

UML to JAVA Converter for Vitians

CSE3001_SOFTWARE ENGINEERING

by

Palash Sarma

19BCE2092

Gokul Gopakumar

19BCI0187



VIT[®]

Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

DECLARATION

June, 2020

We hereby declare that the thesis entitled **“UML to Java Converter for Vitians”** submitted by me, for the award of the degree of *Bachelor of Technology in Computer Science Engineering* to VIT is a record of bonafide work carried out by me under the supervision of **Professor Sureshkumar WI.**

We further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place : Vellore

Date: June, 2021

**Palash
Gokul
Signature of the Candidate**

ACKNOWLEDGEMENTS

In performing our assignment, we had to take the help and guideline of some respected persons, who deserve our greatest gratitude. The completion of this assignment gives us much Pleasure. We would like to show our gratitude to **Mr. Sureshkumar WI, Course Instructor, VIT university** for giving us a good guideline for assignment throughout numerous consultations. We would also like to expand our deepest gratitude to all those who have directly and indirectly guided us in writing this assignment.

Many people, especially our classmates and team members, have made valuable comment suggestions on this proposal which gave us inspiration to improve our assignment. We thank all the people for their help directly and indirectly to complete our assignment.

Palash Sarma and Gokul Gopakumar

Executive Summary

The project is a Jar application made for university coursework under the course Software Engineering. The project is a desktop based application where according to user inputs for class diagrams a Java code will be generated. There are 4 types of classes included in the project namely; Normal, Inheritance, Abstract and Implement . This will help in covering almost all types of relations in JAVA OOPS concepts. The application targets software newbies who are new to the OOPS concepts and also industry experts who want to avoid writing huge lines of code or creating code templates in JAVA sharing the same relation logic. This software even provides custom methods like get and set which will help the end user to access the class fields. The software has the ability to generate even object data types or standard as per user. There is an option of printing any statement in a method which might state the method requirement for that class. There is an option to add comments for each class selected in order to convey extra information for that class.

CONTENTS	Page
Acknowledgement	
i Executive Summary	3
ii Table of Contents	4
iii List of Figures	6
iv List of Tables	8
v Abbreviations	9
1 INTRODUCTION	7
1.1 Objective	7
1.2 Motivation	8
1.3 Background	8
2 PROJECT DESCRIPTION AND GOALS	9
3 TECHNICAL SPECIFICATION	10
4 DESIGN APPROACH AND DETAILS (as applicable)	16
4.1 Design Approach / Materials & Methods	17
4.2 Codes and Standards	17
4.3 Constraints, Alternatives and Tradeoffs	43
5 SCHEDULE AND TASKS	44.
6 PROJECT DEMONSTRATION	47.
6.1 TESTING	62
7 COST ANALYSIS / RESULT & DISCUSSION	83
8 CONCLUSION	83
9 REFERENCES	84

1 INTRODUCTION

1.1 Objective

The main goals of the our software are:

1. This software allows the user to choose data types of methods/fields .
2. The software can take a large number of classes for input.
3. The software can track the class numbers by serializing it .
4. Provide a way for users to provide comments for the classes chosen.
5. Users can easily append class details hassle free.
6. A file gets created at the end with the required java code.
7. Benefits the user by providing an option to write a comment for the classes selected in order to convey more information regarding that class.
8. Prevent the user from setting wrong fields/methods for the wrong class by keeping track of every form.
9. The software provides syntax-error free Java Coder

1.2 Motivation

Availability of a software which enables to create larger class templates in Java saving time and effort. A platform where even after having basic OOPS knowledge a user can create several code templates in order to populate later with required algorithm. A software which can save time and effort in writing templates and start early with his/her project implementation.

1.3 Background

There are few tools like Visual paradigm where one can create UML and the platform can convert to JAVA code. These type of tools where user with knowledge of basic class diagram notations enable his/her to produce powerful code templates. The user can either create class diagrams by dragging and dropping entities from toolbox or upload a dot file. The software parses the dot file and generate respective code according the class Diagrams created/upload. Some of them lag the feature of providing custom options which might help the user while populating the templates with code.

