

**UNIVERSITY OF HERTFORDSHIRE**

**School of Computer Science**

**Modular BSc Honours in Computer Science**

**6COM1056 - Software Engineering Project**

**Final Report**

**April 2017**

**TITLE OF PROJECT**

**UML Class Diagram Creation and Code Generation Tool**

**Author's initials and surname**

**RQ Quazi**

**Supervised by: Peter Lane**

## **ABSTRACT**

The purpose of this report is to detail the work that was undertaken in developing a system that allows the creation of UML Class diagrams and also the ability to convert these diagrams into code. The report will go through the planning, designing, implementation and testing that was done in order to achieve this goal. The challenges faced throughout development will also be discussed.

This system was developed in order to allow software developers to increase productivity by reducing the gap between design and development.

## **ACKNOWLEDGEMENTS**

I would like to thank my project supervisor Dr. Peter Lane for his support throughout my project.

# CONTENTS

<b>1 - INTRODUCTION .....</b>	<b>6</b>
1.1 - OVERVIEW OF PROJECT .....	6
1.2 - MOTIVATION.....	6
1.3 - AIMS AND OBJECTIVES .....	6
1.3.1 - MAIN OBJECTIVES .....	6
1.3.2 - DESIRABLE OBJECTIVES.....	6
1.4 - REPORT STRUCTURE .....	7
1.5 - LITERATURE REVIEW .....	7
1.5.1 - MODELLING LANGUAGES.....	7
1.5.2 - WHY UML?.....	8
1.5.3 - UML CLASS DIAGRAMS.....	8
1.5.4 - EXISTING SOFTWARE.....	9
1.5.5 – DESIGN PATTERNS .....	9
1.5.6 - VERSION CONTROL SYSTEMS .....	10
1.6 – GANTT CHART.....	12
<b>2 - PLANNING AND DESIGN .....</b>	<b>13</b>
2.1 USE CASE DIAGRAM .....	13
2.2 - USE CASE DESCRIPTIONS .....	13
2.3 - CLASS DIAGRAM .....	18
2.4 - STORYBOARD.....	19
<b>3 - DEVELOPMENT AND IMPLEMENTATION .....</b>	<b>20</b>
3.1 SETTING UP .....	20
3.2 CLASS STRUCTURE .....	21
3.3 CREATING THE GUI .....	23
3.3.1 WHY SWING? .....	23
3.3.2 CHOOSING THE ROOT PANE CONTAINER .....	23
3.4 ADDING CLASSES .....	24
3.4.1 ADDING AND REMOVING VARIABLES AND METHODS .....	25
3.4.2 EDITING VARIABLES AND METHODS.....	26
3.5 CREATING THE DIAGRAM .....	26

3.5.1 ALLOWING FOR LARGE DIAGRAMS .....	26
3.5.2 CLASS BOX SIZE .....	27
3.6 - CONVERTING TO CODE.....	28
<b>4 – TESTING.....</b>	<b>30</b>
4.1 - UNIT TESTING.....	30
4.2 - FUNCTIONAL TESTING .....	30
4.2.1 – TEST PLAN .....	30
4.3 ATTEMPT TO FIX FAULTS.....	34
<b>5 - EVALUATION.....</b>	<b>36</b>
5.1 - COMPLETION OF OBJECTIVES .....	36
5.2 – WHAT I WOULD DO DIFFERENTLY .....	36
5.3 - FUTURE DEVELOPMENT.....	37
5.4- CONCLUSION .....	37
<b>6 - REFERENCES .....</b>	<b>38</b>
<b>7 - APPENDICES .....</b>	<b>40</b>
7.1 – PROGRAM CODE .....	40
7.1.1 - UMLDC.GUI.....	40
7.1.1.1 - ADDCLASSDIALOG.....	40
7.1.1.2 - ADDMETHODDIALOG .....	53
7.1.1.3 - ADDPARAMDIALOG .....	67
7.1.1.4 ADDVARIABLEDIALOG.....	76
7.1.1.5 – DRAWINGPANEL .....	87
7.1.1.6 - EDITMETHODDIALOG .....	93
7.1.1.7 - EDITPARAMDIALOG.....	99
7.1.1.8 - EDITVARIABLEDIALOG.....	103
7.1.1.9 - MYGUI.....	106
7.1.1.10 - MYJDIALOG .....	115
7.1.1.11 - UMLDC.....	124
7.1.2 – UMLDC.DATA.....	124
7.1.2.1 - ALLTYPES.....	124
7.1.2.2 – ATTRIBUTE .....	127
7.1.2.3 - CMETHOD .....	130
7.1.2.4 - CLASSDATA.....	138

7.1.2.5 - CLASSOBJECT .....	142
7.1.2.6 – PARAMETER .....	152
7.1.2.7 - RELATIONTYPE.....	154
7.1.2.8 - VISIBILITYTYPE .....	154
7.1.2.9 – ICLASSOBSERVER .....	156
7.1.2.10 – ICLASSSUBJECT .....	156
7.1.2.11 – IMETHODOBSERVER .....	157
7.1.2.12 – IMETHODSUBJECT.....	157
7.1.2.13 - METHODMEMENTO.....	158

# 1 - INTRODUCTION

## 1.1 - OVERVIEW OF PROJECT

As software developers, we are constantly looking for ways in which we can increase development productivity. The ultimate objective of my project was to develop a system that enables a user to create a UML class diagram. This diagram should also provide the option to output created classes as code.

## 1.2 - MOTIVATION

Many software businesses have software designers and software developers. The motivation of this project idea was developed due to the repetitiveness of typing basic lines of code, which slowed down the actual development of the main functionality within a software system. The desired outcome of this project was to slightly bridge the gap between design and development by producing a system that allows a software designer/developer to create and convert a UML class diagram directly into code. This will reduce the time gap between the designing stage and actual development of functionality within a software system.

## 1.3 - AIMS AND OBJECTIVES

This system differs from existing systems as it should provide input validation and error free code. To achieve this goal, there were several objectives that had to be completed.

### 1.3.1 - MAIN OBJECTIVES

Objective	Desired Outcome
1. Class Creation	On completing this objective a user should be able to create and delete classes. This includes adding variables and methods and relations.
2. Input validation	Input should be validated to check for valid identifiers etc.
3. Diagram Creation	On completion of this objective, a user should be able to visually represent created classes, to look like a UML class diagram. This includes arrows showing association and multiplicity.
4. Code Conversion	On completing this objective an option should be available for the user to convert the created diagram into working code.

*Table 1.3.1.1 – Main objectives table*

### 1.3.2 - DESIRABLE OBJECTIVES

Objective	Desired Outcome
1. Ensure that the generated code is error free	The generated code should not produce errors
2. Interactive diagram	Diagram objects should be draggable and responsive

3. Allow editing classes, variables, methods and parameters	Edits should be allowed
4. Allow project to be saved	On completion of this objective, it should be possible to save the current project and load it up again.
5. Support for several programming languages	On completing this objective, there should be an option to generate code for several different programming languages

*Table 1.3.1.1 – Desirable objectives table*

## 1.4 - REPORT STRUCTURE

Below is a brief overview of how this report will be structured and what will be discussed in each chapter.

**Chapter 1.5 Literature Review** – This chapter, which is part of the introduction will mention some of the research that was undertaken to gain some background knowledge with regards to the project that was undertaken. Topics that will be covered include: why UML was chosen as the modelling language, existing software that attempt to achieve a similar goal to this project and research into version control.

**Chapter 2 Planning and Design** – This chapter will outline the various diagrams that were produced to get a picture of how the project should look during development.

**Chapter 3 Implementation** – This chapter will detail the process of developing the system as well as the challenges that were met over the course of development.

**Chapter 4 Testing** – This chapter will go through the testing that had taken places and their results.

**Chapter 5 Evaluation and Conclusion** – In this chapter, an analysis of how I thought the project went. This chapter will also mention any improvements and developments that can be made in the future.

**Chapter 6 References** – This chapter will list the references used throughout the project.

**Chapter 7 Appendices** – This will contain the appendices of the project.

## 1.5 - LITERATURE REVIEW

### 1.5.1 - MODELLING LANGUAGES

Modelling concepts have been around since as early as 1958. Modelling has always been an essential part of developing information systems. In the very early attempts of modelling, focus was on describing the domain in strict, formal, and computer independent terms. What were modelled were data and operations on data. Data were modelled in abstract terms using various concepts. (Krogstie, Brinkkemper and Opdahl, 2007)

Modeling languages are mainly used in the field of computer science and engineering for designing models of new software, systems, devices and equipment. The context of modeling language is based on the requirements and specific domain in use. (Techopedia.com, n.d.) A modeling language can be made up of pseudo-code, actual code, pictures, diagrams, or long passages of description; in fact, it can be anything that helps describe a system. (Miles and Hamilton, 2006)

### 1.5.2 - WHY UML?

The Unified Modeling Language (UML) is a visual language for capturing software designs and patterns (Pilone and Pitman, 2005). Ciccozzi et al (2016) states that “UML has emerged and established itself as a de facto standard in industry, the most widely used architectural description language and an ISO/IEC standard.” This view is also supported by Miles and Hamilton (2016).

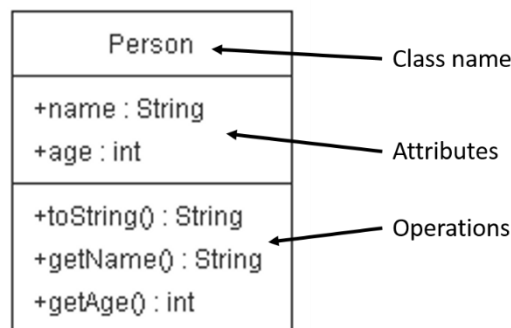
UML has six main advantages as stated by Miles and Hamilton (2016):

1. It is a formal language – Each element has a defined meaning. This makes it difficult for models to be misunderstood.
2. It is concise – The notation is simple and straight forward.
3. It is comprehensive – All the important parts of a system is described.
4. It is scalable –Due to the fact that it is formal, it can cater for extremely large systems as well as small projects.
5. It is built on lessons learned – It is built upon the best practices in the object oriented community.
6. It is the standard - The standard ensures UML’s transformability and interoperability, which means you aren’t tied to a particular product.

There are many modelling languages available. However, it was obvious to base the diagramming tool on a modelling language that is most used and accepted as the standard.

### 1.5.3 - UML CLASS DIAGRAMS

“Class diagrams are one of the most fundamental diagram types in UML. They are used to capture the static relationships of your software; in other words, how things are put together.” (Pilone and Pitman, 2005).



*Figure 1.5.3.1 – UML Class*

As shown in figure 1.5.3.1 UML classes are drawn as a rectangle split into up to three sections consisting of a class name, attributes and operations. The class name being the only section that is compulsory. (Miles and Hamilton, 2006). However, a UML class can also contain additional information like parameters, types, multiplicity, etc., which can make them more complex. In a normal UML class diagram, several can be shown at once with different lines drawn between classes illustrating the type of relationship.



### 1.5.4 - EXISTING SOFTWARE

There are several other software that try to achieve the same goal as mine. The two software that were looked at were ArgoUML and Visual Paradigms. It was apparent that both these software were lacking in several areas.

ArgoUML lacked sufficient error checking and any input validation. It was possible to create many attributes with the same identifier. As expected, this generated code with multiple variables with the same name. Furthermore, deleted attributes and operations were not omitted from the generated code. This is shown in figures 1.5.4.1 and 1.5.4.2. The software also allows users to use inappropriate identifier names such as “new” or “public”. Another issue with the software was that at times, the software produced unexpected code. ArgoUML requires the user to be cautious as to how they go about creating their desired UML class diagram. This makes it less suitable for larger software projects which require many classes in which small errors and mistakes are likely while designing the class diagram.

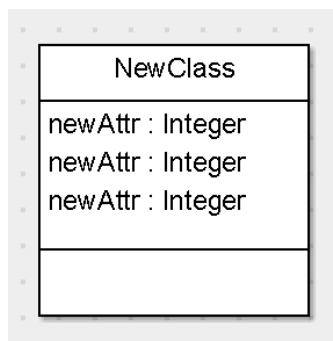


Figure 1.5.4.1 – Class in ArgoUML

```
public class NewClass {  
    public Integer newAttr;  
    public Integer newAttr;  
    public Integer newAttr;  
    public void newOperation() {  
    }  
}
```

Figure 1.5.4.2 – Code generated from class in ArgoUML

Visual Paradigms on the other hand solves many of these issues. However, it checks for many of these errors only during the code conversion process. During the creation of the UML diagram, the software allows most errors to take place. Another issue with Visual Paradigms is the pricing. For the code conversion tool, the user must purchase or subscribe for the standard edition of the software which may be too expensive for open-source or individual developers.

### 1.5.5 – DESIGN PATTERNS

While developing this project, many challenges were faced. To overcome these challenges many solutions were attempted. In several situations, design patterns were found to be the best solution to a problem in their context. For this reason, a lot of research went into design patterns and their uses.

Design patterns in Software engineering were introduced after the publication of a book called *Design Patterns: Elements of Reusable Object-Oriented Software* by a group of four, also known as the Gang of Four (GoF). Patterns, as defined by Christopher Alexander in the context of patterns in buildings and towns are a description of “a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.” (Gamma et al., 1994) The GoF say this is also true for object oriented design patterns.

Design patterns can be divided into three different types (Gamma et al., 1994; Freeman et al., 2004):

**Creational Patterns** – This type of pattern is concerned with how objects are created. They help make a system independent from how its objects are instantiated. Examples of this type of pattern are the singleton and prototype patterns.

**Structural Patterns** – This type of pattern involves how classes and objects are composed to form larger structures. Examples of this type of pattern are the Decorator and Adapter patterns.

**Behavioral Patterns** – This type of pattern is concerned with algorithms and how classes and objects interact and the assignment of responsibility. Examples of this type of pattern are the memento and observer patterns.

There are also more complex uses of patterns which are referred to as compound patterns. This is when two or more patterns are used together and combined to solve a recurring problem. (Freeman et al., 2004) An example of this is the famous architectural pattern known as Model-View-Controller (MVC). MVC use several different patterns together such as the strategy, observer and composite patterns.

It is obvious that design patterns improve development time and efficiency as they are based on tried and tested methods by experienced Software Engineers. However, as the potential use for design patterns were not immediately discovered until mid-development of this project. They could not be implemented as much as they could have due to time constraints.

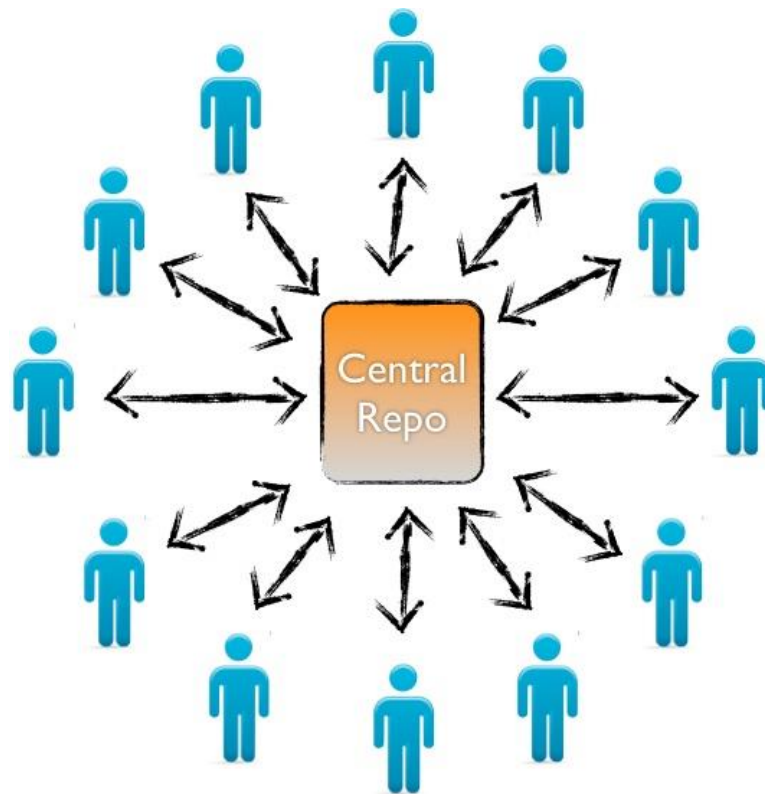
### 1.5.6 - VERSION CONTROL SYSTEMS

Version control was used for this project to ensure that code was backed up properly and any changes made to the code could be reverted if needed. “A tool that manages and tracks different versions of software or other content is referred to generically as a version control system (VCS)” (Loeliger and McCullough, 2012). There are many version control systems that are available, however the choice was narrowed down to two, Git and Subversion (SVN).

SVN and Git both track changes made to files and directories over time. This allows you to recover older versions of your data or examine the history of how your data changed. (Pilato, Collins-Sussman and Fitzpatrick, 2008). However they are both different in their own respect.

SVN is a Centralized Version Control System (CVCS) and Git is a Distributed Version Control System (DVCS). Centralized Version Control Systems store all versioned files within a single server. This is illustrated in figure 1.7.1. CVCSs make it easy for developers to collaborate with one another and has been used as the standard for version control for many years. However, CVCSs increase the risk of losing

everything. This is due to the fact that all files are stored in a central place which means that it also has a single point of failure. (Chacon and Straub, 2014)



*Figure 1.7.1 - Centralized model (Loeliger and McCullough, 2012)*

With Distributed Version Control Systems on the other hand, developers checkout the entire repository and not just a single snapshot of the latest version. This provides extra security for developers as each checkout is a full backup of all the data. Repositories can be copied back to the server to restore it in case of a server failure. (Chacon and Straub, 2014)

Although this is true, Subversion is still a choice of many developers today as it is simple and easy to use. What makes Git more complicated and harder to understand is the fact that Git allows commits to files to be made locally as well as remotely. This added functionality results in Git having more commands than Subversion. For the sake of this project Git was used for the added security. Furthermore, there are many very popular Git repository hosting services such as Bitbucket and GitHub which I can use to further develop the project with other contributors. Bitbucket was used for this project due to it providing free private repositories.

## 1.6 – GANTT CHART

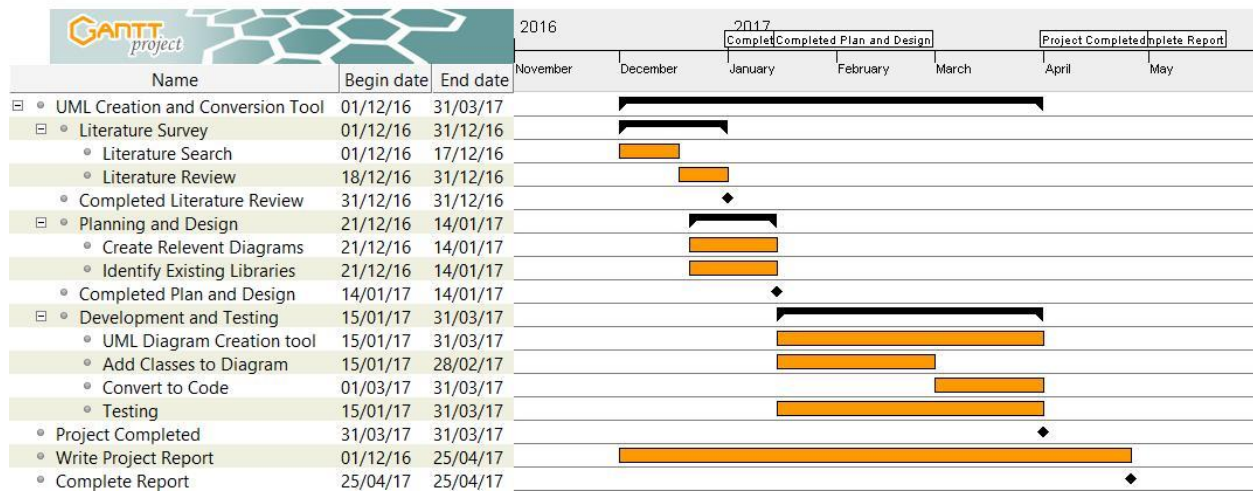


Figure 1.8.1 – Gantt chart (Ganttproject.biz, 2017)

Figure 1.8.1 shows the Gantt chart that was created. The Gantt chart was created to give a plan of how much time should be spent on each part of the project. This was to ensure that too much time was not spent on a single section, which would result in less time for future sections.

## 2 - PLANNING AND DESIGN

### 2.1 USE CASE DIAGRAM

Following the guidelines mentioned by Miles and Hamilton (2006) in Learning UML 2.0, a use case diagram and descriptions were made.

