("LatexLanguage", new ArrayList<PatternComponent>());
        //add a micro Pattern to it
        TemplateFactory micro = new TemplateFactory();
        Pattern firstPattern = micro.createTemplate("Micro-Pattern Template");
        for(int i = 0; i < firstPattern.getComponentsList().size(); i++) {
            if (firstPattern.getComponentsList().get(i).getName().equals("Name")) {
                firstPattern.getComponentsList().get(i).setContents("A");
            }
            else if(firstPattern.getComponentsList().get(i).getName().equals("Template")) {
                firstPattern.getComponentsList().get(i).setContents("b");
            }
            else if(firstPattern.getComponentsList().get(i).getName().equals("Problem")) {
                firstPattern.getComponentsList().get(i).setContents("c");
            }
            else if(firstPattern.getComponentsList().get(i).getName().equals("Solution")) {
                firstPattern.getComponentsList().get(i).setContents("d");
            }
        }

        firstPattern.setName("A");
        LatexLanguage.add(firstPattern);

        //add a mini Pattern to it
        TemplateFactory mini = new TemplateFactory();
        Pattern secondPattern = mini.createTemplate("Inductive Mini-Pattern");
        for(int i = 0; i < secondPattern.getComponentsList().size(); i++) {
            if (secondPattern.getComponentsList().get(i).getName().equals("Name")) {
                secondPattern.getComponentsList().get(i).setContents("B");
```

....

```
54              }
55              else if(secondPattern.getComponentsList().get(i).getName().equals("Template")) {
56                  secondPattern.getComponentsList().get(i).setContents("g");
57              }
58              else if(secondPattern.getComponentsList().get(i).getName().equals("Context")) {
59                  secondPattern.getComponentsList().get(i).setContents("h");
60              }
61              else if(secondPattern.getComponentsList().get(i).getName().equals("Forces")) {
62                  secondPattern.getComponentsList().get(i).setContents("f");
63              }
64              else if(secondPattern.getComponentsList().get(i).getName().equals("Solution")) {
65                  secondPattern.getComponentsList().get(i).setContents("u");
66              }
67          }
68          secondPattern.setName("B");
69          LatexLanguage.add(secondPattern);
70
71          //save PatternLanguage in a LaTeX file
72          String fileName = LatexLanguage.getName();
73          LatexDecoratorFactory dFactory = new LatexDecoratorFactory();
74          LatexLanguage.decorateComponents(dFactory); //decorate the PatternLanguage
75          Decorator decoratedLanguage = (Decorator)(dFactory.createLanguageDecorator(LatexLanguage));
76
77          FileOutputStream outputStream = null;
78          try
79          {
80              outputStream = new FileOutputStream(fileName+".tex"); //open file with PatternLanguage's Name f
81          }
82          catch(FileNotFoundException e)
83          {
84              System.out.println("Error opening the file "+fileName+".tex.");
85              System.exit(0);
86          }
87          PrintWriter outputWriter = new PrintWriter(outputStream);
88          outputWriter.write("\\documentclass{article}\n"); //write start labels for LaTeX file
89          outputWriter.write("\\begin{document}\n");
```

.....

```
90          decoratedLanguage.saveName(outputWriter); //call saveName method for the decorated PatternLanguage
91          decoratedLanguage.saveContents(outputWriter); //call saveContents method for the decorated PatternLanguage
92          outputWriter.write("\\end{document}"); //write end label for LaTeX file
93          outputWriter.close( ); //close file
94
95          //load PatternLanguage from LaTeX file
96          Scanner inputStream = null;
97          String fileName2 =  "LatexLanguage";
98          try
99          {
100             inputStream = new Scanner(new FileInputStream(fileName2+".tex")); //open file for loading
101         }
102         catch(FileNotFoundException ex)
103         {
104             System.out.println("File "+ fileName2+".tex was not found");
105             System.out.println("or could not be opened.");
106             System.exit(0);
107         }
108
109         inputStream.nextLine(); //1st Line nothing
110         inputStream.nextLine(); //2nd Line nothing
111         String title = inputStream.nextLine(); //3rd line has the Name for PatternLanguage
112         String PatternLanguageName = "";
113         for(int i = 7;i<title.length()-1;i++) {
114             PatternLanguageName += title.charAt(i); //load the name for PatternLanguage
115         }
116         PatternLanguage newPatternLanguage = new PatternLanguage("", new ArrayList<PatternComponent>()); //create a new PatternL
117         newPatternLanguage.setName(PatternLanguageName); //the new PatternLanguage has the loaded Name
118         inputStream.nextLine(); //4th line nothing
119         int store = 0; //flag
120         ArrayList<PatternComponent> Parts =  new ArrayList<PatternComponent>(); //store PatternParts from loading in this ArrayL
121         ArrayList<PatternComponent> Patterns =  new ArrayList<PatternComponent>(); //store Patterns from loading in this ArrayLi
122         while (inputStream.hasNextLine( )){ //while it is not end of file
123             String line = inputStream.nextLine( ); //read line from file
124             //check if line is for Pattern or for PatternPart or end of file
125             String sec = ""; //for checking section label
```

....