2 PROJECT DESCRIPTION AND GOALS

Our Project aims to create a software where data from the user in form of Class diagram inputs to generate respective code. The software will apply our own created attribute formatting logic which will parse the inputs according to how the user requires it to be. Placed in the output code. There are 4 types of classes; namely A normal Java class, Inheritance class, Abstract class or a Java interface. There is also an option to choose implement/child class for any super class(can be an Abstract/Inheritance/Interface). Further all the classes are provided with option of adding fields(Class Attributes) and methods(Class operations). The number of class/methods/fields can be set as per user .

The software also keeps track of entity serial numbers to ease the complexity while entering data for class/fields/methods.

The project has a huge scope in the fields of education as well as IT industries. Utilized properly, this project can eliminate many existing problems such as time consumption in converting UML into code manually, as well as making better engineers for the field with better concepts of subjects like OOPS.

3 TECHNICAL SPECIFICATION

System Features

3.1.1 Creation of Class

3.1.1.1 Introduction

This feature helps in creating new class templates along with all required class attributes.

3.1.1.2 Functional Requirements

Purpose: Receiving input from the user and creating desired class template.

Input: Name of the class, class type, class access specifier, class fields, constructor type, number of methods, one main class.

Processing: Class declaration, template selection, constructor initialization.

Output: Class template with details like name, access specifiers, class field details etc.

3.1.1.3 Stimulus Response

User Actions	System Actions
(1)Entering class name	
(2)Choosing class type(super/sub)	
(3)Choosing appropriate access specifiers.	
	(4)Class header defined
(5)Number of class fields	
(6)Type of class fields	
(7)Name of class fields	
(8)Choosing constructor type(parameterised/plain)	
	(9)Constructor initialized
(10)Entering number of methods for the particular class	
	(11) goto method implementation code block

3.1.2 Creation of method

3.1.2.1 Introduction

This feature helps in creating new class methods along with all required class attributes.

3.1.2.2 Functional Requirements

Purpose: Creation of method along with its attributes according to received inputs.

Input: Name of the method, return type, method type, method access specifier, method input parameter, number of variables.

Processing: Method declaration

Output: Method template with details like name, return type, access specifier type, method type.

3.1.2.3 Stimulus Response

User Actions	System Actions
(1) Name of method entered.	
(2) Return type decided	
(3) Method type declared (parameterized/not)	
(4) Choosing appropriate access specifier	
	(5) Method header declaration initiated
	(6) if parameters exist then take input arguments
(7) number of method parameters	
(8) parameter name	
(9) parameter return type	
	(10) Method header declaration completed
	(11) Ask for custom get and set methods
(12) select custom methods if needed	
	(13) implement custom methods if opted
(14) Entering number of variables for the particular method	
	(15) block allocation for variables
(16) choose variable type	
(17) enter variable name	

3.1.3 Creation of main class

3.1.3.1 Introduction

This feature helps the user to create the main class along with object creation of other classes and main method declaration.

3.1.3.2 Functional Requirements

Purpose: defining the main method and using methods from other classes

Input: choosing which classes and the respective methods to call

Processing: Constructor calling of other classes

Output: objects of called classes created, chosen methods called.

3.1.3.3 Stimulus Response

User Actions	System Actions
(1)choose classes for which objects are to be created	
	(2)main method header declared
	(3) object creation method called
(4)choose methods from classes to be used	
	(5) methods called
	(6) return statement added

3.1.4 Custom methods

3.1.4.1 Introduction

A user may choose to use the custom get,set or print method to ease the work of writing extra code.

3.1.4.2 Functional Requirements

Purpose: using some standard java methods and print statement.

Input: choose any of the custom methods.

Processing: None

Output: Adds the method to the chosen class

3.1.4.3 Stimulus Response

User Actions	System Actions
(1) Choose the desired custom method for the class selected.	
	(2) Add the chosen method to the class
(3) choose whether to add print statement to the method	
	(4) Add the System.out.println statement

3.1.5 Attributes formatting

3.1.5.1 Introduction

This software shall format the input given by the user to preprocess before code generation.

3.1.5.2 Functional Requirements

Purpose: To preprocess input before applying code logic.

Input: Confirm the Dialogue box

Processing: The uml is processed through the predefined formatting logic

Output: The code template declared is sent to the code logic block for completing its definition.