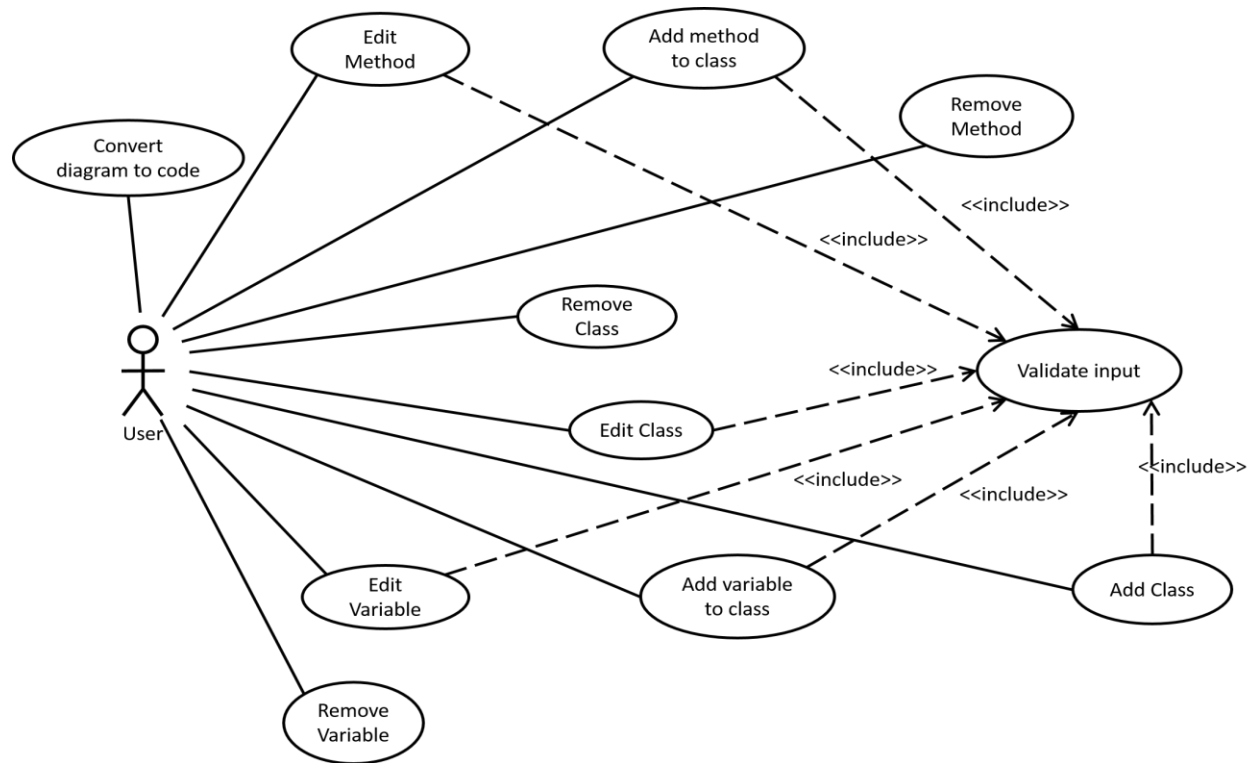


Figure 2.1.1 – Use Case diagram

### 2.2 - USE CASE DESCRIPTIONS

<b>Use case name</b>	Add Class	
<b>Related Requirements</b>	Edit class, remove class, add variable to class, edit variable, remove variable, add method to class, edit method, remove method, convert diagram to code	
<b>Goal in Context</b>	A user can create a class	
<b>Precondition</b>	The input of the user is valid	
<b>Successful End Condition</b>	A new class is added to the diagram	
<b>Failed End Condition</b>	A class object is not added to the diagram	
<b>Actor</b>	User	
<b>Trigger</b>	The user asks the system to create a new class	
<b>Main Flow</b>	<b>Step</b>	<b>Action</b>
	1	The User asks the system to create a new class

	2	The User enters information about the class
	3	Input is checked
	<b>Include:: Validate input</b>	
	4	A new class is added to the diagram
<b>Extensions</b>	<b>Step</b>	<b>Branching Action</b>
	3.1	Class is not created

<b>Use case name</b>	Edit Class	
<b>Related Requirements</b>	None	
<b>Goal in Context</b>	A user can edit a class	
<b>Precondition</b>	The class has been created. The input of the user is valid	
<b>Successful End Condition</b>	The class is updated and changes are made to the diagram	
<b>Failed End Condition</b>	The class is not updated	
<b>Actor</b>	User	
<b>Trigger</b>	The user asks the system to edit an existing class	
<b>Main Flow</b>	<b>Step</b>	<b>Action</b>
	1	The User asks the system to edit an existing class
	2	The User makes changes to the class
	3	Input is checked
	<b>Include:: Validate input</b>	
	4	The class is updated
<b>Extensions</b>	<b>Step</b>	<b>Branching Action</b>
	3.1	Class is not edited

<b>Use case name</b>	Remove Class	
<b>Related Requirements</b>	None	
<b>Goal in Context</b>	A user can remove a class	
<b>Precondition</b>	The class has been created. The input of the user is valid	
<b>Successful End Condition</b>	The selected class is removed	
<b>Failed End Condition</b>	The selected class is not removed	
<b>Actor</b>	User	
<b>Trigger</b>	The user tells the system to remove an existing class	
<b>Main Flow</b>	<b>Step</b>	<b>Action</b>
	1	The User asks the system to remove an existing class
	2	The class is removed

<b>Use case name</b>	Add Variable	
<b>Related Requirements</b>	Edit variable, remove variable	
<b>Goal in Context</b>	A user can add a variable to a class	

<b>Precondition</b>	The input of the user is valid	
<b>Successful End Condition</b>	A new variable is added to the class	
<b>Failed End Condition</b>	The variable is not added	
<b>Actor</b>	User	
<b>Trigger</b>	The User asks the system to add a variable to a class	
<b>Main Flow</b>	<b>Step</b>	<b>Action</b>
	1	The User asks the system to add a variable to a class
	2	The User enters information about the variable
	3	Input is checked
	<b>Include:: Validate input</b>	
	4	A new variable is added to the class
<b>Extensions</b>	<b>Step</b>	<b>Branching Action</b>
	3.1	Variable is not created

<b>Use case name</b>	Edit Variable	
<b>Related Requirements</b>	None	
<b>Goal in Context</b>	A user can edit a variable	
<b>Precondition</b>	The variable already exists. The input of the user is valid	
<b>Successful End Condition</b>	The variable is edited and changes are made to the diagram	
<b>Failed End Condition</b>	The variable is not updated	
<b>Actor</b>	User	
<b>Trigger</b>	The user asks the system to edit an existing variable	
<b>Main Flow</b>	<b>Step</b>	<b>Action</b>
	1	The User asks the system to edit an existing variable
	2	The User makes changes to the variable
	3	Input is checked
	<b>Include:: Validate input</b>	
	4	The variable is updated
<b>Extensions</b>	<b>Step</b>	<b>Branching Action</b>
	3.1	Variable is not edited

<b>Use case name</b>	Remove Variable	
<b>Related Requirements</b>	None	
<b>Goal in Context</b>	A user can remove a variable from a class	
<b>Precondition</b>	The variable has been created. The input of the user is valid	
<b>Successful End Condition</b>	The selected variable is removed	
<b>Failed End Condition</b>	The selected variable is not removed	
<b>Actor</b>	User	
<b>Trigger</b>	The user tells the system to remove an existing variable	
<b>Main Flow</b>	<b>Step</b>	<b>Action</b>

	1	The User asks the system to remove an existing variable
	2	The variable is removed

<b>Use case name</b>	Add Method	
<b>Related Requirements</b>	Edit Method, remove method	
<b>Goal in Context</b>	A user can add a method to a class	
<b>Precondition</b>	The input of the user is valid	
<b>Successful End Condition</b>	A new method is added to the class	
<b>Failed End Condition</b>	The method is not added	
<b>Actor</b>	User	
<b>Trigger</b>	The User asks the system to add a method to a class	
<b>Main Flow</b>	<b>Step</b>	<b>Action</b>
	1	The User asks the system to add a method to a class
	2	The User enters information about the method
	3	Input is checked
	<b>Include:: Validate input</b>	
	4	A new method is added to the class
<b>Extensions</b>	<b>Step</b>	<b>Branching Action</b>
	3.1	Method is not created

<b>Use case name</b>	Edit method	
<b>Related Requirements</b>	None	
<b>Goal in Context</b>	A user can edit a method	
<b>Precondition</b>	The method already exists. The input of the user is valid	
<b>Successful End Condition</b>	The method is edited and changes are made to the diagram	
<b>Failed End Condition</b>	The method is not updated	
<b>Actor</b>	User	
<b>Trigger</b>	The user asks the system to edit an existing method	
<b>Main Flow</b>	<b>Step</b>	<b>Action</b>
	1	The User asks the system to edit an existing method
	2	The User makes changes to the method
	3	Input is checked
	<b>Include:: Validate input</b>	
	4	The method is updated
<b>Extensions</b>	<b>Step</b>	<b>Branching Action</b>
	3.1	Method is not edited

<b>Use case name</b>	Remove method
----------------------	---------------



<b>Related Requirements</b>	None	
<b>Goal in Context</b>	A user can remove a method from a class	
<b>Precondition</b>	The method has been created. The input of the user is valid	
<b>Successful End Condition</b>	The selected method is removed	
<b>Failed End Condition</b>	The selected method is not removed	
<b>Actor</b>	User	
<b>Trigger</b>	The user tells the system to remove an existing method	
<b>Main Flow</b>	<b>Step</b>	<b>Action</b>
	1	The User asks the system to remove an existing method
	2	The method is removed

<b>Use case name</b>	Validate Input	
<b>Related Requirements</b>	Add Class, Edit class, add variable, edit variable, add method, remove method	
<b>Goal in Context</b>	The users input is validated	
<b>Precondition</b>	There is input from the user	
<b>Successful End Condition</b>	The user's input is validated	
<b>Failed End Condition</b>	The user's input is invalid	
<b>Actor</b>	User	
<b>Trigger</b>	The user enters information that should be validated by the system	
<b>Main Flow</b>	<b>Step</b>	<b>Action</b>
	1	The user enters information that should be validated by the system
	2	The system verifies the input
	3	The user's input is valid
<b>Extensions</b>	<b>Step</b>	<b>Branching Action</b>
	2.1	The user's input is invalid
	2.2	A warning is displayed to the user

<b>Use case name</b>	Convert Diagram to code	
<b>Related Requirements</b>	Add Class	
<b>Goal in Context</b>	A user can convert the diagram into code	
<b>Precondition</b>	At least one class exists	
<b>Successful End Condition</b>	The diagram is converted into code	
<b>Failed End Condition</b>	Code is not generated	
<b>Actor</b>	User	
<b>Trigger</b>	The user asks the system to convert the diagram into code	
<b>Main Flow</b>	<b>Step</b>	<b>Action</b>
	1	The user asks the system to convert the diagram into code
	2	The diagram is converted into code
<b>Extensions</b>	<b>Step</b>	<b>Branching Action</b>
	1.1	There are no classes to convert to code
	1.2	A warning is displayed to the user

## 2.3 - CLASS DIAGRAM

To understand the static relationships between classes, a class diagram was developed.

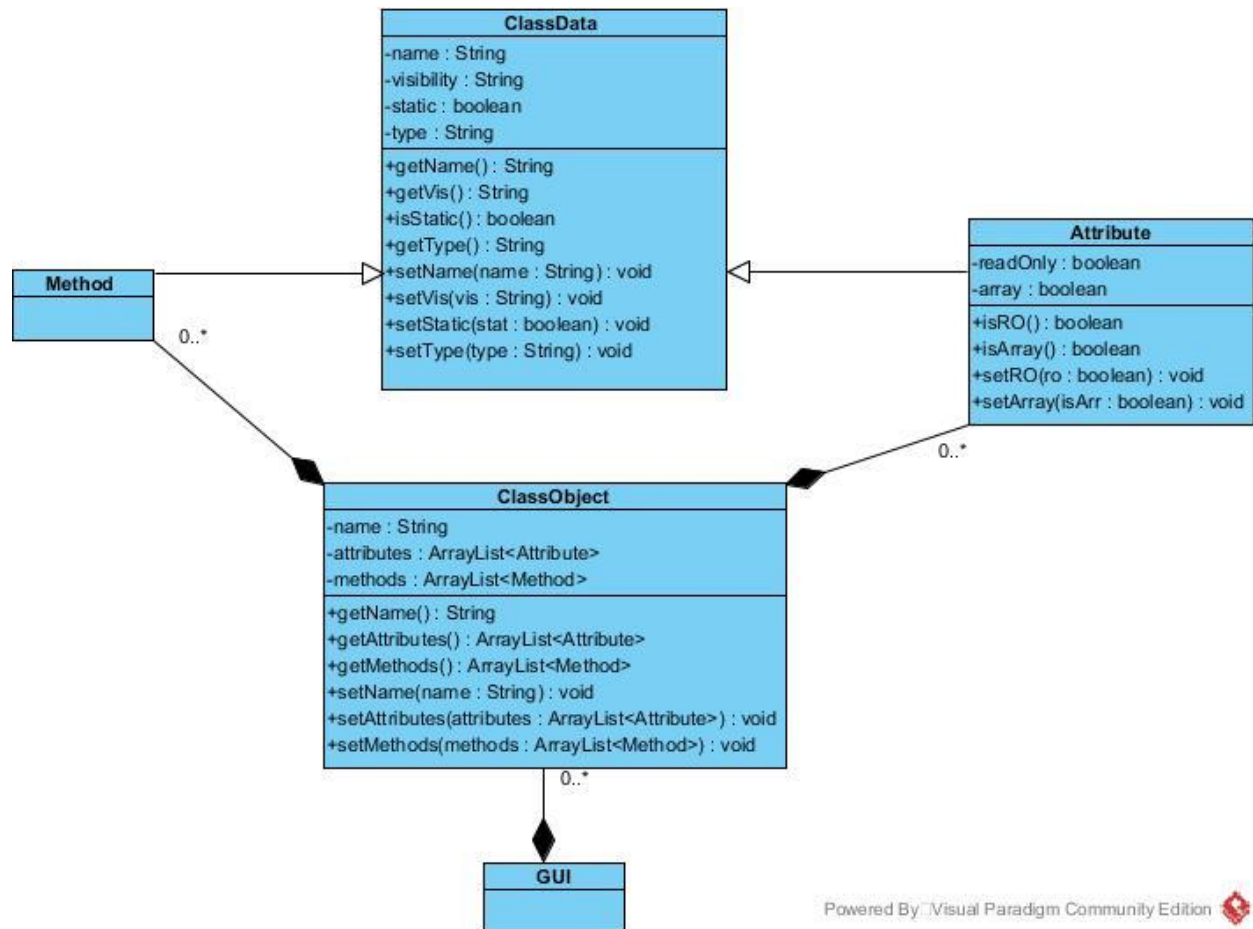


Figure 2.3.1 – Initial Class diagram

As much of the implementation of the system was unclear at this point, it was not possible to create a very detailed class diagram. At this stage it was also quite unclear as to how I will implement the GUI. During the course of development, there were changes to the class diagram which made the diagram different than it had already been designed.

One of the changes was that methods now also contain parameters, which are their own object. This was done because the way in which attributes and methods were added to classes had changed. This meant adding more detail to methods specifically as it was decided that methods would have a list of parameters.

## 2.4 - STORYBOARD

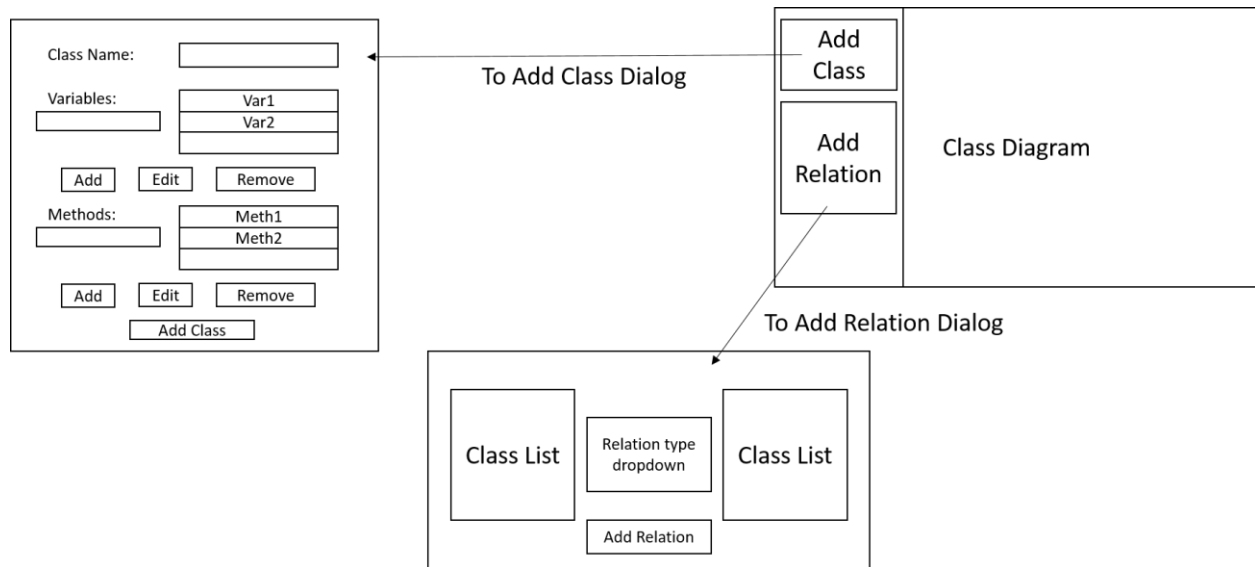


Figure 2.4.1 - Storyboard

Storyboards are generally used in visual media for the sake of outlining a story. I created a storyboard for this software project to gain an understanding of how the user interface should look and how navigation should work. The storyboard helped me to layout components within the GUI. Changes were made during development which will be discussed further in Chapter 3.

### 3 - DEVELOPMENT AND IMPLEMENTATION

The project was developed incrementally as the main aim was to produce fully working parts of code in segments. Certain parts of the system are dependent on others and certain parts are independent. For instance, it is necessary to add a class before adding variables to it. For this reason I created a diagram illustrating in more detail what will happen during the implementation phase.

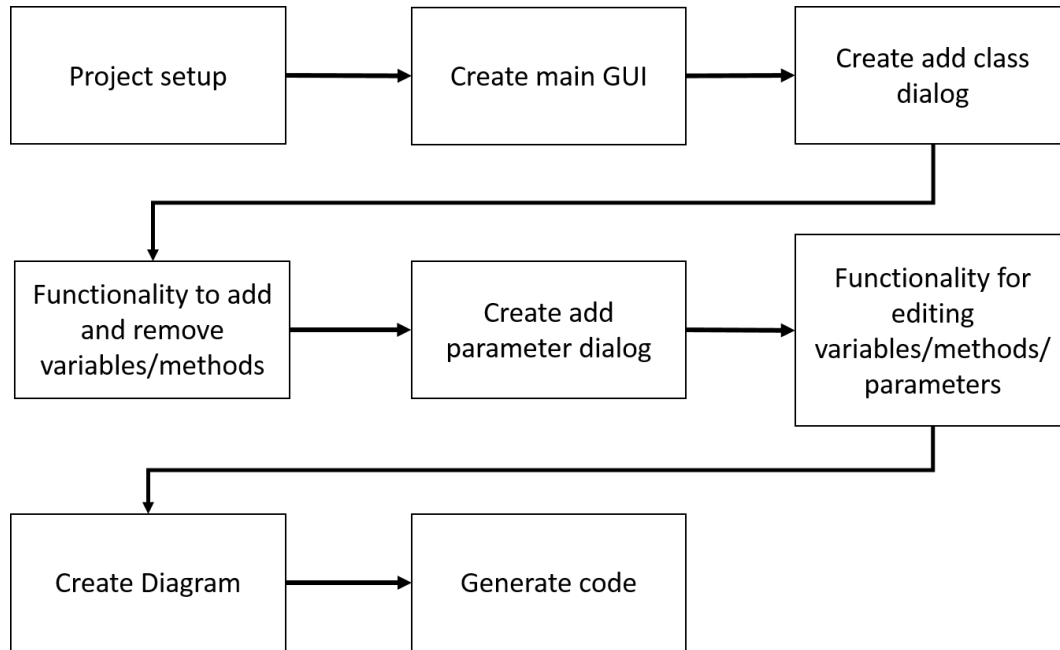


Figure 3.1 – Development plan

#### 3.1 SETTING UP

The first thing that needed to be done with regards to implementing the design, was to setup the project folder to ensure that everything was organized. As development progressed, classes needed to be organized in different packages to prevent confusion.

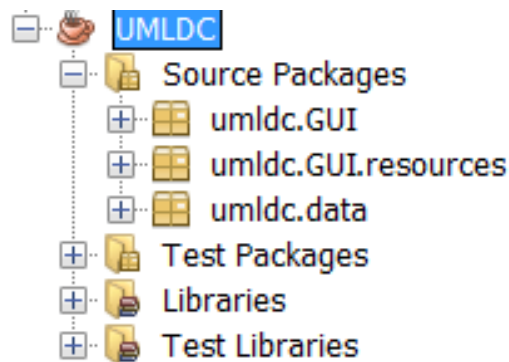


Figure 3.1.1 – File structure

I organized the project files as show in figure 3.1.1. The umldc.GUI package contains all classes related to the graphical user interfaces. The umldc.GUI.resources package contains the image files that were used in the GUI. The umldc.data file contains the files that were related to the backend of the project.

## 3.2 CLASS STRUCTURE

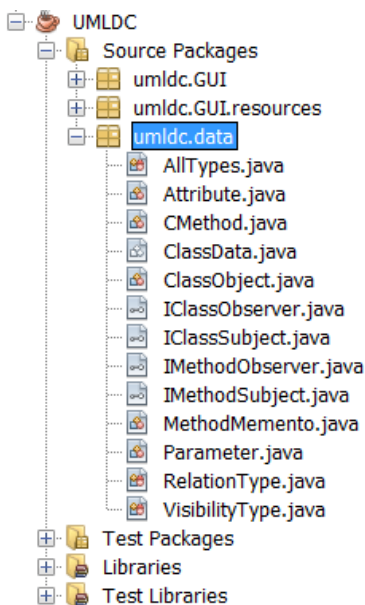


Figure 3.2.1 – Backend classes

The umlhc.data package contains the classes that are related to the backend of the project. Below is a brief description as to what each class represents with regards to the implementation of the software. **ClassObject**, **CMethod**, **Attribute** and **Parameter** are objects used for the creation of classes, methods, variables (attributes) and parameters. The classes ending in Observer or Subject are related to the implementation of the observer pattern. MethodMemento was created in an attempt to implement the memento pattern which will be discussed in more detail in Chapter 4.3. AllTypes, RelationType and VisibilityType are enum classes. AllTypes is for the variable and return types. VisibilityType is for the types of visibility Attributes (variables) and Operations (methods) can have. RelationType has not yet been implemented.

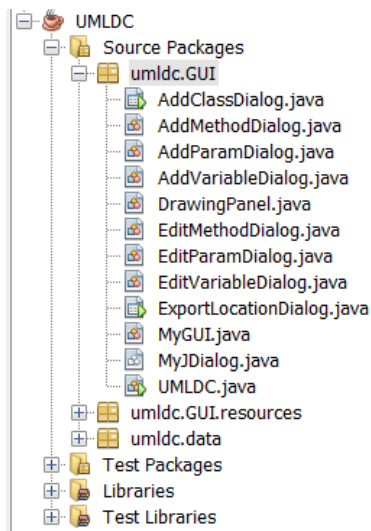


Figure 3.2.2 – GUI classes

The umldc.GUI folder contains the classes related to the GUI. The class diagram was not created for the GUI during the design stage as the implementation was unclear at that stage. Initially the Add/EditMethodDialog, Add/EditParamDialog and Add/EditVariableDialog were unrelated to each other as shown in figure 3.2.3.

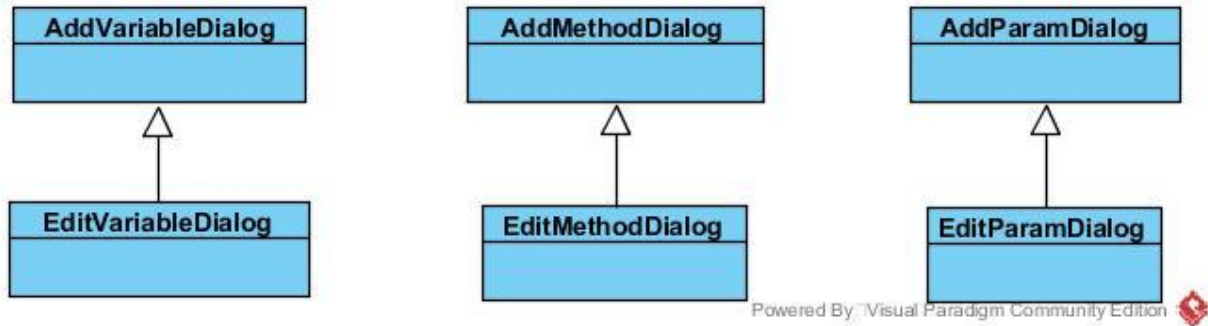


Figure 3.2.3 – Initial dialog relations

However, I found that there was a lot of duplicate code. Especially with regards to the input validation. For this reason the code was eventually refactored and the class structure was updated to what is shown in figure 3.2.4.