```
126             String sub = ""; //for checking subsection label
127             String end = ""+ line.charAt(1) + line.charAt(2) + line.charAt(3); //for checking end of file label
128             if (((""+line.charAt(0)).equals("\\") || line.length()>6)
129             {
130                 for(int i=2;i<5;i++) {
131                     sub += line.charAt(i);
132                 }
133 
134                 for(int i=1;i<8;i++) {
135                     sec += line.charAt(i);
136                 }
137             }
138 
139             if(sec.equals("section")) { //if it is section label
140                 if(store == 0){ //if it's the first section we read
141                     Pattern loadPattern = new Pattern("", new ArrayList<PatternComponent>()); //create new Pattern
142                     Patterns.add(loadPattern); //add it to Pattern's ArrayList
143                     PatternPart loadPart = new PatternPart("",""); //create new PatternPart
144                     String loadName = ""; //line with section label has the name for the Pattern
145                     for(int i=9;i<line.length()-1;i++) {
146                         loadName += line.charAt(i);
147                     }
148                     Patterns.get(0).setName(loadName); //the Pattern has the loaded Name
149                     loadPart.setName("Name"); //the PatternPart with the Name "Name" has also this name
150                     loadPart.setContents(loadName); //So, the PatternPart with Name "Name" has as contents the loaded Name
151                     Parts.add(loadPart); //add it to PatternPart's ArrayList
152                     store = 1; //change flag's value
153                     inputStream.nextLine(); //do nothing
154                     inputStream.nextLine(); //do nothing
155                 }
156                 else { //if it's not the first section we read
157                     for(PatternComponent p : Parts) { //read now a New Pattern so add all the PatternPart's we had to the previous Pattern
158                         Patterns.get(0).add(p);
159                     }
160                     Parts.clear(); //delete the previous PatternParts, so Parts ArrayList is empty
161                     newPatternLanguage.add(Patterns.get(0)); //add the previous Pattern to PatternLanguage
```

....

```
                Patterns.remove(0); //delete it
                Pattern loadPattern = new Pattern("", new ArrayList<PatternComponent>()); //create new Pattern
                Patterns.add(loadPattern); //add it to Pattern's ArrayList
                PatternPart loadPart = new PatternPart("",""); //create new PatternPart
                String loadName = ""; //line with section label has the name for the Pattern
                for(int i=9;i<line.length()-1;i++)
                {
                    loadName += line.charAt(i);
                }
                Patterns.get(0).setName(loadName); //the Pattern has the loaded Name
                loadPart.setName("Name"); //the PatternPart with the Name "Name" has also this name
                loadPart.setContents(loadName); //So, the PatternPart with Name "Name" has as contents the loaded Name
                Parts.add(loadPart); //add it to PatternPart's ArrayList
                inputStream.nextLine(); //do nothing
                inputStream.nextLine(); //do nothing
            }
        }
        else if(sub.equals("sub")) {  //if it is subsection label
            PatternPart loadPart = new PatternPart("",""); //create new PatternPart
            String partName = ""; //load from this line it's Name
            for(int i=13;i<line.length()-1;i++)
            {
                partName += line.charAt(i);
            }
            loadPart.setName(partName); //PatternPart has the loaded Name
            String loadContentLine = inputStream.nextLine( ); //next line has the contents for the PatternPart
            String loadContent = ""; //load from this line the contents

            for(int i=1;i<loadContentLine.length();i++)
            {
                loadContent += loadContentLine.charAt(i);
            }
            loadPart.setContents(loadContent); //PatternPart has the loaded contents
            Parts.add(loadPart); //add the PatternPart to PatternPart's ArrayList
        }
        else if(end.equals("end")){ //if it is end label
```

....

```
198                 for(PatternComponent p : Parts) { //add all the PatternPart's we had to the last Pattern
199                     Patterns.get(0).add(p);
200                 }
201                 Parts.clear(); //delete the last PatternParts, so Parts ArrayList is empty
202                 newPatternLanguage.add(Patterns.get(0)); //add the last Pattern to PatternLanguage
203                 Patterns.remove(0); //delete it
204             }
205         }
206         inputStream.close(); //close file
207 
208         //check if PatternLanguages have same name
209         assertEquals("LatexLanguage", newPatternLanguage.getName());
210 
211         //check if patterns have same name
212         assertEquals("A", newPatternLanguage.getComponentsList().get(0).getName());
213         assertEquals("B", newPatternLanguage.getComponentsList().get(1).getName());
214 
215         Pattern A = (Pattern)newPatternLanguage.getComponentsList().get(0);
216         Pattern B = (Pattern)newPatternLanguage.getComponentsList().get(1);
217 
218         //check if parts in first Pattern have same name
219         assertEquals("Name", A.getComponentsList().get(0).getName());
220         assertEquals("Template", A.getComponentsList().get(1).getName());
221         assertEquals("Problem", A.getComponentsList().get(2).getName());
222         assertEquals("Solution", A.getComponentsList().get(3).getName());
223 
224         //check if parts in second Pattern have same name
225         assertEquals("Name", B.getComponentsList().get(0).getName());
226         assertEquals("Template", B.getComponentsList().get(1).getName());
227         assertEquals("Context", B.getComponentsList().get(2).getName());
228         assertEquals("Forces", B.getComponentsList().get(3).getName());
229         assertEquals("Solution", B.getComponentsList().get(4).getName());
230 
231         //check if parts in first Pattern have same contents
232         assertEquals("A", A.getComponentsList().get(0).getContents());
233         assertEquals("b", A.getComponentsList().get(1).getContents());
```

....

```
234         assertEquals("c", A.getComponentsList().get(2).getContents());
235         assertEquals("d", A.getComponentsList().get(3).getContents());
236 
237         //check if parts in second Pattern have same contents
238         assertEquals("B", B.getComponentsList().get(0).getContents());
239         assertEquals("g", B.getComponentsList().get(1).getContents());
240         assertEquals("h", B.getComponentsList().get(2).getContents());
241         assertEquals("f", B.getComponentsList().get(3).getContents());
242         assertEquals("u", B.getComponentsList().get(4).getContents());
243     }
244 }
```