3.1.5.3 Stimulus Response

User Actions	System Actions
(1) Confirm the dialogue box	
	(2) All the classes are identified in the UML diagram and processed differently
	(3) Each method in the class is passed through method generation code block
	(4) The preprocessed code is sent to code generator

3.1.6 Code Generation

3.1.6.1 Introduction

This software shall help the user to convert the uml generated to java code.

3.1.6.2 Functional Requirements

Purpose: To generate the java code from uml.

Input: Confirm the Dialogue box

Processing: The formatted code template is run through code logic

Output: The desired java code template.

3.1.6.3 Stimulus Response

User Actions	System Actions
(1) Confirm the dialogue box	
	(2) all the methods are compiled to define the class
	(3)the classes are connected among themselves with logic provided by the user.
(4) Final code is generated	

3.2.1 Product features

The following tables list the performance requirements of the UML2JAVA software.

Table of Performance Requirements

Performance Requirement	Description
Classes Storage Capacity	There is a limit to efficient running of the software based on number of related class
Software Runtime Errors	The UML2JAVA software will handle the runtime errors consistently and as gracefully as possible.

3.2.2 User characteristics

The following table identifies and describes the different users of the UML2JAVA. The information gathered about the different users of the system helped define what the software needs to do. Also, these users are referenced in the requirements and diagrams.

Table of User Characteristics

User	Description
Software Analyst	Software analyst analyses the the logic and correctness of code derived from UML.
Software Programmer	Software Programmer takes help of the project to produce relative changes in code by programming it.
Backend Developer	Backend Developer uses the software to write code for the given UML by his senior colleague.
Student	Anyone who uses the project for clearing JAVA OOPS concepts is a student.

3.2.3 User Requirements and Product Specific System Requirements

3.2.4 System Requirements

The system running our project should be able to connect to high-speed internet, process multiple queries at once, secure access to confidential data (user details). Use AWS servers and expect 24 x 7 availability on any kind of Browser or any kind of device. Our system does not offer a movie trailer on the icon tap, users must google it separately.

4 DESIGN APPROACH AND DETAILS

4.1 Design Approach & Methods

4.1.1 Architectural Design

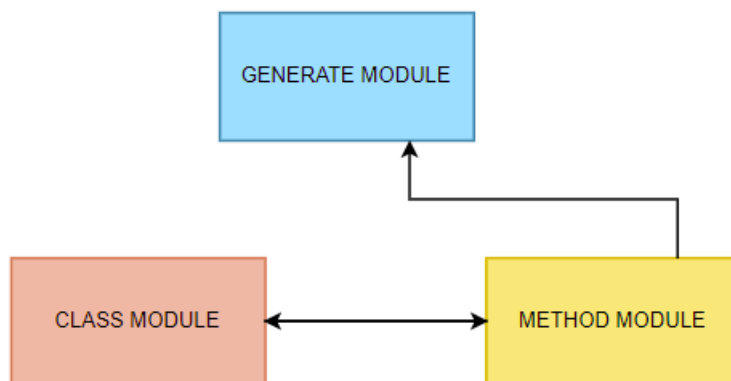


Fig: Data Flow Type Architectural Design

Description:

- We have chosen Data Flow Architecture for this project
- The main reason behind this choice is because it involves a lot of manipulative transformation(concurrent attribute formatting process) of the input data while

- generating output data(JAVA code).
- Further the execution sequence is chosen that of Pipe and Filter.
- This execution sequence will help us to give direction to the input data through pipes. Like in this project we have to correctly place the methods in the chosen respective class only.
- Similarly if user opts for custom get()& set() method for a particular class field only that class must be populated with get() & set() method.

4.1.2 Control Style:

- We have chosen Call-return model under Centralized control.
- The main reason behind this choice is because our system is mostly sequential (class->method)
- It is a small system with less number of modules interacting hence most simple approach to analyze control flow in this case can be Call-return.
- Whenever user does not confirm the dialogue box after entering all details, the system will send back the user to the home page as here, attribute formatting takes place concurrently hence it is not possible to track a single change or else the relationship between different classes will be affected generating undesired code.