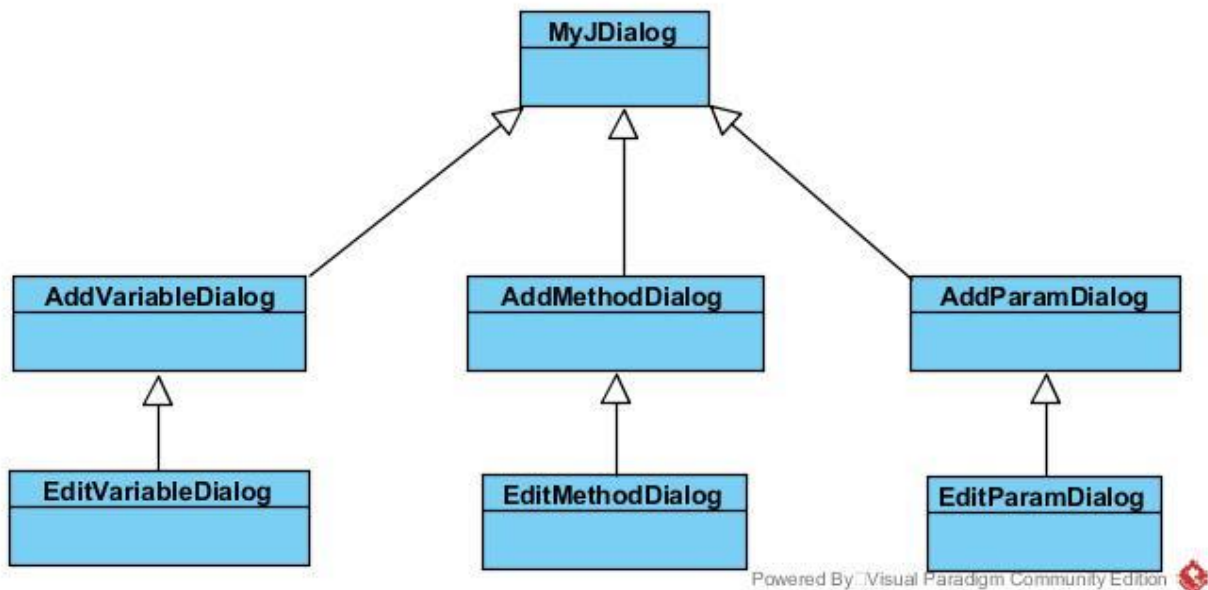


Figure 3.2.4 – Current dialog relations

The MyGUI class is the main interface for the user to add classes. The AddClassDialog is the class that is opened when the add class button is clicked. From there it is possible to add or edit variables or methods. The DrawingPanel is a JPanel which draws the class diagram. MyJDialog is inherited to provide common functionality to other dialogs as shown in figure 3.2.4. UMLDC contains the main method from which the program is run.

The implementation of these classes will be discussed in more detail in subsequent chapters and the full code for all classes is available in the appendices.

### 3.3 CREATING THE GUI

During this stage of development it was necessary for me to distinguish between the different types of components that are provided by Java.

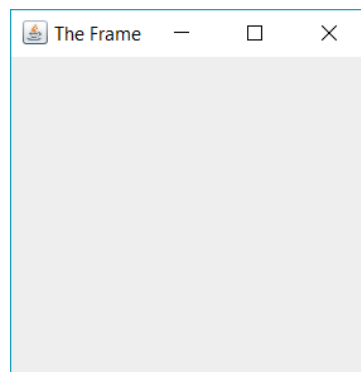
#### 3.3.1 WHY SWING?

It became obvious to use the swing library over the AWT library as it provided the benefits of being; platform-independent, lightweight and providing more powerful components than the AWT library. (www.javatpoint.com, n.d.)

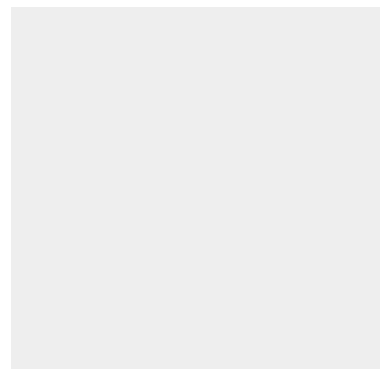
“The Swing component set was originally created because the basic AWT components that came with the original version of the Java libraries were insufficient for real-world, forms-based applications. All the basic components were there, but the existing set was too small and far too restrictive. For instance, you couldn’t even put an image on a button. To alleviate this situation, the Swing component set offers replacements for each of the AWT components. The Swing components support all the capabilities of the original set and offer a whole lot more besides.” (Zukowski, 2005)

#### 3.3.2 CHOOSING THE ROOT PANE CONTAINER

For the first stage of development I needed to create a main graphical user interface so that it was possible to navigate through the program. This is the main container which contains all other components. However, the swing library provides several different types of components which implement the RootPaneContainer interface. These are the: JFrame, JWindow, JDialog, JApplet and JInternalFrame.



*Figure 3.3.2.1 - Simple JFrame*



*Figure 3.3.2.2 – Simple JWindow*

As shown in figure 3.3.2.2 JWindow lacks the adornments of a JFrame such as the title bar and menu bar. JApplets are made to be embedded inside a web page and run within the context of a browser. JInternalFrames are usually used within a desktop pane. (Docs.oracle.com, n.d.) The choice was narrowed down to using a JDialog or JFrame. The JDialog represents the standard pop-up window for displaying information related to a Frame. (Zukowski, 2005) Also the main root container had no parents and also did not need to be modal. Therefore, I had decided to use the JFrame as the main container for the program.

Following the storyboard that was designed in the Planning and Design stage, the GUI was implemented.

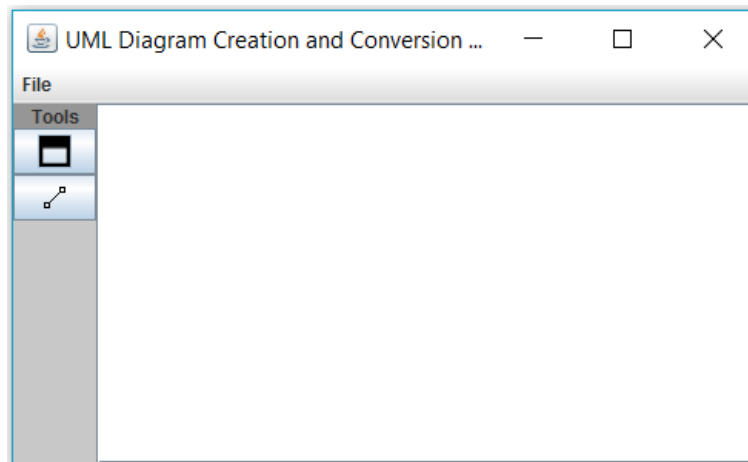


Figure 3.3.2.3 – Implementation of the main root container

### 3.4 ADDING CLASSES

When the add class button is clicked, a JDialog appears. On initialization a temporary class object (ClassObject) is created using an empty constructor. This makes it possible to add methods and variables to it. On clicking submit, the class object is added to an ArrayList.

For this stage of development, it was necessary for me to create a class structure which enabled the user to add variables and methods to a class in the backend. This was done by following the class diagram which was designed during the planning stage so that ClassObjects have an ArrayList of CMethods and Attributes.

An interface was then needed to allow the user to input the relevant information about a class. As mentioned in chapter 2, the design was changed during development. This was due to the fact that during the planning and design stage, the way in which input validation was taking place was not fully understood. I had initially planned that text would be entered into a JTextField which would then use a regular expression to check whether the user had entered a correct class, variable or method name.

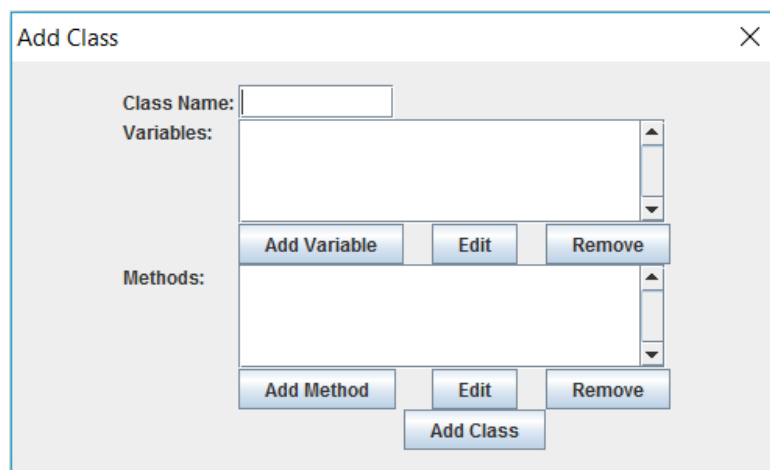


Figure 3.4.1 – Add class JDialog



However, this would be harder for a user to understand especially if they had a weaker programming background. Therefore a better solution was developed.

As shown in figure 3.4.1, instead of using a JTextField for adding variables and methods, add and edit variable/method buttons were created. These open a separate JDialog containing their relevant forms.

### 3.4.1 ADDING AND REMOVING VARIABLES AND METHODS

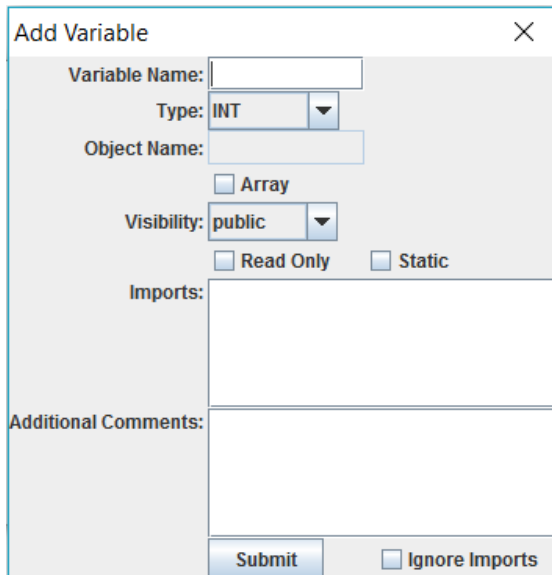
The 'Add Variable' dialog box contains the following fields and controls: 'Variable Name:' text field, 'Type:' dropdown menu with 'INT' selected, 'Object Name:' text field, an 'Array' checkbox, 'Visibility:' dropdown menu with 'public' selected, 'Read Only' and 'Static' checkboxes, an 'Imports:' text area, and an 'Additional Comments:' text area. At the bottom are 'Submit' and 'Ignore Imports' checkboxes.

Figure 3.4.1.1 – Add Variable JDialog

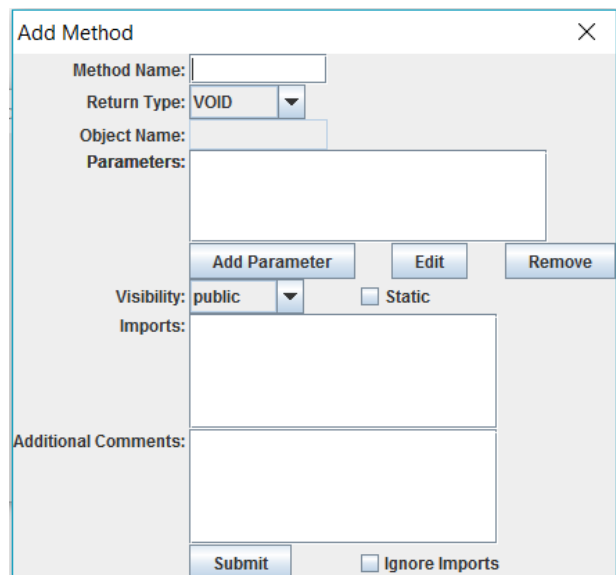
The 'Add Method' dialog box contains the following fields and controls: 'Method Name:' text field, 'Return Type:' dropdown menu with 'VOID' selected, 'Object Name:' text field, a 'Parameters:' text area, 'Add Parameter', 'Edit', and 'Remove' buttons, 'Visibility:' dropdown menu with 'public' selected, a 'Static' checkbox, an 'Imports:' text area, and an 'Additional Comments:' text area. At the bottom are 'Submit' and 'Ignore Imports' checkboxes.

Figure 3.4.1.2 – Add Method JDialog

Using various components, I generated forms using the Netbeans GUI Builder. This made it possible to add variables and methods to the class object. The added variables and methods are added to the JLists shown in figure 3.4.1 when the submit button is pressed. The variable, method and object name fields validate the input of the user to ensure it is a valid identifier as outlined in the Java Docs.

If the 'Object' type is selected, the object name field is enabled. The user is then allowed to enter a variable type that is not listed. In addition to a class name, a user is allowed to enter parameterized type(s) as an input. The input is validated to check if the input is correctly formatted. This was done by using a regular expression.

Currently, the object name field and imports text area do not scan the entire Java API to check whether or not the package or object name exists. This will be discussed further in the Future Development section. However it does provide some form of validation. The last word after a dot in the imports must be the same as the object name. For example, "asd.qwery.dsa.ArrayList" will be a valid import for the "ArrayList" object name. If the user selects ignore imports, the object name is not validated against the imports.

Also, as of yet, variables and parameters do not have a multiplicity. The dialog only allows the user to select whether or not a variable or parameter is an array.

### 3.4.2 EDITING VARIABLES AND METHODS

The edit method and variable dialog require data from the selected method and variable to initialize the data. I implemented this by passing the selected variable or method's data through the edit method and variable dialog constructor. When submitting the data, the method is updated. If parameters are added when editing a method, they are added to the method object but they would not be updating the JList where the methods are displayed for a class. This was also the case for editing parameters for methods.

After researching thoroughly it was decided that implementing the observer pattern would be most appropriate for this situation. "The Observer Pattern defines a one-to-many dependency between objects so that when one object changes state, all of its dependents are notified and updated automatically." (Freeman et al., 2004)

Before implementing the observer pattern within the actual project itself. Several test projects were created to fully understand how the observer pattern worked and which classes would implement which interfaces. The observer pattern was implemented for variables, methods and parameters as this provided cleaner code with less coupling.

According to Sarcar (2016), to implement the observer pattern a subject and observer are required. Subjects are objects that are being observed for changes. Observers are registered to a Subject and are notified whenever changes are made to that Subject. For example, in my system, CMethod implements the ISubjectMethod interface as changes to the method should be observed. The Add Method Dialog implements the IObserverMethod interface as it needs to update whenever a class is updated by adding/removing parameters.

The full implementation of the observer pattern is shown in the appendix.

There is another issue with regards to parameters which has not yet been solved. This will be discussed further in Chapter 4.3.

### 3.5 CREATING THE DIAGRAM

After allowing creation of classes it was necessary to add functionality to display them graphically. To do this, I created a custom component that would paint classes. "The JComponent class defines many aspects of AWT components that go above and beyond the capabilities of the original AWT component set. This includes customized painting behavior and several different ways to customize display settings, such as colors, fonts, and any other client-side settings." (Zukowski, 2005) The created component had to be treated as a drawing area which would contain the drawn class objects. Zukowski (2005) states that the JPanel component serves as a replacement for the Canvas component which is suitable as a draw able Swing component area. "JPanel is one of Swing's lightweight containers, which means that it is a component that can be added to the content pane of a JFrame." (Schildt, 2014) For this reason JPanel was chosen to extend and use as the drawing area. The paintComponent() method was overridden to customize the painting of this custom "Drawing Panel" component.

#### 3.5.1 ALLOWING FOR LARGE DIAGRAMS

When more and more classes are added, classes could be drawn outside of the container. Therefore the JPanel was placed within a JScrollPane. The size of the JPanel is decided upon dynamically as classes are added. This allows the scroll bar to appear when needed. To determine the position of each class diagram object, a "point" variable was created which would act as a pen. This will follow the position of

locations at which parts of a class diagram box is created. The resulting effect looks like what is shown in figure 3.3.1.

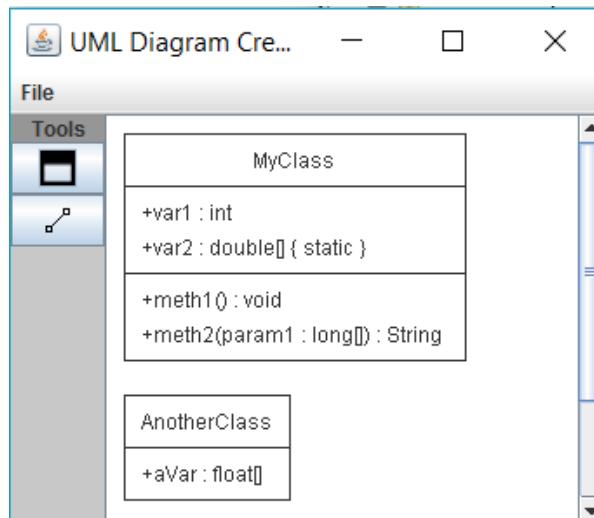


Figure 3.3.1.1 – Diagram generation

### 3.5.2 CLASS BOX SIZE

The size of each class diagram object is also determined dynamically based on the size of the longest string. This was done by utilizing the FontMetrics class. The font metrics class encapsulates various information related to a font. This information helps display text in a window. (Schildt, 2014) In order to get the width of a string in pixels, the FontMetrics provides a method called `stringWidth(String str)`. As shown in figure 3.3.2.1, it was possible to get the largest string width by looping through a `ClassObject`'s list of displayable strings (the class name, its variables and its methods).

```

//Return the longest string of a ClassObject
private int getLongestStringWidth(Graphics2D g2, ClassObject c){

    //List of all strings of a class
    ArrayList<String> strings = getAllStrings(c);
    FontMetrics metrics = g2.getFontMetrics();

    int longestStringSize = 0;

    //set longestStringSize to the size of the longest string
    for (String str : strings){
        if (metrics.stringWidth(str)> longestStringSize){
            longestStringSize = metrics.stringWidth(str);
        }
    }

    return longestStringSize;
}

//Get a list of all displayable strings of a ClassObject (class name,
//methods and variables)
private ArrayList<String> getAllStrings(ClassObject c){
    ArrayList<String> strings = new ArrayList<String>();

    strings.add(c.getClassName());

    for (Attribute var : c.getAttributes()) {
        strings.add(var.toString());
    }
    for (CMethod meth : c.getMethods()) {
        strings.add(meth.toString());
    }

    return strings;
}

```

Figure 3.3.2.1 – Code for getting the longest string width of a class object

Currently, the diagram is generated and classes are placed in fixed positions.

### 3.6 - CONVERTING TO CODE

After being able to visually represent classes, the classes would then need to be outputted as code. I implemented this by displaying a file chooser so that a relevant folder can be chosen for the generated code to go. Strings for class objects were created which would format the outputted code so that it is human-readable. The code is also intended to be error free. However, this is only the case if the variables or methods do not require imports as currently the software does not include the imports added.

```
public class aNewClass {  
    public String var1;  
    public int meth1(boolean param) {  
        int tempVar = 0;  
        return tempVar;  
    }  
}
```

*Figure 3.6.1 – Example code generated*

As you can see in figure 3.6.1 the generated code for a method containing a return type produces a temporary variable so that code is error free.

## 4 – TESTING

Software testing forms an integral part of the software development life cycle. This makes sense as a significant amount is spent every year due to software failures. (Zhivich and Cunningham, 2009) There are two main types of testing, Specification-Based Testing and Code-Based Testing. (Jorgensen, 2014)

**Specification-Based Testing** – Also known as black-box testing, is a form of testing in which the implementation of the system is not known and is used to check the inputs and outputs of the system. (Jorgensen, 2014)

**Code-Based Testing** – The main objective of code-based testing, which is also known as white box testing, is to “verify the correctness of the software’s statements, code paths, conditions, loops and data flow.” (Zhivich and Cunningham, 2009) This form of testing is mainly done by developers who understand the implementation of the system.

Most testing types fall under these two categories of testing. These are some that have been taken from the books *Software Testing: A Craftsman’s Approach* and *Software Testing: Testing Across the Entire Software Development Life Cycle*: Static Testing, Functional Testing, Structural Testing, Performance Testing, Integration Testing, System Testing etc.

### 4.1 - UNIT TESTING

This form of testing allowed me to check if my code was working correctly. This was done by testing functions within classes to see if they produced their expected results. While developing the system a testing framework called JUnit was discovered. JUnit allows developers to unit test the code elegantly. (Acharya, 2014) However, due to time constraints, not all classes were tested using JUnit.

### 4.2 - FUNCTIONAL TESTING

The main aim of functional testing is to “validate the software behavior against the business functionality documented in the software requirements.” (Everett and McLeod, 2007) Using the use case descriptions a test plan was devised to validate whether or not the system behaved as expected. Tests to method parameters were still added as they were not part of the initial design or use case descriptions.