4.2 Data Flow Diagrams

Data Flows

The following figures represent the data flow diagrams for various functionalities of the UML2JAVA software

Fig Data Flow Diagram of class creation

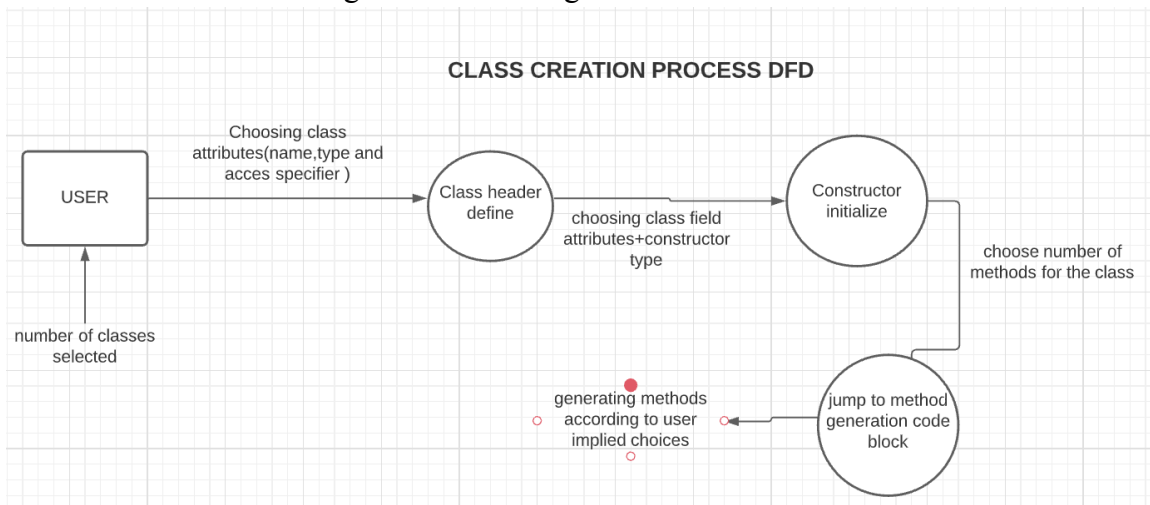


Fig: Data Flow Diagram of method creation

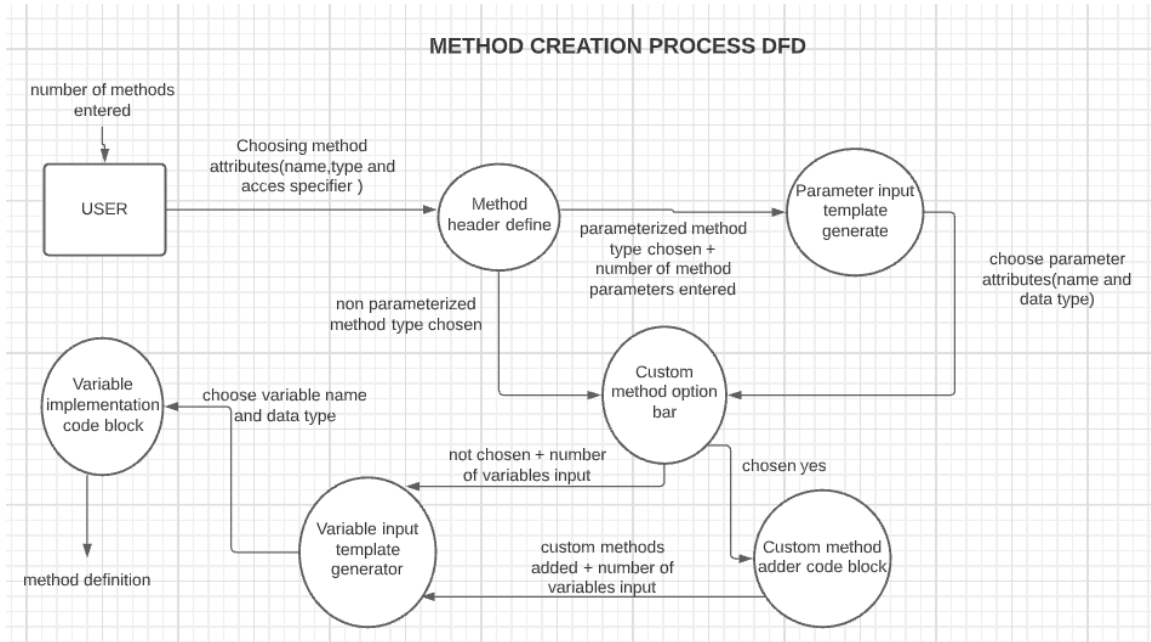


Fig: Data Flow Diagram of defining main class

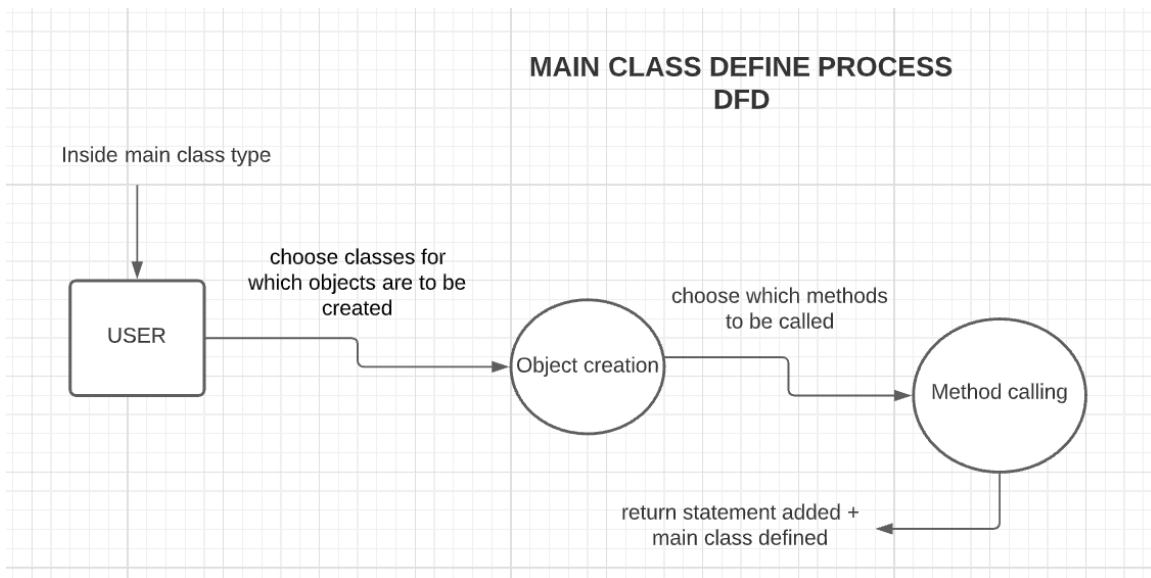


Fig : Data Flow Diagram of custom method selection

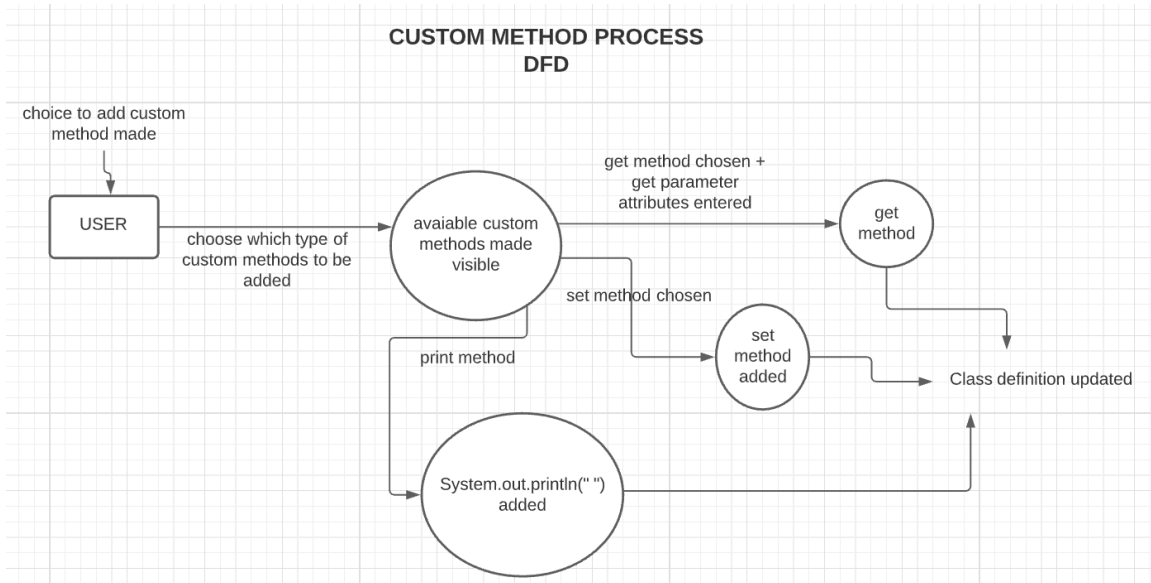