#### 4.2.1 – TEST PLAN

ID	Test	Sample Data	Expected Outcome	Actual Outcome
1	Adding a Class	<b>1</b> – Without variables or methods <b>2</b> – without any variables <b>3</b> – without any methods <b>4</b> – with multiple variables and methods <b>5</b> – Set class name as blank field <b>6</b> – set class name as ClassName	<b>1, 2, 3, 4, 6 &amp; 9</b> – A new class object should be added to the diagram displaying its information. <b>5</b> – User should be warned that the class name cannot be blank	<b>1, 2, 3, 4, 6 &amp; 9</b> – A new class object was added to the diagram displaying its information. <b>5</b> – A warning was displayed saying that the class name cannot be blank

		<b>7</b> – set class name as !ClassName <b>8</b> – set class name as 123Class <b>9</b> – set class name as _Class and \$Class <b>10</b> – set class name as a Java keyword <b>11</b> – set class name to a space	<b>7, 8, 10 &amp; 11</b> – User should be warned that there is invalid input	<b>7, 8, 10 &amp; 11</b> – A warning was displayed saying that there was invalid input
2	Adding variables	<b>1</b> – Set variable name as blank field <b>2</b> – set variable name as var1 <b>3</b> – set variable name as !var <b>4</b> – set variable name as 123var <b>5</b> – set variable name as _var and \$var <b>6</b> – set variable name as a Java keyword <b>7</b> – Set type to Object and object name to a space <b>8</b> – Set type to Object and object name to "type1" <b>9</b> – Ignore imports, set type to object and object name to "asd" <b>10</b> - Ignore imports, set type to object and object name to "asd<>" <b>11</b> - Ignore imports, set type to object and object name to "asd <asd,dsa>" <b>12</b> - Ignore imports, set type to object and object name to "asd < asd , dsa >" <b>13</b> - Ignore imports, set type to object and object name to ".asd < asd , dsa >" <b>14</b> - Ignore imports, set type to object and object name to "asd < !asd , dsa >" <b>15</b> - Ignore imports, set type to object and object name to "asd < asd , dsa >." <b>16</b> - Ignore imports, set type to object and object name to "_asd < asd , _dsa >"	<b>2, 5, 9, 11, 12, 15, 16 &amp; 18</b> – A variable should be added to the class <b>1, 3, 4, 6, 7, 10, 13, 14 &amp; 17</b> – User should be notified that input is incorrect <b>8</b> – User should be notified that imports are required	<b>2, 5, 9, 11, 12, 15, 16 &amp; 18</b> – A variable was added to the class <b>1, 3, 4, 6, 7, 10, 13, 14 &amp; 17</b> – Warning displayed saying that input is incorrect <b>8</b> – Warning displayed showing that imports are required

		<b>17</b> – Uncheck ignore imports, type set to Object, object name and imports should be different strings <b>18</b> – Uncheck ignore imports, type set to Object, object name “asd” and imports set to “qwerty.dsa.asd”		
3	Adding methods	Same as test 1	Same as test 1	Same as test 1
4	Adding Parameters *This test was done while adding a method	Similar inputs to test 1	Similar expected outputs to test 1	Similar outputs as test 1
5	Removing Variables	1 – var1, var2 and var3	The removed variable should not be visible in the list that shows all the variables of a class. The removed variable should not be displayed on the diagram and should not be visible in the generated code	The removed variable does not show in the list of variables of a class. The removed variable is not displayed on the diagram and is not visible in the generated code
6	Removing Methods	1 – meth1(), meth2(String param1), meth3()	The removed method should not be visible in the list that shows all the methods of a class. The removed method should not be displayed on the diagram and should not be visible in the generated code	The removed method does not show in the list of methods of a class. The removed method is not displayed on the diagram and is not visible in the generated code
7	Removing Parameters *This test was done while adding a method	1 – A method with three parameters	The removed parameter should not be visible in the list that shows all the parameters of a method. The removed parameter should not be	The removed parameter does not show in the list of parameters of a method. The removed parameter is not displayed on the



			displayed on the diagram and should not be visible in the generated code	diagram and is not visible in the generated code
8	Editing Parameters *This test was done while adding a method	1 – A method with parameters	<p>Changes to a parameter should update the list of parameters that are displayed to the user when the user clicks submit. The diagram and generated code should also show these changes.</p> <p>Cancelling the edits should not update the list nor should it affect the diagram or generated code</p>	<p>Changes to a parameter updated the list of parameters that were displayed to the user when submit was clicked. The diagram and generated code also showed these changes.</p> <p>Cancelling the edits did not update the list nor did it affect the diagram or generated code</p>
9	Editing Variables	1 – A variable	<p>Changes to the variable should update the list of variables that are displayed to the user when the user clicks submit. The diagram and generated code should also show these changes.</p> <p>Cancelling the edits should not update the list nor should it affect the diagram or generated code</p>	<p>Changes to the variable updated the list of variables that were displayed to the user when submit was clicked. The diagram and generated code also showed these changes.</p> <p>Cancelling the edits did not update the list nor did it affect the diagram or generated code</p>
10	Editing Methods	1 – A method with parameters	Changes to the method should update the list of methods that are displayed to the user when the user	Failed

			clicks submit. The diagram and generated code should also show these changes.  Cancelling the edits should not update the list nor should it affect the diagram or generated code	
11	Generating Code	1 – Add class with no variables or methods 2 – Add Class with variables and methods with primitive data types only 3 – Add class with variables and methods with non-primitive data types 4 – Add class with methods that have parameters 5 – Add class with method that has a return type other than void	<b>1, 2, 4, 5</b> – Should produce error free code that is easily readable <b>3</b> – Should produce code that has errors due to packages that have not been imported	<b>1, 2, 4, 5</b> – Generated code was error free and easily readable <b>3</b> – Generated code had errors due to packages that have not been imported

As you can see from the results produced from the tests mentioned above all the tests passed except one.

#### 4.3 ATTEMPT TO FIX FAULTS

When editing a method, if parameters are added, edited or removed and the JDialog is closed before submitting, the parameters are still added to the method. An attempt has been tried to mitigate this but was not successful.

The selected method is passed into the EditMethodDialog constructor. For this reason, parameters were added directly to the method. Therefore, it was thought that this was the only reason why changes were added to the method regardless of whether or not the dialog was cancelled. It was necessary to create a temporary method object whenever a method was edited. When submitting, the edited method would replace the selected method. However, this solution still did not work.

For this reason, the memento pattern seemed most appropriate to use in this situation. The memento pattern is used to save the state of an object, so that in the future, it can be restored back to the specified state. (Sarcar, 2016) After creating a test project to understand the way the memento pattern could be implemented, it was used within the project. However, this still did not solve the problem.

It was then assumed that there was an issue with regards to the object references. This would explain why changes were being made to the original selected method. However, testing the references of the selected method and the temporary method showed that they were both different.

Attempting to mitigate this problem further would result in more time being lost and would result in other parts of the project not being implemented. For this reason, the issue was not resolved and still remains.

## 5 - EVALUATION

### 5.1 - COMPLETION OF OBJECTIVES

Main Objectives	Complete
5. Class Creation	Partially. Users are able to create classes with methods and variables but currently it is not possible to add relations.
6. Input validation	Yes
7. Diagram Creation	Partially. On creation of a class, the diagram is updated with a class object which shows its information in the format of a UML Class. However, the diagram does not show arrows to indicate relations to other classes.
8. Code Conversion	Yes. The diagram can be converted into working code.

Table 5.1.1 – Main objectives completion

Desirable Objective	Complete
6. Ensure that the generated code is error free	Partially. If the user does not use any custom variable types, the outputted code is error free. However, this issue is solved if the user adds imports for those types to the generated code. This is a minor issue as most IDEs provide functionality for adding dependencies easily.
7. Interactive diagram	No, however the system does allow for a large amount of classes to be created as the panel for drawing the classes is expanded and scrollable when classes are added.
8. Allow editing classes, variables, methods and parameters	Partially. Variables, methods and parameters can be edited. However, classes cannot be edited.
9. Allow project to be saved	No
10. Support for several programming languages	No, Java is the only supported language at this time.

Table 5.1.2 – Desirable objectives completion

### 5.2 – WHAT I WOULD DO DIFFERENTLY

Looking back at the whole project it is obvious that I had made many mistakes. A lot of time was wasted refactoring code due to poor design choices and the discovery of new tools. Researching in more depth about the tools I was using would have reduced a lot of the wasted time. An example of this was the Netbeans GUI builder which allows a simple drag and drop system. Initially I had manually written some GUI classes. Utilizing the Netbeans GUI builder from the start would have saved a lot of time.

Furthermore, doing more in depth research into testing could have helped significantly as testing frameworks could have been discovered and utilized at a much earlier date. Also giving even more time to testing at the end of the project would have helped as I could have tested more parts of the system using a testing framework.

Using the knowledge that I have now, the project could have gone much more smoothly.

### 5.3 - FUTURE DEVELOPMENT

As shown in Chapter 5.1, not all objectives were complete. Making the repository where this system is contained public would be my intended first step to making improvements to this system. This will allow contributors to suggest changes and improvements to my system. I have also added documentation to classes which would make it easier for contributors to understand what has already been written.

I also intend on finishing off all the incomplete objectives. After this there are many potential ideas that can be incorporated within the system.

One idea that came to mind during development but was difficult to implement due to time constraints was an improvement to the way a user can add a custom variable/parameter type. Currently the system checks input using a regular expression. To improve validation, the system should check the Java API as well as imported libraries for all classes that match the user's inputted variable type. Due to the fact there are classes with the same name but with different packages, a list of all potential imports can be shown to the user which they can add.

After all this there are still more improvements that can be made to the system. Instead of outputting a simple structure of classes without functionality this system can be a small part of a much bigger system. Users should be able to create different UML diagrams and have them work together to produce behavioral code. During development it was clear that some parts of UML were difficult to translate directly into code. This could potentially be a problem for other diagrams too. However, modifications can be made that are specific to this system so that the ultimate goal of producing code with functionality can be achieved.

### 5.4- CONCLUSION

I feel that this project was a success. Even though it was not completely finished, I feel that my knowledge of Java and the object-oriented way of developing has increased vastly.

The initial project plan had two main parts. The aim of the first part was to develop a system that would allow a user to create a UML class diagram and then convert that diagram into Java code. The second aim that was initially part of the project plan was to convert an image of a UML Class diagram into code. However, after making a start on the project it was clear that I had underestimated the complexity of what I was developing. Therefore the second part of the project was dropped and the main focus was then put on the first part of the project.

I feel that the development of this project went at a slower pace than expected. This was due to the fact that my plan had not taken into account the learning and research needed to overcome certain challenges during development. Furthermore, over the course of development there were changes in the storyboard as it became clearer as to which elements of a UML class diagram would actually be included within my project. A lot of time was spent on learning and researching new techniques that could be implemented into the system. During the course of development, functionality was added and removed and it was difficult to stay strictly according to the original plan and design. This was due to the fact that while developing certain parts of the system, it became evident that certain parts would be difficult to implement within the given time frame. However, most of the main objectives were mostly complete and the system can still be used to generate code that is error free.

## 6 - REFERENCES

1. Acharya, S. (2014). *Mastering unit testing using Mockito and JUnit*. 1st ed. Birmingham, UK: Packt Publishing.
2. Argouml.tigris.org. (n.d.). *argouml.tigris.org*. [online] Available at: <http://argouml.tigris.org/> [Accessed 2 Mar. 2017].
3. Chacon, S. and Straub, B. (2014). *Pro Git*. 2nd ed. Apress.
4. Ciccozzi, F., Seceleanu, T., Corcoran, D. and Scholle, D. (2016). UML-Based Development of Embedded Real-Time Software on Multi-Core in Practice: Lessons Learned and Future Perspectives. *IEEE Access*, 4, pp.6528-6540.
5. Docs.oracle.com. (n.d.). DefaultListModel (Java Platform SE 7 ). [online] Available at: <https://docs.oracle.com/javase/7/docs/api/javax/swing/DefaultListModel.html> [Accessed 10 Apr. 2017].
6. Docs.oracle.com. (n.d.). JList (Java Platform SE 7 ). [online] Available at: <https://docs.oracle.com/javase/7/docs/api/javax/swing/JList.html> [Accessed 10 Apr. 2017].
7. Docs.oracle.com. (n.d.). Lesson: Java Applets (The Java™ Tutorials > Deployment). [online] Available at: <https://docs.oracle.com/javase/tutorial/deployment/applet/> [Accessed 15 Mar. 2017].
8. Everett, G. and McLeod, R. (2007). *Software Testing: Testing Across the Entire Software Development Life Cycle*. 1st ed. Hoboken, New Jersey: John Wiley & Sons, Inc.
9. Freeman, E., Robson, E., Sierra, K. and Bates, B. (2004). *Head First design patterns*. 1st ed. Sebastopol, CA: O'Reilly.
10. Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1994). *Design patterns*. 1st ed. Reading, Mass.: Addison-Wesley.
11. Ganttproject.biz. (2017). GanttProject: free desktop project management app. [online] Available at: <http://www.ganttproject.biz/> [Accessed 9 Apr. 2017].
12. Jorgensen, P. (2014). *Software testing: A Craftsman's Approach*. 4th ed. CRC Press.
13. Khan, M. (2010). Different Forms of Software Testing Techniques for Finding Errors. *IJCSI International Journal of Computer Science*, 7(3), pp.11-16.
14. Krogstie, J., Brinkkemper, S. and Opdahl, A. (2007). *Conceptual modelling in information systems engineering*. 1st ed. Berlin: Springer.
15. Loeliger, J. and McCullough, M. (2012). *Version control with Git*. 2nd ed. Beijing: O'Reilly.
16. Miles, R. and Hamilton, K. (2006). *Learning UML 2.0*. 1st ed. Beijing: O'Reilly.
17. Netbeans.org. (n.d.). Netbeans. [online] Available at: <https://netbeans.org/> [Accessed 21 Apr. 2017].
18. PANKAJ (2016). Memento Design Pattern in Java - JournalDev. [online] JournalDev. Available at: <http://www.journaldev.com/1734/memento-design-pattern-java> [Accessed 5 Apr. 2017].
19. PANKAJ (2016). Observer Design Pattern in Java - JournalDev. [online] JournalDev. Available at: <http://www.journaldev.com/1739/observer-design-pattern-in-java> [Accessed 5 Apr. 2017].
20. Pilato, M., Collins-Sussman, B. and Fitzpatrick, B. (2008). *Version Control with Subversion*. 2nd ed. O'Reilly.
21. Pilone, D. and Pitman, N. (2005). *UML 2.0 in a nutshell*. 1st ed. Sebastopol, Calif.: O'Reilly Media.
22. Sarcar, V. (2016). *Java Design Patterns*. 1st ed. Berkeley, CA: Apress.
23. Schildt, H. (2014). *Java, The Complete Reference*. 9th ed. McGraw-Hill Education.

24. Sintès, T. (2001). Speaking on the Observer pattern. [online] JavaWorld. Available at: <http://www.javaworld.com/article/2077444/learn-java/speaking-on-the-observer-pattern.html> [Accessed 15 Apr. 2017].
25. Techopedia.com. (n.d.). *What is Modeling Language? - Definition from Techopedia*. [online] Available at: <https://www.techopedia.com/definition/20810/modeling-language> [Accessed 7 Apr. 2017].
26. Visual-paradigm.com. (n.d.). *Software Design Tools for Agile Teams, with UML, BPMN and More*. [online] Available at: <https://www.visual-paradigm.com/> [Accessed 2 Mar. 2017].
27. www.javatpoint.com. (n.d.). Java Swing Tutorial - javatpoint. [online] Available at: <http://www.javatpoint.com/java-swing> [Accessed 15 Mar. 2017].
28. www.tutorialspoint.com. (n.d.). Design Patterns Memento Pattern. [online] Available at: [https://www.tutorialspoint.com/design\\_pattern/memento\\_pattern.htm](https://www.tutorialspoint.com/design_pattern/memento_pattern.htm) [Accessed 10 Apr. 2017].
29. www.tutorialspoint.com. (n.d.). Design Patterns Observer Pattern. [online] Available at: [https://www.tutorialspoint.com/design\\_pattern/observer\\_pattern.htm](https://www.tutorialspoint.com/design_pattern/observer_pattern.htm) [Accessed 9 Apr. 2017].
30. Zhivich, M. and Cunningham, R. (2009). The Real Cost of Software Errors. *IEEE Security & Privacy Magazine*, 7(2), pp.87-90.
31. Zukowski, J. (2005). The definitive guide to Java Swing. 3rd ed. Berkeley, CA: Apress.

## 7 - APPENDICES

### 7.1 – Program Code

#### 7.1.1 - umldc.GUI

The initComponents() methods were generated using the Netbeans GUI builder. Also any code that states that it is not to be modified was also generated by Netbeans.

##### 7.1.1.1 - AddClassDialog

```
package umldc.GUI;

import umldc.data.CMethod;
import umldc.data.ClassObject;
import umldc.data.Attribute;
import java.awt.Frame;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.util.ArrayList;
import java.util.concurrent.CopyOnWriteArrayList;
import static javax.lang.model.SourceVersion.isName;
import javax.swing.*.*;
import umldc.data.IClassObserver;

/**
 *
 * @author rayha
 */
public class AddClassDialog extends javax.swing.JDialog implements IClassObserver {

    /**
     * Creates new form AddClassDialog
     */
    private MyGUI parent;

    /**
     *
     * @param parent the JFrame that ran this dialog
     * @param modal true if you want this dialog to be modal, otherwise false
     */
}
```



```

public AddClassDialog(JFrame parent, boolean modal) {
    super(parent, modal);
    initComponents();
    this.parent = (MyGUI)parent;
    classOb = new ClassObject();
    classOb.registerObserver(this);

    this.addWindowListener(new WindowAdapter() {

        public void windowClosing(WindowEvent e) {
            classOb.removeObserver((AddClassDialog)e.getWindow());

        }
    });
}

/**
 * This method is called from within the constructor to initialize the form.
 * WARNING: Do NOT modify this code. The content of this method is always
 * regenerated by the Form Editor.
 */
@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">
private void initComponents() {
    java.awt.GridBagConstraints gridBagConstraints;

    jLabel1 = new javax.swing.JLabel();
    fieldClassName = new javax.swing.JTextField();
    jLabel2 = new javax.swing.JLabel();
    buttonAddVar = new javax.swing.JButton();
    sPVarList = new javax.swing.JScrollPane();
    variableListView = new javax.swing.JList<>();
    buttonRemVar = new javax.swing.JButton();
    buttonEditVar = new javax.swing.JButton();
    buttonSubmit = new javax.swing.JButton();
    jLabel3 = new javax.swing.JLabel();

```

```

sPMethList = new javax.swing.JScrollPane();
methodListView = new javax.swing.JList<>();
buttonAddMeth = new javax.swing.JButton();
buttonEditMeth = new javax.swing.JButton();
buttonRemMeth = new javax.swing.JButton();

setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
setTitle("Add Class");
setMinimumSize(new java.awt.Dimension(500, 200));
setResizable(false);
getContentPane().setLayout(new java.awt.GridBagLayout());

jLabel1.setText("Class Name: ");
jLabel1.setVerticalTextPosition(javax.swing.SwingConstants.TOP);
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 0;
getContentPane().add(jLabel1, gridBagConstraints);

fieldClassName.setCursor(new java.awt.Cursor(java.awt.Cursor.TEXT_CURSOR));
fieldClassName.setPreferredSize(new java.awt.Dimension(100, 22));
fieldClassName.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        fieldClassNameActionPerformed(evt);
    }
});

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 1;
gridBagConstraints.gridy = 0;
gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
getContentPane().add(fieldClassName, gridBagConstraints);

jLabel2.setHorizontalAlignment(javax.swing.SwingConstants.TRAILING);
jLabel2.setText("Variables: ");
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;

```

```

gridBagConstraints.gridy = 1;
gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
getContentPane().add(jLabel2, gridBagConstraints);

buttonAddVar.setText("Add Variable");
buttonAddVar.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        buttonAddVarActionPerformed(evt);
    }
});

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 1;
gridBagConstraints.gridy = 4;
gridBagConstraints.anchor = java.awt.GridBagConstraints.WEST;
getContentPane().add(buttonAddVar, gridBagConstraints);

SPVarList.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);
SPVarList.setAutoscrolls(true);

variableListView.setModel(varsListModel);
variableListView.setSelectionMode(javax.swing.ListSelectionModel.SINGLE_SELECTION);
variableListView.setMaximumSize(new java.awt.Dimension(46, 500));
variableListView.setVisibleRowCount(4);
SPVarList.setViewPortView(variableListView);

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 1;
gridBagConstraints.gridy = 1;
gridBagConstraints.gridwidth = 3;
gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
getContentPane().add(SPVarList, gridBagConstraints);

buttonRemVar.setText("Remove");
buttonRemVar.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        buttonRemVarActionPerformed(evt);
    }
});

```

```

    }

});

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 3;
gridBagConstraints.gridy = 4;
gridBagConstraints.anchor = java.awt.GridBagConstraints.EAST;
getContentPane().add(buttonRemVar, gridBagConstraints);

buttonEditVar.setText("Edit");
buttonEditVar.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        buttonEditVarActionPerformed(evt);
    }
});

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 2;
gridBagConstraints.gridy = 4;
getContentPane().add(buttonEditVar, gridBagConstraints);

buttonSubmit.setText("Add Class");
buttonSubmit.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        buttonSubmitActionPerformed(evt);
    }
});

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 2;
gridBagConstraints.gridy = 10;
getContentPane().add(buttonSubmit, gridBagConstraints);

jLabel3.setText("Methods:");
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 5;
gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
getContentPane().add(jLabel3, gridBagConstraints);

```

```

sPMethList.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);

sPMethList.setAutoscrolls(true);

methodListView.setModel(methsListModel);

methodListView.setSelectionMode(javax.swing.ListSelectionModel.SINGLE_SELECTION);

methodListView.setMaximumSize(new java.awt.Dimension(46, 500));

methodListView.setVisibleRowCount(4);

sPMethList.setViewportView(methodListView);

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 1;
gridBagConstraints.gridy = 5;
gridBagConstraints.gridwidth = 3;
gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
getContentPane().add(sPMethList, gridBagConstraints);

buttonAddMeth.setText("Add Method");
buttonAddMeth.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        buttonAddMethActionPerformed(evt);
    }
});

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 1;
gridBagConstraints.gridy = 8;
gridBagConstraints.anchor = java.awt.GridBagConstraints.WEST;
getContentPane().add(buttonAddMeth, gridBagConstraints);

buttonEditMeth.setText("Edit");
buttonEditMeth.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        buttonEditMethActionPerformed(evt);
    }
});

gridBagConstraints = new java.awt.GridBagConstraints();

```

```

gridBagConstraints.gridx = 2;
gridBagConstraints.gridy = 8;
getContentPane().add(buttonEditMeth, gridBagConstraints);

buttonRemMeth.setText("Remove");
buttonRemMeth.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        buttonRemMethActionPerformed(evt);
    }
});

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 3;
gridBagConstraints.gridy = 8;
gridBagConstraints.anchor = java.awt.GridBagConstraints.EAST;
getContentPane().add(buttonRemMeth, gridBagConstraints);

pack();
} // </editor-fold>

private void fieldClassNameActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
}

private void buttonAddVarActionPerformed(java.awt.event.ActionEvent evt) {

    AddVariableDialog addVarDialog = new AddVariableDialog(null, true, classOb);
    addVarDialog.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
    addVarDialog.pack();
    addVarDialog.setLocationRelativeTo(this);
    addVarDialog.setVisible(true);
}

private void buttonRemVarActionPerformed(java.awt.event.ActionEvent evt) {

    classOb.removeAttribute(variableListView);

```

```

        System.out.println(classOb.getAttributes());
        System.out.println("*****");
        System.out.println(varsListModel);

    }

    private void buttonEditVarActionPerformed(java.awt.event.ActionEvent evt) {

        if (!classOb.getAttributes().isEmpty() && !variableListView.isSelectionEmpty()) {

            varSelectedIndex = variableListView.getSelectedIndex();

            EditVariableDialog editVarDialog = new EditVariableDialog(null, true, classOb,
varSelectedIndex);

            editVarDialog.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);

            editVarDialog.pack();

            editVarDialog.setLocationRelativeTo(this);

            editVarDialog.setVisible(true);

        }

    }

    private void buttonAddMethActionPerformed(java.awt.event.ActionEvent evt) {

        AddMethodDialog addMethDialog = new AddMethodDialog(null, true, classOb);

        addMethDialog.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);

        addMethDialog.pack();

        addMethDialog.setLocationRelativeTo(this);

        addMethDialog.setVisible(true);

    }

    private void buttonEditMethActionPerformed(java.awt.event.ActionEvent evt) {

```

```

        if (!classOb.getMethods().isEmpty() && !methodListView.isSelectionEmpty()) {

            int selectedMethodIndex = methodListView.getSelectedIndex();

            CMethod methh = methods.get(selectedMethodIndex);

            System.out.println("METHOD REFERENCE");

            System.out.println(methh.hashCode());

            methh.setState(methh);

            AddMethodDialog editMethDialog = new EditMethodDialog(null, true, classOb,
selectedMethodIndex);

            editMethDialog.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);

            editMethDialog.pack();

            editMethDialog.setLocationRelativeTo(this);

            editMethDialog.setVisible(true);

            System.out.println(classOb.getMethods());

        }
    }

    private void buttonRemMethActionPerformed(java.awt.event.ActionEvent evt) {

        classOb.removeMethod(methodListView);

        System.out.println("Class Ob");

        System.out.println(classOb);

        System.out.println("Class Ob Methods");

        System.out.println(classOb.getMethods());

        System.out.println("Method List Model");

        System.out.println(methsListModel);

    }

    private void buttonSubmitActionPerformed(java.awt.event.ActionEvent evt) {

        String className = fieldClassName.getText().trim();

```



```

        if(isName(className) && !parent.classExists(classOb)){
            classOb.setClassName(className);
            parent.addClass(classOb);
            parent.repaint();
            dispose();
        } else {
            JOptionPane.showMessageDialog(this,
                "Class name is invalid or already exists",
                "Warning",
                JOptionPane.WARNING_MESSAGE);
        }
    }

}

/**
 * @param args the command line arguments
 */
public static void main(String args[]) {
    /* Set the Nimbus look and feel */
    //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
    /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and
    feel.
        * For details see
http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
        */
    try {
        for (javax.swing.UIManager.LookAndFeelInfo info :
            javax.swing.UIManager.getInstalledLookAndFeels()) {
            if ("Nimbus".equals(info.getName())) {
                javax.swing.UIManager.setLookAndFeel(info.getClassName());
                break;
            }
        }
    } catch (ClassNotFoundException ex) {

```

```

java.util.logging.Logger.getLogger(AddClassDialog.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

        } catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(AddClassDialog.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

        } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(AddClassDialog.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

        } catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(AddClassDialog.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

    }

    //</editor-fold>

    /* Create and display the dialog */
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            AddClassDialog dialog = new AddClassDialog(new javax.swing.JFrame(), true);
            dialog.addWindowListener(new java.awt.event.WindowAdapter() {
                @Override
                public void windowClosing(java.awt.event.WindowEvent e) {
                    System.exit(0);
                }
            });
            dialog.setVisible(true);
        }
    });

}

/**
 *
 * @return the JList called variableListView
 */
public JList<String> getVariableListView() {
    return variableListView;
}

```

```

/**
 *
 * @return the DefaultListModel called varsListModel
 */
public DefaultListModel getVarsListModel() {
    return varsListModel;
}

/**
 *
 * @return the classOb
 */
public ClassObject getClassOb() {
    return classOb;
}

/**
 *
 * @return JList called methodListView
 */
public JList<String> getMethodListView() {
    return methodListView;
}

/**
 *
 * @return DefaultListModel called methsListModel
 */
public DefaultListModel getMethsListModel() {
    return methsListModel;
}

/**
 *
 * @param cOb The class object to update

```

```

    */

@Override
public void update(ClassObject cOb) {
    this.classOb = cOb;
    varsListModel.clear();
    methsListModel.clear();
    CopyOnWriteArrayList<Attribute> variables= this.classOb.getAttributes();
    methods= this.classOb.getMethods();

    if (!variables.isEmpty()) {
        for (Attribute var : variables) {
            varsListModel.addElement(var);
        }
    }

    if (!methods.isEmpty()) {
        for (CMethod meth : methods) {
            methsListModel.addElement(meth);
        }
    }
}

// Variables declaration - do not modify
private javax.swing.JButton buttonAddMeth;
private javax.swing.JButton buttonAddVar;
private javax.swing.JButton buttonEditMeth;
private javax.swing.JButton buttonEditVar;
private javax.swing.JButton buttonRemMeth;
private javax.swing.JButton buttonRemVar;
private javax.swing.JButton buttonSubmit;
private javax.swing.JTextField fieldClassName;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel2;
private javax.swing.JLabel jLabel3;

```

```

    private javax.swing.JList<String> methodListView;
    private javax.swing.JScrollPane sPMethList;
    private javax.swing.JScrollPane sPVarList;
    private javax.swing.JList<String> variableListView;
    // End of variables declaration

    private DefaultListModel varsListModel = new DefaultListModel();
    private DefaultListModel methsListModel = new DefaultListModel();
    CopyOnWriteArrayList<CMethod> methods;
    private ClassObject classOb;
    private int varSelectedIndex;
}

```

### 7.1.1.2 - AddMethodDialog

```

package umldc.GUI;

import umldc.data.ClassObject;
import umldc.data.CMethod;
import umldc.data.Parameter;
import umldc.data.VisibilityType;
import umldc.data.IMethodObserver;
import umldc.data.AllTypes;
import java.awt.Frame;
import java.awt.event.ActionEvent;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.text.CharacterIterator;
import java.text.StringCharacterIterator;
import java.util.ArrayList;
import java.util.concurrent.CopyOnWriteArrayList;
import static javax.lang.model.SourceVersion.isName;
import javax.swing.DefaultComboBoxModel;
import javax.swing.DefaultListModel;
import javax.swing.JCheckBox;
import javax.swing.JComboBox;
import javax.swing.JDialog;

```

```

import javax.swing.JList;
import javax.swing.JOptionPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;

/**
 *
 * @author rayha
 */
public class AddMethodDialog extends MyJDialog implements IMethodObserver{

    // Created by NetBeans GUI Builder
    private javax.swing.JButton buttonSubmit;
    private javax.swing.JCheckBox cbArray;
    private javax.swing.JCheckBox cbRO;
    private javax.swing.JCheckBox cbStatic;
    private javax.swing.JComboBox<String> comboType;
    private javax.swing.JComboBox<String> comboVisibility;
    private javax.swing.JTextField fieldObName;
    private javax.swing.JTextField fieldMethName;

    /**
     *
     */
    protected javax.swing.JList<String> paramListView;
    private javax.swing.JScrollPane sPParamList;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel labelParam;
    private javax.swing.JScrollPane sPComments;
    private javax.swing.JTextArea taComments;

```

```

private javax.swing.JButton buttonAddParam;
private javax.swing.JButton buttonEditParam;
private javax.swing.JButton buttonRemParam;

private ArrayList<String> imports = new ArrayList<String>();
private boolean ignoreImports = false;
private String readOnly = "";
private String staticVar = "";
private CMethod meth;
private int selectedParamIndex;

/**
 *
 */
protected DefaultListModel paramListModel = new DefaultListModel();
private ArrayList<Parameter> paramsToAdd = new ArrayList<Parameter>();

/**
 *
 * @param parent parent that ran this dialog
 * @param modal true or false if this dialog is modal
 * @param cOb the class object which should contain the method
 */
public AddMethodDialog(Frame parent, boolean modal, ClassObject cOb) {
    super(parent, modal, cOb);
    classOb = cOb;
    this.meth = new CMethod();
    this.meth.registerObserver(this);
    paramListView.setModel(paramListModel);
    System.out.println(meth.getObservers());
}

/**
 * This method is called from within the constructor to initialize the form.

```

```

* WARNING: Do NOT modify this code. The content of this method is always
* regenerated by the Form Editor.
*/

@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">
//Generated by Netbeans GUI Editor
@Override
public void initComponents() {

    //java.awt.GridBagConstraints gridBagConstraints;

    jLabel1 = new javax.swing.JLabel();
    jLabel2 = new javax.swing.JLabel();
    jLabel3 = new javax.swing.JLabel();
    jLabel5 = new javax.swing.JLabel();
    jLabel7 = new javax.swing.JLabel();
    labelParam = new javax.swing.JLabel();
    fieldMethName = new javax.swing.JTextField();
    jLabel6 = new javax.swing.JLabel();
    cbRO = new javax.swing.JCheckBox();
    buttonSubmit = new javax.swing.JButton();
    sPComments = new javax.swing.JScrollPane();
    taComments = new javax.swing.JTextArea();
    paramListView = new javax.swing.JList<>();
    sPParamList = new javax.swing.JScrollPane();
    cbStatic = new javax.swing.JCheckBox();
    cbArray = new javax.swing.JCheckBox();
    jLabel4 = new javax.swing.JLabel();
    comboType = new javax.swing.JComboBox<>();
    fieldObName = new javax.swing.JTextField();
    comboVisibility = new javax.swing.JComboBox<>();
    buttonAddParam = new javax.swing.JButton();
    buttonEditParam = new javax.swing.JButton();
    buttonRemParam = new javax.swing.JButton();

    setTitle("Add Method");

```



```

jLabel1.setText("Method Name: ");

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 0;
gridBagConstraints.anchor = java.awt.GridBagConstraints.EAST;
getContentPane().add(jLabel1, gridBagConstraints);

jLabel2.setText("Visibility: ");
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 5;
gridBagConstraints.anchor = java.awt.GridBagConstraints.EAST;
getContentPane().add(jLabel2, gridBagConstraints);

labelParam.setText("Parameters: ");
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 3;
gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHEAST;
getContentPane().add(labelParam, gridBagConstraints);

paramListView.setModel(new DefaultListModel());
paramListView.setSelectionMode(javax.swing.ListSelectionModel.SINGLE_SELECTION);
paramListView.setMaximumSize(new java.awt.Dimension(46, 500));
paramListView.setVisibleRowCount(4);
sPParamList.setViewPortView(paramListView);

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 1;
gridBagConstraints.gridy = 3;
gridBagConstraints.gridwidth = 3;
gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
getContentPane().add(sPParamList, gridBagConstraints);

```

```

jLabel7.setText("Parameters: ");

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 3;
gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHEAST;
getContentPane().add(jLabel7, gridBagConstraints);

buttonAddParam.setText("Add Parameter");
buttonAddParam.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        buttonAddParamActionPerformed(evt);
    }
});

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 1;
gridBagConstraints.gridy = 4;
gridBagConstraints.anchor = java.awt.GridBagConstraints.WEST;
getContentPane().add(buttonAddParam, gridBagConstraints);

buttonEditParam.setText("Edit");
buttonEditParam.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        buttonEditParamActionPerformed(evt);
    }
});

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 2;
gridBagConstraints.gridy = 4;
gridBagConstraints.anchor = java.awt.GridBagConstraints.CENTER;
getContentPane().add(buttonEditParam, gridBagConstraints);

buttonRemParam.setText("Remove");
buttonRemParam.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {

```

```

        buttonRemParamActionPerformed(evt);
    }

});

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 3;
gridBagConstraints.gridy = 4;
gridBagConstraints.anchor = java.awt.GridBagConstraints.EAST;
getContentPane().add(buttonRemParam, gridBagConstraints);

jLabel5.setText("Additional Comments: ");
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 7;
gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHEAST;
getContentPane().add(jLabel5, gridBagConstraints);

fieldMethName.setPreferredSize(new java.awt.Dimension(100, 22));
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridwidth = java.awt.GridBagConstraints.REMAINDER;
gridBagConstraints.anchor = java.awt.GridBagConstraints.WEST;
getContentPane().add(fieldMethName, gridBagConstraints);
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 1;
gridBagConstraints.anchor = java.awt.GridBagConstraints.EAST;
getContentPane().add(jLabel6, gridBagConstraints);

buttonSubmit.setText("Submit");
buttonSubmit.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        buttonSubmitActionPerformed(evt);
    }
});

```

```

    }

});

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 1;
gridBagConstraints.gridy = 10;
gridBagConstraints.anchor = java.awt.GridBagConstraints.WEST;
getContentPane().add(buttonSubmit, gridBagConstraints);

taComments.setColumns(20);
taComments.setRows(5);
sPComments.setViewportView(taComments);

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 1;
gridBagConstraints.gridy = 7;
gridBagConstraints.gridwidth = java.awt.GridBagConstraints.REMAINDER;
gridBagConstraints.anchor = java.awt.GridBagConstraints.WEST;
getContentPane().add(sPComments, gridBagConstraints);

jLabel4.setText("Return Type: ");
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 1;
gridBagConstraints.anchor = java.awt.GridBagConstraints.EAST;
getContentPane().add(jLabel4, gridBagConstraints);

cbStatic.setText("Static");
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 2;
gridBagConstraints.gridy = 5;
gridBagConstraints.anchor = java.awt.GridBagConstraints.WEST;
getContentPane().add(cbStatic, gridBagConstraints);

jLabel3.setText("Object Name: ");
gridBagConstraints = new java.awt.GridBagConstraints();

```

```

gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 2;
gridBagConstraints.anchor = java.awt.GridBagConstraints.EAST;
getContentPane().add(jLabel3, gridBagConstraints);

this.fieldObjectName.setEditable(false);
fieldObjectName.setToolTipText("Object name");
fieldObjectName.setPreferredSize(new java.awt.Dimension(100, 22));
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 1;
gridBagConstraints.gridy = 2;
gridBagConstraints.anchor = java.awt.GridBagConstraints.WEST;
getContentPane().add(fieldObjectName, gridBagConstraints);

comboBoxType.setModel(new DefaultComboBoxModel(AllTypes.values()));
comboBoxType.setPreferredSize(null);
comboBoxType.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        comboBoxTypeActionPerformed(evt);
    }
});

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 1;
gridBagConstraints.gridy = 1;
gridBagConstraints.anchor = java.awt.GridBagConstraints.WEST;
getContentPane().add(comboBoxType, gridBagConstraints);

comboBoxVisibility.setModel(new DefaultComboBoxModel(VisibilityType.values()));
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 1;
gridBagConstraints.gridy = 5;
gridBagConstraints.anchor = java.awt.GridBagConstraints.WEST;
getContentPane().add(comboBoxVisibility, gridBagConstraints);

this.addWindowListener(new WindowAdapter() {

```



```

        meth.setName(methName);

        meth.setType(methType);
        meth.setIsStatic(isStatic);
        meth.setComments(comms);
        meth.setVisibility(visType);
        System.out.println(meth);
        if (!classOb.methodExists(meth)) {
            classOb.addMethod(meth);
            System.out.println("METHOD PARAMETERS after method submission");
            System.out.println(meth.getParameters());

            dispose();
        }
    } else {
        JOptionPane.showMessageDialog(this,
            "Invalid method name or type",
            "Warning",
            JOptionPane.WARNING_MESSAGE);
    }
}

private void comboRTypeActionPerformed(java.awt.event.ActionEvent evt) {

    AllTypes selectedType = (AllTypes) comboType.getSelectedItem();

    if (selectedType.equals(AllTypes.OBJECT)) {
        fieldObName.setEditable(true);
    } else {
        fieldObName.setEditable(false);
    }
}
}

```

```

/**
 *
 * @param evt
 */
protected void buttonAddParamActionPerformed(ActionEvent evt) {
    MyJDialog addParamDialog = new AddParamDialog(null, true, meth);
    addParamDialog.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
    addParamDialog.setLocationRelativeTo(this);
    addParamDialog.pack();
    addParamDialog.setVisible(true);
    System.out.println("AddParamActionPerformed");
    System.out.println(meth.getParameters());
}

/**
 *
 * @param evt
 */
protected void buttonEditParamActionPerformed(ActionEvent evt) {

    System.out.println("METHOD PARAMETERS EditParamActionPerformed");
    System.out.println(meth.getParameters());

    if ( !meth.getParameters().isEmpty() && !paramListView.isSelectionEmpty() ) {

        selectedParamIndex = paramListView.getSelectedIndex();
        MyJDialog editParamDialog = new EditParamDialog(null, true, meth, selectedParamIndex);
        editParamDialog.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
        editParamDialog.setLocationRelativeTo(this);
        editParamDialog.pack();
        editParamDialog.setVisible(true);
    }
}

/**
 *

```



```

    * @param evt
    */
protected void buttonRemParamActionPerformed(ActionEvent evt) {
    selectedParamIndex = paramListView.getSelectedIndex();
    meth.removeParam(selectedParamIndex);
}

/**
 *
 * @return the JComboBox called comboType
 */
public JComboBox<String> getComboType() {
    return comboType;
}

/**
 *
 * @return the JComboBox called comboVisibility
 */
public JComboBox<String> getComboVisibility() {
    return comboVisibility;
}

/**
 *
 * @return the JTextField called fieldMethName
 */
public JTextField getFieldMethName() {
    return fieldMethName;
}

/**
 *
 * @return JTextField called fieldObName
 */
public JTextField getFieldObName() {

```

```

        return fieldObName;
    }

    /**
     *
     * @return JCheckBox called cbStatic
     */
    public JCheckBox getCbStatic() {
        return cbStatic;
    }

    /**
     *
     * @return JList called ParamListView
     */
    public JList<String> getParamListView() {
        return paramListView;
    }

    /**
     *
     * @return JTextArea called taComments
     */
    public JTextArea getTaComments() {
        return taComments;
    }

    /**
     *
     * @return currently no implementation
     */
    public String generateComments() {
        return "";
    }

```

```

    }

    /**
     *
     * @param meth the method that should be updated
     */
    @Override
    public void update(CMethod meth) {
        this.meth = meth;

        paramListModel.clear();
        CopyOnWriteArrayList<Parameter> parameters= this.meth.getParameters();

        if (!parameters.isEmpty()) {
            for (Parameter param : parameters) {
                paramListModel.addElement(param);
            }
        }

        System.out.println(this.meth.getParameters());
        paramListView.setModel(paramListModel);
    }

}

```

### 7.1.1.3 - AddParamDialog

```

package umldc.GUI;

import umldc.data.CMethod;
import umldc.data.Parameter;
import umldc.data.AllTypes;
import java.awt.Frame;
import java.awt.event.ActionEvent;
import java.util.ArrayList;
import static javax.lang.model.SourceVersion.isName;
import javax.swing.DefaultComboBoxModel;
import javax.swing.JCheckBox;
import javax.swing.JComboBox;

```

```

import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;

/**
 *
 * @author rayha
 */
public class AddParamDialog extends MyJDialog{

    // Created by NetBeans GUI Builder
    private javax.swing.JButton buttonSubmit;
    private javax.swing.JCheckBox cbArray;
    private javax.swing.JComboBox<String> comboType;
    private javax.swing.JTextField fieldObName;
    private javax.swing.JTextField fieldVarName;
    private javax.swing.JLabel labelName;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel labelComments;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JScrollPane sPComments;
    private javax.swing.JTextField jTextField3;
    private javax.swing.JTextArea taComments;
    // End of variables declaration

    private CMethod methh;
    private ArrayList<String> imports = new ArrayList<String>();
    private Parameter param;

    /**
     *
     * @param parent parent that ran this dialog
     * @param modal true or false if this dialog is modal

```

```

    * @param meth the method which should contain the parameter
    */

    public AddParamDialog(Frame parent, boolean modal, CMethod meth ) {

        super(parent, modal);

        this.methh = meth;

        System.out.println(meth.hashCode());

        this.setTitle("Add Parameter");


        labelName.setText("Parameter Name: ");
    }

    /**
     *
     */

    @Override

    public void initComponents() {

        labelName = new javax.swing.JLabel();

        jLabel3 = new javax.swing.JLabel();

        labelComments = new javax.swing.JLabel();

        fieldVarName = new javax.swing.JTextField();

        jLabel6 = new javax.swing.JLabel();

        buttonSubmit = new javax.swing.JButton();

        sPComments = new javax.swing.JScrollPane();

        taComments = new javax.swing.JTextArea();

        cbArray = new javax.swing.JCheckBox();

        jLabel4 = new javax.swing.JLabel();

        comboType = new javax.swing.JComboBox<>();

        fieldObName = new javax.swing.JTextField();

        jLabel7 = new javax.swing.JLabel();

        setTitle("Add Variable");
    }

```

```

labelName.setText("Variable Name: ");

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 0;
gridBagConstraints.anchor = java.awt.GridBagConstraints.EAST;
getContentPane().add(labelName, gridBagConstraints);


jLabel3.setText("Object Name: ");

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 2;
gridBagConstraints.anchor = java.awt.GridBagConstraints.EAST;
getContentPane().add(jLabel3, gridBagConstraints);


labelComments.setText("Additional Comments: ");

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 7;
gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHEAST;
getContentPane().add(labelComments, gridBagConstraints);


fieldVarName.setPreferredSize(new java.awt.Dimension(100, 22));

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridwidth = java.awt.GridBagConstraints.REMAINDER;
gridBagConstraints.anchor = java.awt.GridBagConstraints.WEST;
getContentPane().add(fieldVarName, gridBagConstraints);

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 1;
gridBagConstraints.anchor = java.awt.GridBagConstraints.EAST;
getContentPane().add(jLabel6, gridBagConstraints);


buttonSubmit.setText("Submit");

```

```

buttonSubmit.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        buttonSubmitActionPerformed(evt);
    }
});

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 1;
gridBagConstraints.gridy = 10;
gridBagConstraints.anchor = java.awt.GridBagConstraints.WEST;
getContentPane().add(buttonSubmit, gridBagConstraints);

taComments.setColumns(20);
taComments.setRows(5);
sPComments.setViewportViewView(taComments);

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 1;
gridBagConstraints.gridy = 7;
gridBagConstraints.gridwidth = java.awt.GridBagConstraints.REMAINDER;
gridBagConstraints.anchor = java.awt.GridBagConstraints.WEST;
getContentPane().add(sPComments, gridBagConstraints);

cbArray.setText("Array");
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 1;
gridBagConstraints.gridy = 3;
gridBagConstraints.anchor = java.awt.GridBagConstraints.WEST;
getContentPane().add(cbArray, gridBagConstraints);

jLabel4.setText("Type: ");
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 1;
gridBagConstraints.anchor = java.awt.GridBagConstraints.EAST;
getContentPane().add(jLabel4, gridBagConstraints);

```

```

        comboType.setModel(new DefaultComboBoxModel(AllTypes.valuesNoVoid()));
        comboType.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                comboTypeActionPerformed(evt);
            }
        });
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 1;
        gridBagConstraints.gridy = 1;
        gridBagConstraints.anchor = java.awt.GridBagConstraints.WEST;
        getContentPane().add(comboType, gridBagConstraints);

        this.fieldObName.setEditable(false);
        fieldObName.setToolTipText("Object name");
        fieldObName.setPreferredSize(new java.awt.Dimension(100, 22));
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 1;
        gridBagConstraints.gridy = 2;
        getContentPane().add(fieldObName, gridBagConstraints);

        pack();
    }

    private void comboTypeActionPerformed(java.awt.event.ActionEvent evt) {

        AllTypes selectedType = (AllTypes) comboType.getSelectedItem();

        if (selectedType.equals(AllTypes.OBJECT)) {
            fieldObName.setEditable(true);
        } else {
            fieldObName.setEditable(false);
        }
    }

    /**@Override