Fig: Data Flow Diagram of code generation

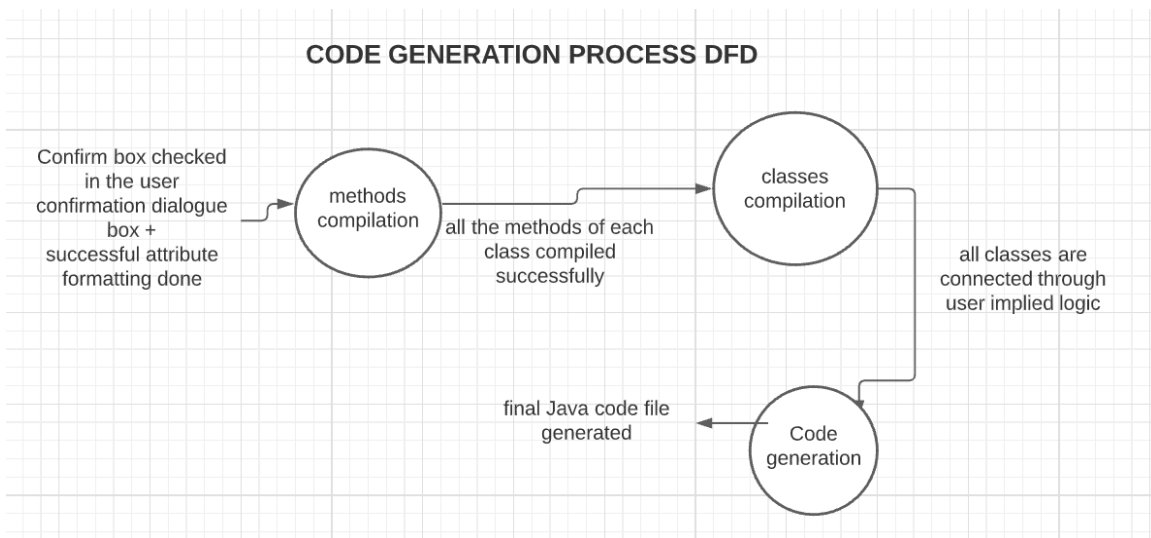
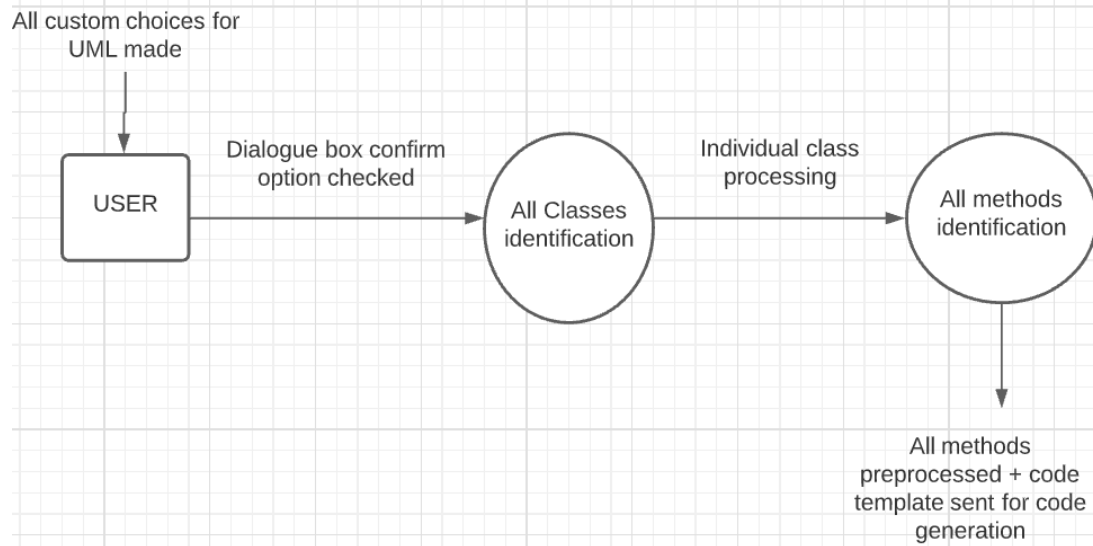
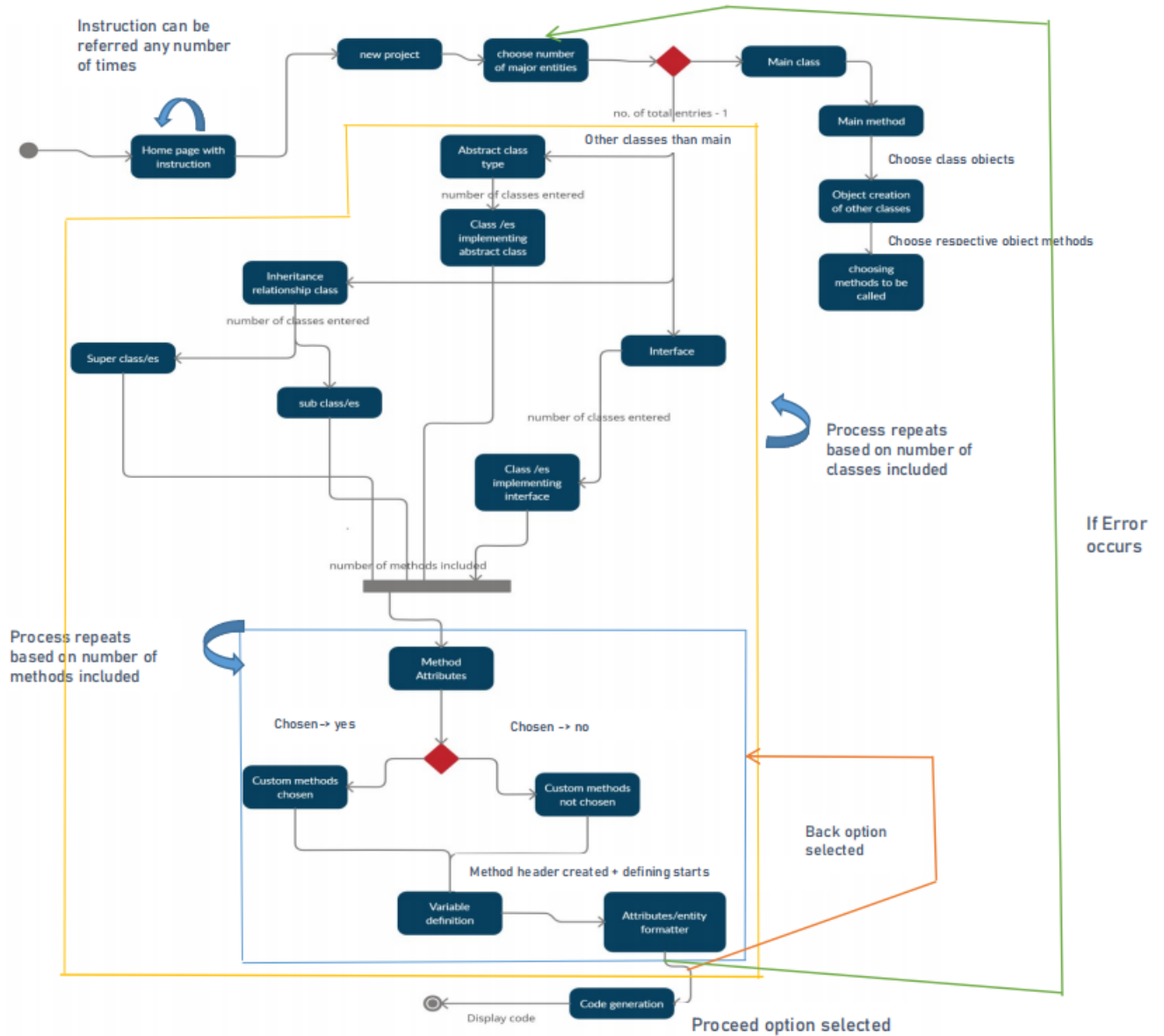