```



```

/**
 *
 * @param evt
 */
public void buttonSubmitActionPerformed(ActionEvent evt) {

    boolean isArray = getCbArray().isSelected();

    String paramName = "";

    String paramType = "";

    //check if fieldObName is same as any import
    if (isValidObName(getComboType(), getFieldObName()) &&
isName(getFieldVarName().getText())){

        AllTypes paramTypeSelected = (AllTypes)getComboType().getSelectedItem();

        paramName = getFieldVarName().getText();

        if(paramTypeSelected.equals(AllTypes.OBJECT)){

            paramType = getFieldObName().getText().replaceAll(" ", "");

        } else {

            paramType = paramTypeSelected.getString();

        }

        param = new Parameter(paramName, paramType, isArray, getImports());

        if (this.methh.isValidParam(param)){

            param.setName(paramName);

            param.setType(paramType);

            param.setIsArray(isArray);

            param.setLibImports(getImportsArrayList());

            this.methh.addParam(param);

            System.out.println("AddParameterDialog method parameters");

            System.out.println(methh.getParameters());

            dispose();

        }
    }
}

```

```

    } else {
        JOptionPane.showMessageDialog(null,
            "Invalid parameter name or type",
            "Warning",
            JOptionPane.WARNING_MESSAGE);
    }

}

/**
 *
 * @return cbArray
 */
public JCheckBox getCbArray() {
    return cbArray;
}

/**
 *
 * @return comboType
 */
public JComboBox<String> getComboType() {
    return comboType;
}

/**
 *
 * @return fieldObName
 */
public JTextField getFieldObName() {
    return fieldObName;
}

/**

```

```

*
* @return fieldVarName
*/
public JTextField getFieldVarName() {
    return fieldVarName;
}

/**
*
* @return taComments
*/
public JTextArea getTaComments() {
    return taComments;
}

/**
*
* @return ArrayList of imports entered
*/
public ArrayList<String> getImports() {
    return imports;
}

/**
*
* @return labelName
*/
public JLabel getLabelName() {
    return labelName;
}

/**
*
* @return labelComments
*/
public JLabel getLabelComments() {

```

```

        return labelComments;
    }

    /**
     *
     * @return currently not implemented
     */
    @Override
    public String generateComments() {
        return "";
    }
}

```

#### 7.1.1.4 AddVariableDialog

```

package umldc.GUI;

import umldc.data.ClassObject;
import umldc.data.VisibilityType;
import umldc.data.AllTypes;
import umldc.data.Attribute;
import java.util.ArrayList;
import static javax.lang.model.SourceVersion.isName;
import javax.swing.DefaultComboBoxModel;
import javax.swing.JCheckBox;
import javax.swing.JComboBox;
import javax.swing.JDialog;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import javax.swing.SwingUtilities;

/**
 *
 * @author rayha

```

```

*/

public class AddVariableDialog extends MyJDialog {

    // Created by NetBeans GUI Builder
    private javax.swing.JButton buttonSubmit;
    private javax.swing.JCheckBox cbArray;
    //private javax.swing.JCheckBox cbIgnoreImports;
    private javax.swing.JCheckBox cbRO;
    private javax.swing.JCheckBox cbStatic;
    private javax.swing.JComboBox<String> comboType;
    private javax.swing.JComboBox<String> comboVisibility;
    private javax.swing.JTextField fieldObName;
    private javax.swing.JTextField fieldVarName;
    private javax.swing.JLabel labelName;
    private javax.swing.JLabel labelVis;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel labelComments;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JScrollPane sPComments;
    private javax.swing.JTextField jTextField3;
    private javax.swing.JTextArea taComments;
    // End of variables declaration

    private ArrayList<String> imports = new ArrayList<String>();
    private String readOnly = "";
    private String staticVar = "";

    /**
     *
     * @param parent parent that ran this dialog
     * @param modal true or false if this dialog is modal
     * @param cOb the class object which should contain the attribute
     */
    public AddVariableDialog(java.awt.Frame parent, boolean modal, ClassObject cOb) {

```

```

        super(parent, modal, cOb);

    }

    // @Override

    /**
     *
     * @param evt
     */
    public void buttonSubmitActionPerformed(java.awt.event.ActionEvent evt) {

        boolean isArray = getCbArray().isSelected();
        boolean isStatic = getCbStatic().isSelected();
        boolean isRO = getCbRO().isSelected();
        VisibilityType visType = (VisibilityType)getComboVisibility().getSelectedItem();
        String varName = "";
        String varType = "";
        String comms = generateComments();

        //check if fieldObName is same as any import
        if (isValidObName(getComboType(), getFieldObName()) &&
            isName(getFieldVarName().getText())){

            AllTypes varTypeSelected = (AllTypes)getComboType().getSelectedItem();
            varName = getFieldVarName().getText();
            if(varTypeSelected.equals(AllTypes.OBJECT)){
                varType = getFieldObName().getText().replaceAll(" ", "");
            } else {
                varType = varTypeSelected.getString();
            }
        }

        //addClassDial.addAttribute(new Attribute(varName, visType, varType, isArray,
        isStatic, isRO, imports, comms));

        Attribute att = new Attribute(varName, visType, varType, isArray, isStatic,
        isRO, getImports(), comms);
    }

```

```

        if (!classOb.attributeExists(att)){
            classOb.addAttribute(new Attribute(varName, visType, varType, isArray,
isStatic, isRO, getImports(), comms));
            dispose();
        }

    } else {
        JOptionPane.showMessageDialog(this,
            "Invalid variable name or type",
            "Warning",
            JOptionPane.WARNING_MESSAGE);
    }

}

/**
 *
 */
public void initComponents() {
    labelName = new javax.swing.JLabel();
    labelVis = new javax.swing.JLabel();
    jLabel3 = new javax.swing.JLabel();
    labelComments = new javax.swing.JLabel();
    fieldVarName = new javax.swing.JTextField();
    jLabel6 = new javax.swing.JLabel();
    cbRO = new javax.swing.JCheckBox();
    buttonSubmit = new javax.swing.JButton();
    sPComments = new javax.swing.JScrollPane();
    taComments = new javax.swing.JTextArea();
//    jTextField3 = new javax.swing.JTextField();
    cbStatic = new javax.swing.JCheckBox();
    cbArray = new javax.swing.JCheckBox();
    jLabel4 = new javax.swing.JLabel();

```

```

comboType = new javax.swing.JComboBox<>();

fieldObName = new javax.swing.JTextField();

jLabel7 = new javax.swing.JLabel();

comboVisibility = new javax.swing.JComboBox<>();

setTitle("Add Variable");

labelName.setText("Variable Name: ");
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 0;
gridBagConstraints.anchor = java.awt.GridBagConstraints.EAST;
getContentPane().add(labelName, gridBagConstraints);

labelVis.setText("Visibility: ");
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 4;
gridBagConstraints.anchor = java.awt.GridBagConstraints.EAST;
getContentPane().add(labelVis, gridBagConstraints);

jLabel3.setText("Object Name: ");
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 2;
gridBagConstraints.anchor = java.awt.GridBagConstraints.EAST;
getContentPane().add(jLabel3, gridBagConstraints);

labelComments.setText("Additional Comments: ");
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 7;
gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHEAST;
getContentPane().add(labelComments, gridBagConstraints);

fieldVarName.setPreferredSize(new java.awt.Dimension(100, 22));

```



```

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridwidth = java.awt.GridBagConstraints.REMAINDER;
gridBagConstraints.anchor = java.awt.GridBagConstraints.WEST;
getContentPane().add(fieldVarName, gridBagConstraints);
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 1;
gridBagConstraints.anchor = java.awt.GridBagConstraints.EAST;
getContentPane().add(jLabel6, gridBagConstraints);

cbRO.setText("Read Only");
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 1;
gridBagConstraints.gridy = 5;
gridBagConstraints.anchor = java.awt.GridBagConstraints.WEST;
getContentPane().add(cbRO, gridBagConstraints);

buttonSubmit.setText("Submit");
buttonSubmit.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        buttonSubmitActionPerformed(evt);
    }
});

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 1;
gridBagConstraints.gridy = 10;
gridBagConstraints.anchor = java.awt.GridBagConstraints.WEST;
getContentPane().add(buttonSubmit, gridBagConstraints);

taComments.setColumns(20);
taComments.setRows(5);
sPComments.setViewportViewView(taComments);

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 1;
gridBagConstraints.gridy = 7;

```

```

gridBagConstraints.gridwidth = java.awt.GridBagConstraints.REMAINDER;
gridBagConstraints.anchor = java.awt.GridBagConstraints.WEST;
getContentPane().add(sPComments, gridBagConstraints);

cbStatic.setText("Static");
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 2;
gridBagConstraints.gridy = 5;
gridBagConstraints.anchor = java.awt.GridBagConstraints.WEST;
getContentPane().add(cbStatic, gridBagConstraints);

cbArray.setText("Array");
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 1;
gridBagConstraints.gridy = 3;
gridBagConstraints.anchor = java.awt.GridBagConstraints.WEST;
getContentPane().add(cbArray, gridBagConstraints);

jLabel4.setText("Type: ");
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 1;
gridBagConstraints.anchor = java.awt.GridBagConstraints.EAST;
getContentPane().add(jLabel4, gridBagConstraints);

comboType.setModel(new DefaultComboBoxModel(AllTypes.valuesNoVoid()));
comboType.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        comboTypeActionPerformed(evt);
    }
});

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 1;
gridBagConstraints.gridy = 1;
gridBagConstraints.anchor = java.awt.GridBagConstraints.WEST;
getContentPane().add(comboType, gridBagConstraints);

```

```

        this.fieldObName.setEditable(false);
        fieldObName.setToolTipText("Object name");
        fieldObName.setPreferredSize(new java.awt.Dimension(100, 22));
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 1;
        gridBagConstraints.gridy = 2;
        getContentPane().add(fieldObName, gridBagConstraints);

        comboVisibility.setModel(new DefaultComboBoxModel(VisibilityType.values()));
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 1;
        gridBagConstraints.gridy = 4;
        gridBagConstraints.anchor = java.awt.GridBagConstraints.WEST;
        getContentPane().add(comboVisibility, gridBagConstraints);

        pack();
    }

    /**
     *
     * @param evt
     */
    protected void comboTypeActionPerformed(java.awt.event.ActionEvent evt) {

        AllTypes selectedType = (AllTypes) comboType.getSelectedItem();

        //If OBJECT is selected then enable the fieldObName field
        if (selectedType.equals(AllTypes.OBJECT)) {
            fieldObName.setEditable(true);
        } else {
            fieldObName.setEditable(false);
        }
    }

    /**

```

```

*
* @return cbArray
*/
public JCheckBox getCbArray() {
    return cbArray;
}

/**
*
* @return cbRO
*/
public JCheckBox getCbRO() {
    return cbRO;
}

/**
*
* @return cbStatic
*/
public JCheckBox getCbStatic() {
    return cbStatic;
}

/**
*
* @return comboType
*/
public JComboBox<String> getComboType() {
    return comboType;
}

/**
*
* @return comboVisibility
*/
public JComboBox<String> getComboVisibility() {

```

```

        return comboVisibility;
    }

    /**
     *
     * @return fieldObName
     */
    public JTextField getFieldObName() {
        return fieldObName;
    }

    /**
     *
     * @return fieldVarName
     */
    public JTextField getFieldVarName() {
        return fieldVarName;
    }

    /**
     *
     * @return taComments
     */
    public JTextArea getTaComments() {
        return taComments;
    }

    /**
     *
     * @return imports
     */
    public ArrayList<String> getImports() {
        return imports;
    }

    /**

```

```

    *

    * @return labelName
    */

    public JLabel getLabelName() {

        return labelName;

    }


    /**
     *
     * @return labelVis
     */

    public JLabel getLabelVis() {

        return labelVis;

    }


    /**
     *
     * @return labelComments
     */

    public JLabel getLabelComments() {

        return labelComments;

    }


    /**
     *
     * @return generates a string of comments based on the users input of static, readonly and
    comments
     */

    @Override

    public String generateComments() {

        String roStatic = "";

        String comments = taComments.getText().trim();

        if (cbRO.isSelected() && (cbStatic.isSelected() || !comments.isEmpty())) {

            readOnly = "readOnly, ";

        } else if (cbRO.isSelected()) {

            readOnly = "readOnly";

        }

    }

```

```

        if (cbStatic.isSelected() && !comments.isEmpty()) {
            staticVar = "static, ";
        } else if (cbStatic.isSelected()) {
            staticVar = "static";
        }

        roStatic = readOnly + staticVar + comments;

        return roStatic;
    }
}

```

#### 7.1.1.5 - DrawingPanel

```

package umldc.GUI;

import umldc.data.CMethod;
import umldc.data.ClassObject;
import umldc.data.Attribute;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.FontMetrics;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.LayoutManager;
import java.awt.Rectangle;
import java.awt.geom.Line2D;
import java.awt.geom.Point2D;
import java.awt.geom.Rectangle2D;
import java.util.ArrayList;
import javafx.scene.shape.Line;
import javax.swing.JPanel;

/**
 *
 * @author rayha

```

```

*/

public class DrawingPanel extends JPanel{

    private ArrayList<ClassObject> classes;

    Point2D.Double point;

    ArrayList<Rectangle2D.Double> classBoxes = new ArrayList<Rectangle2D.Double>();

    /**
     *
     * @param classes the ArrayList of classes that have been added by the user
     */
    public DrawingPanel(ArrayList<ClassObject> classes) {

        this.classes = classes;

        setBackground(Color.WHITE);

    }

    @Override
    protected void paintComponent(Graphics g) {

        super.paintComponent(g);

        Graphics2D g2 = (Graphics2D) g;

        point = new Point2D.Double(10, 10);

        for (ClassObject clazz : classes){

            drawClass(g2, point, clazz);

        }

        this.setPreferredSize(this.getArea());

        this.revalidate();

    }

```



```

/**
 * Draws a single class object as a diagram object
 * @param g2 graphics 2d object
 * @param point the point that should act as a pen when drawing the diagram
 * @param c a class object
 */
private void drawClass(Graphics2D g2, Point2D.Double point, ClassObject c){

    String className = c.getClassName();

    FontMetrics metrics = g2.getFontMetrics();

    int areaX = 0;

    int areaY = 0;

    int longestString = getLongestStringWidth(g2, c);

    Rectangle2D.Double classBox = new Rectangle2D.Double(point.x, point.y, longestString +
20, getBoxHeight(c));

    classBoxes.add(classBox);

    g2.draw(classBox);

    int stringX = (int)((classBox.width - metrics.stringWidth(className)) / 2) +10;

    point.setLocation(point.x, point.y + 20);

    g2.drawString(className, (int) stringX, (int) point.y);

    if (!c.getAttributes().isEmpty() || !c.getMethods().isEmpty()) {

        point.setLocation(point.x, point.y + 10);

        Line2D.Double lineSeparator = new Line2D.Double(point.x, point.y, classBox.width +
point.x, point.y);

        g2.draw(lineSeparator);

        //ArrayList<String> variables = new ArrayList<String>();

        point.setLocation(point.x, point.y + 20);

```

```

        for (Attribute var : c.getAttributes()) {
            String variable = var.toString();
            g2.drawString(variable, (int) point.x + 10, (int) point.y);

            point.setLocation(point.x, point.y + 20);
        }

        if (!c.getAttributes().isEmpty() && !c.getMethods().isEmpty()){
            point.setLocation(point.x, point.y - 10);
            lineSeperator = new Line2D.Double(point.x, point.y, classBox.width + point.x,
point.y);

            g2.draw(lineSeperator);
            point.setLocation(point.x, point.y + 20);
        }

        for (CMethod meth : c.getMethods()) {
            String method = meth.toString();
            g2.drawString(method, (int) point.x + 10, (int) point.y);

            point.setLocation(point.x, point.y + 20);
        }

    }

    this.point.setLocation(classBox.x, classBox.height + classBox.y + 20);
}

/**
 * Gets the longest string inside a classObject
 * @param g2 a Graphics2D object
 * @param c a class object
 *
 *
 */
private int getLongestStringWidth(Graphics2D g2, ClassObject c){

```

```

//List of all strings of a class
ArrayList<String> strings = getAllStrings(c);
FontMetrics metrics = g2.getFontMetrics();

int longestStringSize = 0;

//set longestStringSize to the size of the longest string
for (String str : strings){
    if (metrics.stringWidth(str)> longestStringSize){
        longestStringSize = metrics.stringWidth(str);
    }
}

return longestStringSize;
}

//Get a list of all displayable strings of a ClassObject (class name,
//methods and variables)
private ArrayList<String> getAllStrings(ClassObject c){
    ArrayList<String> strings = new ArrayList<String>();

    strings.add(c.getClassName());

    for (Attribute var : c.getAttributes()) {
        strings.add(var.toString());
    }

    for (CMethod meth : c.getMethods()) {
        strings.add(meth.toString());
    }

    return strings;
}

//get the height of a class diagram box
private int getBoxHeight(ClassObject c){

```

```

        ArrayList<String> strings = getAllStrings(c);

        int height = 20;

        if (!c.getAttributes().isEmpty() && !c.getMethods().isEmpty()){
            height += 10;
        }

        for (String str : strings){
            height += 20;
        }

        return height;
    }

    /**
     *
     * @return the area that should be drawn on based on the size of each class diagram object
     */
    public Dimension getArea(){

        int furthestX = 0;
        int furthestY = 0;

        for (Rectangle2D.Double rect : classBoxes){
            if (rect.x + rect.width > furthestX){
                furthestX = (int)(rect.x + rect.width);
            }
            if (rect.y + rect.height > furthestY){
                furthestY = (int)(rect.y + rect.height);
            }
        }

        Dimension area = new Dimension(furthestX + 20, furthestY + 20);

        return area;
    }

```

```

    }

}

```

#### 7.1.1.6 - EditMethodDialog

```

package umldc.GUI;

import umldc.data.ClassObject;
import umldc.data.CMethod;
import umldc.data.Parameter;
import umldc.data.MethodMemento;
import umldc.data.VisibilityType;
import umldc.data.AllTypes;
import java.awt.Frame;
import java.awt.event.ActionEvent;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.util.ArrayList;
import java.util.concurrent.CopyOnWriteArrayList;
import static javax.lang.model.SourceVersion.isName;
import javax.swing.DefaultListModel;
import javax.swing.JDialog;
import javax.swing.JOptionPane;

/**
 *
 * @author rayha
 */
public class EditMethodDialog extends AddMethodDialog{

    private int selectedIndex;

    private CopyOnWriteArrayList<CMethod> methods = classOb.getMethods();

    private CMethod selectedMeth;

    private CMethod editedMethod;

    private ArrayList<Parameter> paramsToAdd;

    private MethodMemento _memento;

```

```

/**
 *
 * @param parent parent that ran this dialog
 * @param modal true or false if this dialog is modal
 * @param cOb the class object which should contain the method
 * @param selectedIndex index of the selected method
 */
public EditMethodDialog(Frame parent, boolean modal, ClassObject cOb, int selectedIndex) {
    super(parent, modal, cOb);

    editedMethod = new CMethod();
    paramsToAdd = new ArrayList<Parameter>();
    this.selectedIndex = selectedIndex;
    setTitle("Edit Method");
    this.selectedMeth= methods.get(selectedIndex);
    System.out.println(selectedMeth.hashCode());
    System.out.println(editedMethod.hashCode());
    System.out.println("Selected Method Params");
    System.out.println(selectedMeth.getParameters());

    saveClass(selectedMeth.saveStateToMemento());
    this.addWindowListener(new WindowAdapter() {

        public void windowClosed(WindowEvent e) {
            selectedMeth.removeObserver((EditMethodDialog)e.getWindow());
            System.out.println("Before "+selectedMeth.getState());
            selectedMeth.getStateFromMemento(_memento);
            System.out.println("After "+selectedMeth.getState());
            System.out.println("MementoState "+_memento.getState());
        }
    });

    this.editedMethod.registerObserver(this);
    setInitValues();
}

```

```

        System.out.println(selectedMeth);

    }

    /**
     *
     * @param evt
     */
    @Override
    public void buttonSubmitActionPerformed(java.awt.event.ActionEvent evt) {

        VisibilityType visType = (VisibilityType)getComboVisibility().getSelectedItem();
        boolean isStatic = getCbStatic().isSelected();

        String methName = "";
        String methType = "";
        String comms = generateComments();

        //check if fieldObName is same as any import
        if (isValidObName(getComboType(), getFieldObName()) &&
isName(getFieldMethName().getText())){

            AllTypes paramTypeSelected = (AllTypes)getComboType().getSelectedItem();

            methName = getFieldMethName().getText();
            if (paramTypeSelected.equals(AllTypes.OBJECT)) {
                methType = getFieldObName().getText().replaceAll(" ", "");
            } else {
                methType = paramTypeSelected.getString();
            }

            editedMethod.setName(methName);
            editedMethod.setVisibility(visType);
            editedMethod.setIsStatic(isStatic);
            editedMethod.setLibImports(getImportsArrayList());
            editedMethod.setComments(comms);
            editedMethod.setType(methType);

```

```

        //CMethod editedMethod = new CMethod(methName, visType, methType, isStatic,
getImportsArrayList(), comms);

        //editMethod(methName, visType, methType, meth.getParameters(), isStatic ,
getImportsArrayList(), comms);

        classOb.editMethod(editedMethod, selectedIndex);

        System.out.println("Parameters after editing");

        System.out.println(selectedMeth.getParameters());

        dispose();

    } else {

        JOptionPane.showMessageDialog(this,

            "Invalid variable name or type",

            "Warning",

            JOptionPane.WARNING_MESSAGE);

    }

}

/**
 *
 * @param evt
 */
@Override
protected void buttonAddParamActionPerformed(ActionEvent evt) {

    MyJDialog addParamDialog = new AddParamDialog(null, true, editedMethod);

    addParamDialog.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);

    addParamDialog.setLocationRelativeTo(this);

    addParamDialog.pack();

    addParamDialog.setVisible(true);

    System.out.println("AddParamActionPerformed");

    System.out.println(selectedMeth.getParameters());

    System.out.println(editedMethod.getParameters());

}

```



```

/**
 *
 * @param evt
 */
@Override
protected void buttonEditParamActionPerformed(ActionEvent evt) {

    if ( !editedMethod.getParameters().isEmpty() && !paramListView.isSelectionEmpty()) {

        int selectedParamIndex = paramListView.getSelectedIndex();

        MyJDialog editParamDialog = new EditParamDialog(null, true, editedMethod,
selectedParamIndex);

        editParamDialog.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);

        editParamDialog.setLocationRelativeTo(this);

        editParamDialog.pack();

        editParamDialog.setVisible(true);

    }
}

/**
 *
 * @param evt
 */
@Override
protected void buttonRemParamActionPerformed(ActionEvent evt) {

    editedMethod.removeParam(selectedIndex);

}

/**
 *
 */
public void setInitValues() {

    //paramListModel = new DefaultListModel();

    getFieldMethName().setText(selectedMeth.getName());

    getComboType().setSelectedItem(selectedMeth.getType());

    if(selectedMeth.getType().equals(AllTypes.OBJECT)){

```

```

        getFieldObName().setText(getFieldObName().getText());
    }

    getComboVisibility().setSelectedItem(selectedMeth.getVisibility());
    getCbStatic().setSelected(selectedMeth.isStatic());
    getTaImports().setText(selectedMeth.importsToString());
    getTaComments().setText(selectedMeth.getComments());
    editedMethod.setParameters(selectedMeth.getParameters());

    pack();
}

/**
 *
 * @param meth the method that should be updated
 */
@Override
public void update(CMethod meth) {
    editedMethod = meth;

    paramListModel.clear();

    CopyOnWriteArrayList<Parameter> parameters = editedMethod.getParameters();

    if (!parameters.isEmpty()) {
        for (Parameter param : parameters) {
            paramListModel.addElement(param);
        }
    }
}

/**
 *
 * @param state save the current state of a method as a memento
 */

```

```

    public void saveClass(MethodMemento state){
        _memento = state;
    }

    /**
     *
     * @return get the state of a method from the memento
     */
    public MethodMemento retrieveClass(){
        return _memento;
    }
}

```

#### 7.1.1.7 - EditParamDialog

```

package umldc.GUI;

import umldc.data.CMethod;
import umldc.data.Parameter;
import umldc.data.AllTypes;
import java.awt.Frame;
import java.util.ArrayList;
import java.util.concurrent.CopyOnWriteArrayList;
import static javax.lang.model.SourceVersion.isName;
import javax.swing.DefaultListModel;
import javax.swing.JList;
import javax.swing.JOptionPane;

/**
 *
 * @author rayha
 */
public class EditParamDialog extends AddParamDialog{

    private int selectedIndex;

    private CMethod meth;

    private CopyOnWriteArrayList<Parameter> params;

```

```

private Parameter param;
private Parameter editedParameter;

/**
 *
 * @param parent
 * @param modal
 * @param meth
 * @param selectedIndex
 */
public EditParamDialog(Frame parent, boolean modal, CMethod meth , int selectedIndex) {
    super(parent, modal, meth);

    this.meth = meth;

    this.params = this.meth.getParameters();
    this.param = params.get(selectedIndex);
    this.selectedIndex = selectedIndex;
    setInitValues();

    editedParameter = new Parameter();

    setTitle("Edit Parameter");

}

/**
 *
 * @param evt
 */
@Override
public void buttonSubmitActionPerformed(java.awt.event.ActionEvent evt) {

    boolean isArray = getCbArray().isSelected();

    String paramName = "";
    String paramType = "";

```

```

String comms = generateComments();

//check if fieldObName is same as any import
if (isValidObName(getComboType(), getFieldObName()) &&
isName(getFieldVarName().getText())){

    AllTypes paramTypeSelected = (AllTypes)getComboType().getSelectedItem();

    paramName = getFieldVarName().getText();
    if (paramTypeSelected.equals(AllTypes.OBJECT)) {
        paramType = getFieldObName().getText().replaceAll(" ", "");

    } else {
        paramType = paramTypeSelected.getString();
    }

    editedParameter.setName(paramName);
    editedParameter.setType(paramType);
    editedParameter.setIsArray(isArray);
    editedParameter.setLibImports(getImportsArrayList());

    meth.editParam(editedParameter, selectedIndex);
    dispose();

} else {
    JOptionPane.showMessageDialog(this,
        "Invalid variable name or type",
        "Warning",
        JOptionPane.WARNING_MESSAGE);
}

}

/**

```

```

*
* @param name name of attribute
* @param varType type of attribute
* @param isArray is it an array
* @paramimps the arraylist of imports
* @param comms a string of comments
*/

public void editAttribute(String name, String varType, boolean isArray, ArrayList<String>
imps, String comms){

    String paramName = getFieldVarName().getText();

    boolean exists = false;

    for (int count = 0; count < params.size(); count++) {

        if (count == selectedIndex) {

            try {

                count++;

            } catch (ArrayIndexOutOfBoundsException e) {

                //Do nothing

            }

        }

        try {

            if (params.get(count).getName().equals(paramName)) {

                exists = true;

            }

        } catch (ArrayIndexOutOfBoundsException e) {

            //Do nothing

        }

    }

    if (exists){

        JOptionPane.showMessageDialog(null,

```

```

        "Parameter name already exists",
        "Warning",
        JOptionPane.WARNING_MESSAGE);
    } else {
        param.setName(paramName);
        param.setType(varType);
        param.setIsArray(isArray);
        param.setLibImports(imps);
        params = meth.getParameters();
    }
}

/**
 * set the initial values of the parameter that should be edited
 */
public void setInitValues() {
    getFieldVarName().setText(param.getName());
    getComboType().setSelectedItem(param.getType());

    if(param.getType().equals(AllTypes.OBJECT)){
        getFieldObName().setText(getFieldObName().getText());
    }

    getCbArray().setSelected(param.isArray());
    getTaImports().setText(param.importsToString());
}
}

```

#### 7.1.1.8 - EditVariableDialog

```

package umldc.GUI;

import umldc.data.ClassObject;
import umldc.data.VisibilityType;
import umldc.data.AllTypes;
import umldc.data.Attribute;
import java.util.ArrayList;

```

```

import java.util.concurrent.CopyOnWriteArrayList;

import static javax.lang.model.SourceVersion.isName;

import javax.swing.JOptionPane;

/**
 *
 * @author rayha
 */

public class EditVariableDialog extends AddVariableDialog{

    private int selectedIndex;

    private CopyOnWriteArrayList<Attribute> atts = classOb.getAttributes();

    private Attribute att;// = atts.get(selectedIndex);

    /**
     *
     * @param parent parent that ran this dialog
     * @param modal true or false if this dialog is modal
     * @param cOb the class object which should contain the attribute
     * @param selectedIndex index of the selected attribute
     */

    public EditVariableDialog(java.awt.Frame parent, boolean modal, ClassObject cOb, int
selectedIndex) {

        super(parent, modal, cOb);

        this.selectedIndex = selectedIndex;

        att = atts.get(selectedIndex);

        setInitValues();

    }

    /**
     *
     * @param evt
     */

    @Override

    public void buttonSubmitActionPerformed(java.awt.event.ActionEvent evt) {

```



```

        boolean isArray = getCbArray().isSelected();

        boolean isStatic = getCbStatic().isSelected();

        boolean isRO = getCbRO().isSelected();

        VisibilityType visType = (VisibilityType)getComboVisibility().getSelectedItem();

        String varName = "";

        String varType = "";

        String comms = generateComments();

        //check if fieldObName is same as any import
        if (isValidObName(getComboType(), getFieldObName()) &&
isName(getFieldVarName().getText())){

            AllTypes varTypeSelected = (AllTypes)getComboType().getSelectedItem();

            varName = getFieldVarName().getText();

            if(varTypeSelected.equals(AllTypes.OBJECT)){

                varType = getFieldObName().getText().replaceAll(" ", "");

            } else {

                varType = varTypeSelected.getString();

            }

            Attribute editedAtt = new Attribute(varName, visType, varType, isArray, isStatic,
isRO, getImportsArrayList(), comms);

            classOb.editAttribute(editedAtt, selectedIndex);

            dispose();

        } else {

            JOptionPane.showMessageDialog(this,

                "Invalid variable name or type",

                "Warning",

                JOptionPane.WARNING_MESSAGE);

        }

    }

    /**

```

```

        * sets the initial values of each field based on the selected attribute
        */
    public void setInitValues() {
        getFieldVarName().setText(att.getName());
        getComboType().setSelectedItem(att.getType());

        if(att.getType().equals(AllTypes.OBJECT)){
            getFieldObName().setText(getFieldObName().getText());
        }

        getCbArray().setSelected(att.isArray());
        getComboVisibility().setSelectedItem(att.getVisibility());
        getCbRO().setSelected(att.isReadOnly());
        getCbStatic().setSelected(att.isStatic());
        getTaImports().setText(att.importsToString());
        getTaComments().setText(att.getComments());
    }
}

```

#### 7.1.1.9 - MyGUI

```

package umldc.GUI;

import umldc.data.CMethod;
import umldc.data.ClassObject;
import umldc.data.Parameter;
import umldc.data.Attribute;
import java.awt.BorderLayout;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.Writer;
import java.util.ArrayList;

```

```

import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.*;

/**
 *
 * @author rayha
 */
public class MyGUI extends JFrame {

    DrawingPanel panelCenter;

    JScrollPane scrollPane;

    private ArrayList<ClassObject> classes = new ArrayList<ClassObject>();

    /**
     *
     */
    public MyGUI() {

        super("UML Diagram Creation and Conversion Tool");

        createView();
    }

    //Initialize UI Components

    /**
     * add components the the GUI
     */
    public void createView(){

        //Create Main content pane with Border Layout
        Container contentPane = getContentPane();
        contentPane.setLayout(new BorderLayout());
        //

```

```

//Create Panel for West with GridBagLayout
JPanel panelWest = new JPanel();
panelWest.setLayout(new GridBagLayout());
panelWest.setBackground(new Color(200, 200, 200));
contentPane.add(panelWest, BorderLayout.WEST);

//Set Layout?
//scrollableDesktop.setBackground(new Color(255,255,255));
panelCenter = new DrawingPanel(classes);
// contentPane.add(panelCenter, BorderLayout.CENTER);
scrollPane = new JScrollPane(panelCenter);

scrollPane.setPreferredSize(new Dimension(200,200));
add(scrollPane, BorderLayout.CENTER);

//JMenuBar
createMenu();
//

//Create Buttons on West Panel
createToolButtons(panelWest);

//setLayout(new BorderLayout());
setExtendedState(JFrame.MAXIMIZED_BOTH);
setMinimumSize(new Dimension(300, 200));
setVisible(true);
}

private void createMenu(){
    JMenuBar menubar = new JMenuBar();
    setJMenuBar(menubar);

    JMenu fileMenu = new JMenu("File");
    menubar.add(fileMenu);

```

```

JMenuItem exportItem = new JMenuItem("Create Code...");
exportItem.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {

        if (!classes.isEmpty()) {
            JFileChooser folderChooser = new JFileChooser();

            folderChooser.setAcceptAllFileFilterUsed(false);
            folderChooser.setCurrentDirectory(new java.io.File("\\"));
            folderChooser.setDialogTitle("Choose Folder to save files");

folderChooser.setFileSelectionMode(javax.swing.JFileChooser.DIRECTORIES_ONLY);

            int selected = folderChooser.showDialog(null, "OK");

            File saveDir= null;

            if (selected == JFileChooser.APPROVE_OPTION){
                saveDir = folderChooser.getSelectedFile();
                createClassFiles(saveDir);
            }
        } else {
            JOptionPane.showMessageDialog(null,
                "There are no classes",
                "Attention",
                JOptionPane.WARNING_MESSAGE);
        }

    }
});

fileMenu.add(exportItem);
}

```

//genertae the class file, takes a directory for output as an argument

```

private void createClassFiles(File dir) {

    if (!classes.isEmpty()){

        for (ClassObject c : classes) {

            File classFile = new File(dir.getPath()+ "\\\" + c.getClassName() + ".java");

            try {

                classFile.createNewFile();

            } catch (IOException ex) {

                Logger.getLogger(MyGUI.class.getName()).log(Level.SEVERE, null, ex);

            }

            try {

                BufferedWriter writer = new BufferedWriter(new FileWriter(classFile));

                writer.write("public class " + c.getClassName() + " {\n\n\t");

                generateVarCode(writer, c);

                generateMethCode(writer, c);

                writer.write("\n}");

                writer.close();

            } catch (IOException ex) {

                Logger.getLogger(MyGUI.class.getName()).log(Level.SEVERE, null, ex);

            }

        }

    }

}

//generate code for a variable
private void generateVarCode(Writer writer, ClassObject c) throws IOException {

    for (Attribute att : c.getAttributes()) {

        String array = "";

        if (att.isArray()) {

            array = "[]";

        }

    }

}

```

```

        String stat = "";
        if (att.isStatic()) {
            stat = "static ";
        }

        String rO = "";
        if (att.isReadOnly()) {
            rO = "final ";
        }

        writer.write(att.getVisibility().toString() + " " + stat + rO + att.getType() + array
+ " " + att.getName() + ";\n\t");
    }

    writer.write("\n");
}

//generate code for a method
private void generateMethCode(Writer writer, ClassObject c) throws IOException{
    for (CMethod meth : c.getMethods()){

        writer.write("\t");

        String stat = "";
        if (meth.isStatic()){
            stat = "static ";
        }

        String parameters = "(";
        for (Parameter param : meth.getParameters()){
            String array = "";
            if (param.isArray()){
                array = "[";
            }

            String paramString = param.getType() + array + " " + param.getName() + ", ";

```

```

        parameters += paramString;

    }

    if (parameters.endsWith(", ")){

        int index = parameters.lastIndexOf(", ");

        if (index >= 0){

            parameters = parameters.substring(0, index);

        }

    }

    parameters += " ";

    writer.write(meth.getVisibility().toString() + " " + stat + meth.getType() + " " +
meth.getName() + parameters + "{\n\n\t");

    if (!meth.getType().equals("void")){

        //writer.write(meth.getType() + "tempVar;\n\n");

        createTempVar(writer, meth);

        writer.write("\t\treturn tempVar;");

    }

    writer.write("\n\t}\n\n");

}

}

private void createTempVar(Writer writer, CMethod meth) throws IOException{

    if (meth.getType().equals("int") || meth.getType().equals("byte") ||
meth.getType().equals("short") || meth.getType().equals("double") ||
meth.getType().equals("long") || meth.getType().equals("float") ||
meth.getType().equals("char")){

        writer.write("\t" + meth.getType() + " tempVar = 0;\n\n");
    }
}

```



```

    } else if (meth.getType().equals("String")){
        writer.write("\t" + meth.getType() + " tempVar = \"\";\n\n");
    } else if (meth.getType().equals("boolean")){
        writer.write("\t" + meth.getType() + " tempVar = true;\n\n");
    } else {
        writer.write("\t" + meth.getType() + " tempVar = null;\n\n");
    }
}

//Create the tool buttons
private void createToolButtons(JPanel panel){
    GridBagConstraints constraints = new GridBagConstraints();
    constraints.gridx = 0;
    constraints.gridy = 0;
    constraints.anchor = GridBagConstraints.NORTHWEST;
    constraints.fill = GridBagConstraints.HORIZONTAL;

    JLabel toolLabel = new JLabel("Tools");
    toolLabel.setOpaque(true);
    toolLabel.setBackground(new Color(150, 150, 150));
    toolLabel.setHorizontalAlignment(JLabel.CENTER);
    panel.add(toolLabel, constraints);
    constraints.gridy ++;

    //Add Class Button
    JButton addClassButton = new JButton(new
    ImageIcon(getClass().getResource("resources/addClass.png")));
    addClassButton.setToolTipText("Add Class");
    addClassButton.addActionListener(new ActionListener(){
        //Action to open class dialog
        @Override
        public void actionPerformed(ActionEvent e) {
            //Component component = (Component) e.getSource();
            JFrame frame = (JFrame) SwingUtilities.getRoot(addClassButton);
            AddClassDialog ACDialog = new AddClassDialog(frame, true);
            ACDialog.setLocationRelativeTo(frame);
            ACDialog.setVisible(true);
        }
    });
}

```

```

    }

});

//

//Add Relation Button

JButton addRelButton = new JButton(new
ImageIcon(getClass().getResource("resources/addRel.png")));

addRelButton.setToolTipText("Add Relation");

//

//For Testing

addRelButton.addActionListener(new ActionListener() {

    @Override

    public void actionPerformed(ActionEvent e) {

        System.out.println(classes);

    }

});

panel.add(addClassButton, constraints);

constraints.gridy ++;

constraints.weighty = 1;

panel.add(addRelButton, constraints);

}

private MyGUI getMyGUI() {

    return this;

}

/**
 *
 * @param cOb classObject that should be checked if it exists

```

```

        * @return true if the class already exists
        */
    public boolean classExists(ClassObject cOb){
        boolean exists = false;
        for (ClassObject clazz : classes){
            if (clazz.getClassName().equals(cOb.getClassName())){
                exists = true;
            }
        }
        return exists;
    }

    /**
     *
     * @param cOb classObject to be added to the ArrayList of classObjects
     */
    public void addClass(ClassObject cOb){

        classes.add(cOb);

    }

}

```

#### **7.1.1.10 - MyJDialog**

```

package umldc.GUI;

import umldc.data.CMethod;
import umldc.data.ClassObject;
import umldc.data.AllTypes;
import java.awt.Frame;
import java.util.ArrayList;
import static javax.lang.model.SourceVersion.isName;
import javax.swing.JCheckBox;
import javax.swing.JComboBox;
import javax.swing.JDialog;

```

```

import javax.swing.JOptionPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;

/**
 *
 * @author rayha
 */
public abstract class MyJDialog extends JDialog{

    private boolean ignoreImports = false;
    private ArrayList<String> imports = new ArrayList<String>();
    private javax.swing.JTextArea taImports;
    private javax.swing.JLabel labelImports;
    private javax.swing.JScrollPane sPImports;
    private javax.swing.JCheckBox cbIgnoreImports;

    /**
     *
     */
    protected java.awt.GridBagConstraints gridBagConstraints;

    /**
     *
     */
    protected ClassObject classOb;

    /**
     *
     */
    protected CMethod meth;

    /**
     *
     * @param parent parent that ran this dialog
     * @param modal true or false if this dialog is modal

```

```

    * @param cOb the class object which is being added to or edited
    */
public MyJDialog(Frame parent, boolean modal, ClassObject cOb) {
    super(parent, modal);

    this.classOb = cOb;

    setModal(true);

    setResizable(false);

    setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);

    getContentPane().setLayout(new java.awt.GridBagLayout());

    initComponents();

    initImport();
}

/**
 *
 * @param parent parent that ran this dialog
 * @param modal true or false if this dialog is modal
 */
public MyJDialog(Frame parent, boolean modal) {
    super(parent, modal);

    setModal(true);

    setResizable(false);

    setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);

    getContentPane().setLayout(new java.awt.GridBagLayout());

    initComponents();

    initImport();
}

/**
 *
 * @param parent parent that ran this dialog
 * @param modal true or false if this dialog is modal
 * @param meth method to be added to or edited
 */

```

```

public MyJDialog(Frame parent, boolean modal, CMethod meth) {
    super(parent, modal);
    this.meth = meth;
    setModal(true);
    setResizable(false);
    setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
    getContentPane().setLayout(new java.awt.GridBagLayout());

    initComponents();
    initImport();
}

private void initImport(){

    sPImports = new javax.swing.JScrollPane();
    taImports = new javax.swing.JTextArea();
    labelImports = new javax.swing.JLabel();
    cbIgnoreImports = new javax.swing.JCheckBox();

    labelImports.setText("Imports: ");
    gridBagConstraints = new java.awt.GridBagConstraints();
    gridBagConstraints.gridx = 0;
    gridBagConstraints.gridy = 6;
    gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHEAST;
    getContentPane().add(labelImports, gridBagConstraints);

    taImports.setColumns(20);
    taImports.setRows(5);
    taImports.addFocusListener(new java.awt.event.FocusAdapter() {
        public void focusLost(java.awt.event.FocusEvent evt) {
            taImportsFocusLost(evt);
        }
    });

    sPImports.setViewportViewView(taImports);
}

```

```

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 1;
gridBagConstraints.gridy = 6;
gridBagConstraints.gridwidth = java.awt.GridBagConstraints.REMAINDER;
gridBagConstraints.anchor = java.awt.GridBagConstraints.WEST;
getContentPane().add(sPImports, gridBagConstraints);

cbIgnoreImports.setText("Ignore Imports");
cbIgnoreImports.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        cbIgnoreImportsActionPerformed(evt);
    }
});

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 2;
gridBagConstraints.gridy = 10;
getContentPane().add(cbIgnoreImports, gridBagConstraints);
}

private void taImportsFocusLost(java.awt.event.FocusEvent evt) {
    imports.clear();
    imports = getImportsArrayList();
}

private void cbIgnoreImportsActionPerformed(java.awt.event.ActionEvent evt) {
    if (cbIgnoreImports.isSelected()) {
        ignoreImports = true;
    } else if (!cbIgnoreImports.isSelected()) {
        ignoreImports = false;
    }
}
}

```

```

/**
 * how to initialize components
 */
public abstract void initComponents();

/**
 *
 * @return generate comments based on input
 */
public abstract String generateComments();

private String removeParams (String str){
    String removed = str.replaceAll("(\\s*)(<(.*)>))$", "");
    return removed;
}

private boolean isValidObjParTypes(JTextField fieldObjName) {
    boolean valid = false;
    String objNameNoWhitespace = fieldObjName.getText().replaceAll(" ", "");

    if
(objNameNoWhitespace.matches("(\\w+) (\\s*) ((<((\\s*) (\\w+) (\\s*)) (, (\\s*) (\\w+) (\\s*)) *>) ?) $"))
{// old (\\w+) ((<(.+) (, ?(.+)>)) ?) $
        valid = true;
    }

    return valid;
}

//Checks if the inputted object name is a valid object

/**
 * checks if the object name is valid or not based on a regular expression

```



```

    * @param comboType the JComboBox that contains the type of the attribute/method
    * @param fieldObName the Object name field
    * @return true of false based on whether or not the object name is correct
    */
protected boolean isValidObName(JComboBox comboType, JTextField fieldObName){
    boolean valid = true;

    AllTypes selected = (AllTypes)comboType.getSelectedItem();

    //String removedParams = fieldObName.getText().replaceAll("<(.*?)>$", "");
    String removedParams = removeParams(fieldObName.getText());

    //If the selected Item in the combolist is OBJECT, do the following checks
    if (selected.equals(AllTypes.OBJECT)) {

        if (isName(removeParams(fieldObName.getText())) && hasValidObParTypes(fieldObName)){
//
            //If ignore imports is not checked

            if (ignoreImports) {

                if (isName(removedParams)) {
                    valid = true;
                } else {
                    JOptionPane.showMessageDialog(null,
                        "Please enter a valid object name",
                        "Invalid object name",
                        JOptionPane.WARNING_MESSAGE);
                    valid = false;
                }
            } else {

                //If the text area Imports is not empty
                if (taImports.getText().trim().length() != 0){

```



```

*/
protected ArrayList<String> getImportsArrayList() {
    imports.clear();

    ArrayList<String> impsArray = new ArrayList<String>();

    int start = 0;

    int end;

    String taImportsText = taImports.getText();

    for (int index = 0; index< taImportsText.length(); index++){
        char character = taImportsText.charAt(index);

        if (character == '\n' || taImportsText.length() - 1 == index){
            end = index+1;

            String sub = taImportsText.substring(start, end);

            if (isName(sub.trim())) { //MyUtils.isFullyQualifiedClassname(sub.trim())
                imports.add(sub.trim());
            }

            start = index;
        }
    }

    return impsArray;
}

/**
 *
 * @return taImports
 */
public JTextArea getTaImports() {
    return taImports;
}

/**
 *
 * @param ignoreImports true if to set ignoreImports as true
 */

```

```

        public void setIgnoreImports(boolean ignoreImports) {
            this.ignoreImports = ignoreImports;
        }
    }
}

```

#### 7.1.1.11 - UMLDC

```
package umldc.GUI;
```

```
import javax.swing.JFrame;
```

```

/**
 *
 * @author rayha
 */
public class UMLDC {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        MyGUI gui = new MyGUI();
        gui.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    }

}

```

### 7.1.2 – umldc.data

#### 7.1.2.1 - AllTypes

```
package umldc.data;
```

```

/**
 *
 * @author rayha
 */
public enum AllTypes {

    /**
     * void type

```

```

*/
VOID("void"),

/**
 * int type
 */
INT("int"),

/**
 * String type
 */
STRING("String"),

/**
 * byte type
 */
BYTE("byte"),

/**
 * short type
 */
SHORT("short"),

/**
 * long type
 */
LONG("long"),

/**
 * float type
 */
FLOAT("float"),

/**
 * double type
 */

```

```

DOUBLE("double"),

/**
 * boolean type
 */
BOOLEAN("boolean"),

/**
 * char type
 */
CHAR("char"),

/**
 * object type
 */
OBJECT("object");

private String str;

AllTypes(String nm){
    str = nm;
}

//Return string version of enum

/**
 *
 * @return string version of enum
 */
public String getString() {
    return str;
}

/**
 *
 * @return all enum values except void (for attributes)

```

```

    */

    public static AllTypes[] valuesNoVoid(){
        int index = 0;

        AllTypes types[] = new AllTypes[10];

        for (AllTypes type: AllTypes.values()){
            if(type!=AllTypes.VOID){
                types[index] = type;
                index ++;
            }
        }

        return types;
    }
}

```

### 7.1.2.2 - Attribute

```

package umldc.data;

import java.util.ArrayList;

/**
 *
 * @author rayha
 */
public class Attribute extends ClassData{

    private boolean readOnly;
    private ArrayList<String> libImports;
    private boolean isArray;
    //is Array or Collection

    //Variable Constructor

    /**
     *
     * @param nm name of attribute
     * @param visType visibility type
     * @param typ attribute type
     * @param array is it an array
     */
}

```

```

    * @param isStatic is it static
    * @param rO is it read only (final)
    * @param imports array list of imports
    * @param comms string of comments
    */

    public Attribute(String nm, VisibilityType visType, String typ, boolean array, boolean
isStatic, boolean rO, ArrayList<String> imports, String comms) {

        super(nm, visType, typ, isStatic, imports, comms);

        readOnly = rO;

        libImports = imports;

        isArray = array;

    }

    @Override

    public String toString(){

        String vis = super.getVisibility().getValShort();

        String name = super.getName();

        String type = super.getType();

        String array = "";

        String comments = super.getComments();

        if (!comments.trim().isEmpty()){

            comments = "{ " + comments + " }";

        } else {

            comments = "";

        }

        if(isArray){ array = "[]";}

        String attributeString = vis + name + " : " + type + array + " " +

            comments;

        return attributeString;

    }

```



```

/**
 *
 * @return is attribute readOnly
 */
public boolean isReadOnly() {
    return readOnly;
}

/**
 *
 * @return list of imports
 */
public ArrayList<String> getLibImports() {
    return libImports;
}

/**
 *
 * @return is attribute an array
 */
public boolean isArray() {
    return isArray;
}

/**
 *
 * @param readOnly set to true of false if it is read only
 */
public void setIsReadOnly(boolean readOnly) {
    this.readOnly = readOnly;
}

/**
 *
 * @param libImports list of imports

```

```

    */

    public void setLibImports(ArrayList<String> libImports) {
        this.libImports = libImports;
    }

    /**
     *
     * @param isArray is it an array
     */
    public void setIsArray(boolean isArray) {
        this.isArray = isArray;
    }
}

```

#### 7.1.2.3 - CMethod

```

package umldc.data;

import java.util.ArrayList;
import java.util.concurrent.CopyOnWriteArrayList;
import javax.swing.DefaultListModel;
import javax.swing.JList;
import javax.swing.JOptionPane;

/**
 *
 * @author rayha
 */
public class CMethod extends ClassData implements IMethodSubject {

    private CopyOnWriteArrayList<Parameter> parameters = new CopyOnWriteArrayList<Parameter>();
    private JList<String> paramListView;
    private DefaultListModel paramListModel;
    private ArrayList<IMethodObserver> observers;
    private CMethod state;

    /**
     * used to create an empty method object

```

```

    */

public CMethod() {
    observers = new ArrayList<IMethodObserver>();
}

/**
 *
 * @param state set the state of the method
 */
public void setState(CMethod state) {
    this.state = state;
}

/**
 *
 * @return get state of method
 */
public CMethod getState() {
    return state;
}

/**
 *
 * @return save the state as a new memento
 */
public MethodMemento saveStateToMemento() {
    return new MethodMemento(state);
}

/**
 *
 * @param memento the memento's state to get
 */
public void getStateFromMemento(MethodMemento memento) {
    state = memento.getState();
}

```

```

/**
 *
 * @param o observer to register
 */
@Override
public void registerObserver(IMethodObserver o) {
    observers.add(o);
}

/**
 *
 * @param o observer to remove
 */
@Override
public void removeObserver(IMethodObserver o) {
    int index = observers.indexOf(o);
    if (index >= 0) {
        observers.remove(index);
    }
}

/**
 * notify an observer if a change is made
 */
@Override
public void notifyObservers() {

    for (IMethodObserver observer : observers) {
        observer.update(this);
    }
}

/**
 *
 * @return get list of observers

```

```

*/

public ArrayList<IMethodObserver> getObservers() {
    return observers;
}

/**
 *
 * @param param the parameter to edit
 * @param selectedIndex the selected parameters index
 */
public void editParam(Parameter param, int selectedIndex) {

    boolean exists = false;

    for (int count = 0; count < parameters.size(); count++) {

        if (count == selectedIndex) {

            try {
                count++;
            } catch (ArrayIndexOutOfBoundsException e) {
                //Do nothing
            }
        }

        try {
            if (parameters.get(count).getName().equals(param.getName())) {
                exists = true;
            }
        } catch (ArrayIndexOutOfBoundsException e) {
            //Do nothing
        }

    }

    if (exists) {
        JOptionPane.showMessageDialog(null,

```

```

        "Parameter name already exists",
        "Warning",
        JOptionPane.WARNING_MESSAGE);
    } else {
        Parameter edit = parameters.get(selectedIndex);
        edit.setName(param.getName());
        edit.setType(param.getType());
        edit.setIsArray(param.isArray());
        edit.setLibImports(param.getLibImports());
        notifyObservers();
    }
    System.out.println("Parameters");
    System.out.println(parameters);
}

@Override
public String toString() {

    String methString = "";

    String vis = getVisibility().getValShort();
    String comments = getComments();

    if (!comments.trim().isEmpty()) {
        comments = "{ " + comments + " }";
    } else {
        comments = "";
    }

    methString = vis + getName() + paramsToString() + " : " + getType() + " " + comments;

    return methString;//methString;
}

/**
 *

```

```

    * @return a temporary string (not used in implementation)
    */

    public String toStringTemp() {

        return "CMethod{Name: " + getName() + "Visibility: " + getVisibility() + "Parameters: " +
paramsToString() + "Comments: " + getComments() + '}';

    }

    /**
     *
     * @return string version of a parameter, used to show in the class diagram
     */
    public String paramsToString() {

        String params = "";

        for (Parameter parameter : parameters) {
            if (parameter.equals(parameters.get(parameters.size() - 1))) {
                params += parameter.toString();
            } else {
                params += parameter.toString() + ", ";
            }
        }

        params = "(" + params + ")";

        return params;
    }

    /**
     *
     * @param param parameter to add to method
     */
    public void addParam(Parameter param) {

        parameters.add(param);

        notifyObservers();
    }

```

```

}

/**
 *
 * @param param parameter to check if its valid
 * @return returns true if the parameter is valid
 */
public boolean isValidParam(Parameter param) {
    boolean added = true;

    if (!parameters.isEmpty()) {
        for (Parameter item : parameters) {
            if (item.getName().equals(param.getName())) {
                added = false;
                //exists = true;
            }
        }
    }

    if (!added) {
        JOptionPane.showMessageDialog(null,
            "Parameter name already exists",
            "Warning",
            JOptionPane.WARNING_MESSAGE);
    }

    return added;
}

/**
 *
 * @param index index of parameter to be removed
 */
public void removeParam(int index) {

```



```

        try {
            //int index = paramListView.getSelectedIndex();
            //paramListModel.remove(index);

            parameters.remove(index);

            notifyObservers();
        } catch (Exception e) {
            //Ignore
        }
    }

    /**
     *
     * @param parameters set method parameters to these parameters
     */
    public void setParameters(CopyOnWriteArrayList<Parameter> parameters) {
        this.parameters = parameters;
        notifyObservers();
    }

    /**
     *
     * @return returns parameters
     */
    public CopyOnWriteArrayList<Parameter> getParameters() {
        return parameters;
    }

    /**
     *
     * @return parameterListView
     */
    public JList<String> getParamListView() {
        return paramListView;
    }

    /**

```

```

        *

        * @return DefaultListModel paramListModel
        */

    public DefaultListModel getParamListModel() {

        return paramListModel;

    }

}

```

#### 7.1.2.4 - ClassData

```

package umldc.data;

import java.util.ArrayList;

/**
 *
 * @author rayha
 */
public abstract class ClassData {

    private String name;
    private VisibilityType visibility;
    private String comments;
    private boolean isStatic;
    private String type;
    private ArrayList<String> libImports;

    //Variable Constructor

    /**
     *
     * @param nm name of ClassData
     * @param visType visibility type of ClassData
     * @param typ type of ClassData
     * @param isStatic is the ClassData Static
     * @param imports list of imports
     * @param comms comments
     */
}

```

```

    public ClassData(String nm, VisibilityType visType, String typ, boolean isStatic,
ArrayList<String> imports, String comms){

        name = nm;

        visibility = visType;

        this.isStatic = isStatic;

        comments = comms;

        type = typ;

        libImports = imports;

    }

    //Parameter Constructor

    /**
     *
     * @param nm name of ClassData
     * @param typ type of ClassData
     */
    public ClassData(String nm, String typ){

        name = nm;

        type = typ;

    }

    /**
     *
     */
    public ClassData(){

    }

    /**
     *
     * @return string version of imports
     */
    public String importsToString(){

        String importsString = "";

        try {

```

```

        for (String imp : libImports) {
            importsString += imp + "\n";
        }
    } catch (NullPointerException e) {

    }

    return importsString;
}

/**
 *
 * @return name of ClassData
 */
public String getName() {
    return name;
}

/**
 *
 * @return visibility type
 */
public VisibilityType getVisibility() {
    return visibility;
}

/**
 *
 * @return comments
 */
public String getComments() {
    return comments;
}

/**
 *
 * @return if it is static

```

```

    */

    public boolean isStatic() {

        return isStatic;

    }

    /**
     *
     * @return type of ClassData
     */
    public String getType() {

        return type;

    }

    /**
     *
     * @return imports list
     */
    public ArrayList<String> getLibImports() {

        return libImports;

    }

    /**
     *
     * @param name name to be set
     */
    public void setName(String name) {

        this.name = name;

        System.out.println("Name Set");

    }

    /**
     *
     * @param visibility visibility type to be set
     */
    public void setVisibility(VisibilityType visibility) {

        this.visibility = visibility;

```

```

    }

    /**
     *
     * @param comments comments to be set
     */
    public void setComments(String comments) {
        this.comments = comments;
    }

    /**
     *
     * @param isStatic is it static
     */
    public void setIsStatic(boolean isStatic) {
        this.isStatic = isStatic;
    }

    /**
     *
     * @param type type to be set
     */
    public void setType(String type) {
        this.type = type;
    }

    /**
     *
     * @param libImports list of imports to be set
     */
    public void setLibImports(ArrayList<String> libImports) {
        this.libImports = libImports;
    }
}

```

#### **7.1.1.2.5 - ClassObject**

```
package umldc.data;
```

```

import umldc.data.Attribute;

import java.awt.Window;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.concurrent.CopyOnWriteArrayList;

import javax.swing.DefaultListModel;

import javax.swing.JDialog;

import javax.swing.JList;

import javax.swing.JOptionPane;

import javax.swing.JTextField;

/**
 *
 * @author rayha
 */
public class ClassObject implements IClassSubject{

    private String className;

    private CopyOnWriteArrayList<Attribute> attributes = new CopyOnWriteArrayList<Attribute>();

    private CopyOnWriteArrayList<CMethod> methods = new CopyOnWriteArrayList<CMethod>();

    private ArrayList<IClassObserver> observers;

    /**
     *
     */
    public ClassObject(){

        observers = new ArrayList<IClassObserver>();

    }

    /**
     *
     * @param cName class name
     * @param attList attribute list
     * @param methList method list
     * @param relationMap class relationships (not yet implemented)
     */

```

```

    public ClassObject(String cName, CopyOnWriteArrayList<Attribute> attList,
CopyOnWriteArrayList<CMethod> methList, HashMap<ClassObject, RelationType> relationMap){

        className = cName;

        attributes = attList;

        methods = methList;

    }

    /**
     *
     * @param o observer to be registered
     */
    @Override
    public void registerObserver(IClassObserver o) {
        observers.add(o);
    }

    /**
     *
     * @param o observer to be removed
     */
    @Override
    public void removeObserver(IClassObserver o) {
        int index = observers.indexOf(o);
        if(index >= 0){
            observers.remove(index);
        }
    }

    /**
     * notify all observers of any changes
     */
    @Override
    public void notifyObservers() {
        for (IClassObserver observer : observers){
            observer.update(this);
        }
    }

```



```

    }
}

/**
 *
 * @param varsListView jlist to remove attribute from
 */
public void removeAttribute(JList varsListView) {
    try {
        int index = varsListView.getSelectedIndex();
        attributes.remove(index);
        notifyObservers();
    } catch (Exception e) {
        //Ignore
    }
}

/**
 *
 * @param methsListView jlist to remove method from
 */
public void removeMethod(JList methsListView) {
    try {
        int index = methsListView.getSelectedIndex();
        methods.remove(index);
        notifyObservers();
    } catch (Exception e) {
        //Ignore
    }
}

/**
 *
 * @param var attribute to add
 */
public void addAttribute(Attribute var) {

```

```

        attributes.add(var);
        notifyObservers();
    }

    /**
     *
     * @param var attribute to check if it exists
     * @return
     */
    public boolean attributeExists(Attribute var) {
        boolean exists = false;

        if (!attributes.isEmpty()) {
            for (Attribute item : attributes) {
                if (item.getName().equals(var.getName())) {
                    exists = true;
                }
            }
        }

        if (exists) {
            JOptionPane.showMessageDialog(null,
                "Variable name already exists",
                "Warning",
                JOptionPane.WARNING_MESSAGE);
        }

        return exists;
    }

    /**
     *
     * @param meth method to add
     */
    public void addMethod(CMethod meth) {

```

```

        methods.add(meth);

        System.out.println(methods);

        notifyObservers();

        System.out.println("MY METHODS");

    }

    /**
     *
     * @param meth method to check if it exists
     * @return
     */
    public boolean methodExists(CMethod meth) {
        boolean exists = false;

        if (!methods.isEmpty()) {
            for (CMethod item : methods) {
                if (item.getName().equals(meth.getName())) {
                    exists = true;
                }
            }
        }

        if (exists) {
            JOptionPane.showMessageDialog(null,
                "Method name already exists",
                "Warning",
                JOptionPane.WARNING_MESSAGE);
        }

        return exists;
    }

    /**

```

```

*
* @param meth method to edit
* @param selectedIndex selected methods index in jlist
*/
public void editMethod(CMethod meth, int selectedIndex){

    String methName = meth.getName();

    boolean exists = false;

    for (int count = 0; count < methods.size(); count++) {

        if (count == selectedIndex) {

            try {

                count++;

            } catch (ArrayIndexOutOfBoundsException e) {

                //Do nothing

            }

        }

        try {

            if (methods.get(count).getName().equals(methName)) {

                exists = true;

            }

        } catch (ArrayIndexOutOfBoundsException e) {

            //Do nothing

        }

    }

    if (exists){

        JOptionPane.showMessageDialog(null,

            "Variable name already exists",

            "Warning",

            JOptionPane.WARNING_MESSAGE);
    }
}

```

```

    } else {

        CMethod edit = methods.get(selectedIndex);

        edit.setName(meth.getName());

        edit.setVisibility(meth.getVisibility());

        edit.setType(meth.getType());

        edit.setIsStatic(meth.isStatic());

        edit.setLibImports(meth.getLibImports());

        edit.setComments(meth.getComments());

        notifyObservers();

    }
}

//

/**
 *
 * @param att attribute to be edited
 * @param selectedIndex selected attributes index in jlist
 */
public void editAttribute(Attribute att, int selectedIndex){

    String varName = att.getName();

    boolean exists = false;

    for (int count = 0; count < attributes.size(); count++) {

        if (count == selectedIndex) {

            try {

                count++;

            } catch (ArrayIndexOutOfBoundsException e) {

                //Do nothing

            }

        }

    }
}

```

```

        try {
            if (attributes.get(count).getName().equals(varName)) {
                exists = true;
            }
        } catch (ArrayIndexOutOfBoundsException e) {
            //Do nothing
        }

    }

    if (exists){
        JOptionPane.showMessageDialog(null,
            "Variable name already exists",
            "Warning",
            JOptionPane.WARNING_MESSAGE);
    } else {

        Attribute edit = attributes.get(selectedIndex);
        edit.setName(varName);
        edit.setVisibility(att.getVisibility());
        edit.setType(att.getType());
        edit.setIsArray(att.isArray());
        edit.setIsStatic(att.isStatic());
        edit.setIsReadOnly(att.isReadOnly());
        edit.setLibImports(att.getLibImports());
        edit.setComments(att.getComments());
        notifyObservers();
    }

}

@Override
public String toString() {

```

```

        return "ClassObject{" + "className=" + className + ", attributes=" + attributes + ",
methods=" + methods + '}'';
    }

    /**
     *
     * @return method list
     */
    public CopyOnWriteArrayList<CMethod> getMethods() {
        return methods;
    }

    /**
     *
     * @return class name
     */
    public String getClassName() {
        return className;
    }

    /**
     *
     * @param className clas name to set
     */
    public void setClassName(String className) {
        this.className = className;
    }

    /**
     *
     * @return list of attributes
     */
    public CopyOnWriteArrayList<Attribute> getAttributes() {
        return attributes;
    }

    /**

```

```

    *
    * @param attributes set class attributes to this
    */
    public void setAttributes(CopyOnWriteArrayList<Attribute> attributes) {
        this.attributes = attributes;
    }
}

```

### 7.1.2.6 - Parameter

```

package umldc.data;

import java.util.ArrayList;

/**
 *
 * @author rayha
 */
public class Parameter extends ClassData{

    private boolean isArray;
    private ArrayList<String> libImports;

    /**
     * used to create empty parameter object
     */
    public Parameter(){

    }

    /**
     *
     * @param nm name of parameter
     * @param typ type of parameter
     * @param array is parameter an array
     * @param imports list of imports of parameter
     */
}

```



```

public Parameter(String nm, String typ, boolean array, ArrayList<String> imports){
    super(nm, typ);

    isArray = array;
    libImports = imports;
}

@Override
public String toString(){
    String name = super.getName();
    String type = super.getType();
    String array = "";

    if(isArray){ array = "[]";}

    String attributeString = name + " : " + type + array;

    return attributeString;
}

/**
 *
 * @return if the parameter is an array
 */
public boolean isArray() {
    return isArray;
}

/**
 *
 * @return imports of parameter
 */
public ArrayList<String> getLibImports() {
    return libImports;
}

```

```

/**
 *
 * @param isArray true if parameter is an array
 */
public void setIsArray(boolean isArray) {
    this.isArray = isArray;
}

/**
 *
 * @param libImports list of imports to be set
 */
public void setLibImports(ArrayList<String> libImports) {
    this.libImports = libImports;
}
}

```

#### **7.1.2.7 - RelationType**

```
package umldc.data;
```

```

/**
 *
 * @author rayha
 */
public enum RelationType {

}

```

#### **7.1.2.8 - VisibilityType**

```

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package umldc.data;

/**
 *

```

```

* @author rayha
*/
public enum VisibilityType {

    /**
     * public visibility
     */
    PUBLIC("public", "+"),

    /**
     * protected visibility
     */
    PROTECTED("protected", "#"),

    /**
     * package visibility
     */
    PACKAGE("package", "~"),

    /**
     * private visibility
     */
    PRIVATE("private", "-");

    private String value;
    private String valShort;

    VisibilityType(String val, String shrt){
        value = val;
        valShort = shrt;
    }

    @Override
    public String toString() {
        return value; //To change body of generated methods, choose Tools | Templates.
    }
}

```

```

/**
 *
 * @return short version of the visibility type e.g. +, -, #, ~
 */
public String getValShort() {
    return valShort;
}
}

```

#### **7.1.2.9 - IClassObserver**

```
package umldc.data;
```

```

/**
 *
 * @author rayha
 */
public interface IClassObserver {

    /**
     *
     * @param cOb class object to update
     */
    public void update(ClassObject cOb);
}

```

#### **7.1.2.10 - IClassSubject**

```
package umldc.data;
```

```

/**
 *
 * @author rayha
 */
public interface IClassSubject {

    /**

```

```

    *

    * @param o observer to register
    */

    public void registerObserver(IClassObserver o);

    /**
     *
     * @param o observer to remove
     */

    public void removeObserver(IClassObserver o);

    /**
     * notify all observers of any changes made to class object
     */

    public void notifyObservers();

}

```

#### **7.1.2.11 - IMethodObserver**

```
package umldc.data;
```

```

/**
 *
 * @author rayha
 */

public interface IMethodObserver {

    /**
     *
     * @param meth method to update
     */

    public void update(CMethod meth);

}

```

#### **7.1.2.12 - IMethodSubject**

```
package umldc.data;
```

```

/**
 *

```

```

    * @author rayha
    */

public interface IMethodSubject {

    /**
     *
     * @param o observer to register
     */
    public void registerObserver(IMethodObserver o);

    /**
     *
     * @param o observer to remove
     */
    public void removeObserver(IMethodObserver o);

    /**
     * notify all observers of any changes made to method
     */
    public void notifyObservers();
}

```

### 7.1.2.13 - MethodMemento

```
package umldc.data;
```

```

/**
 *
 * @author rayha
 */

public class MethodMemento {

    private CMethod state;

    /**
     *
     * @param state current state of method
     */
    public MethodMemento(CMethod state){

```

```
        this.state = state;
    }

    /**
     *
     * @return state of method
     */
    public CMethod getState(){
        return state;
    }
}
```