Fig: Data Flow Diagram of attribute formatting

ATTRIBUTE FORMATTING PROCESS DFD



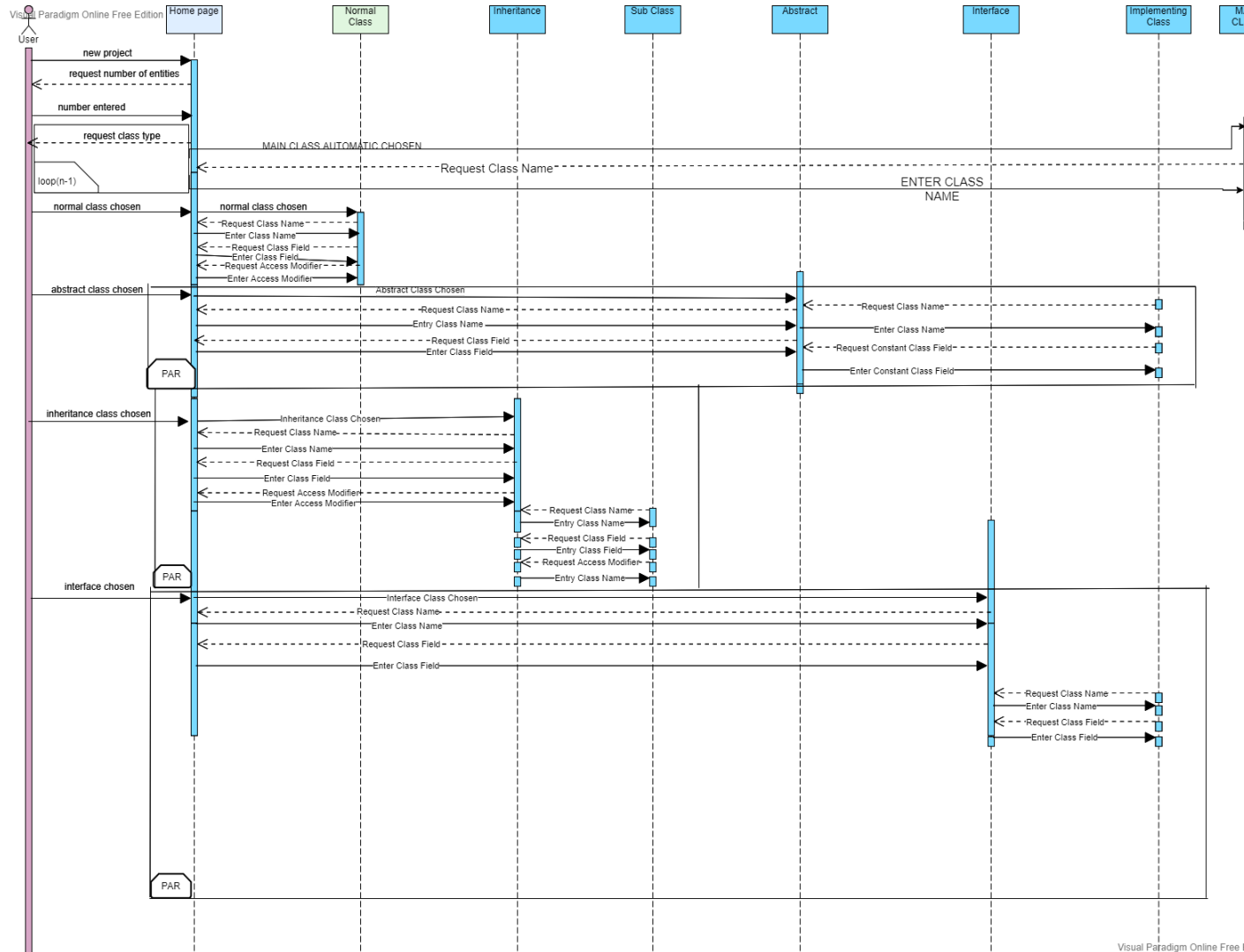
State Transition Diagram:



Module wise Detailed Designs

5.1.1 Class module

Design of sequence and use case diagram



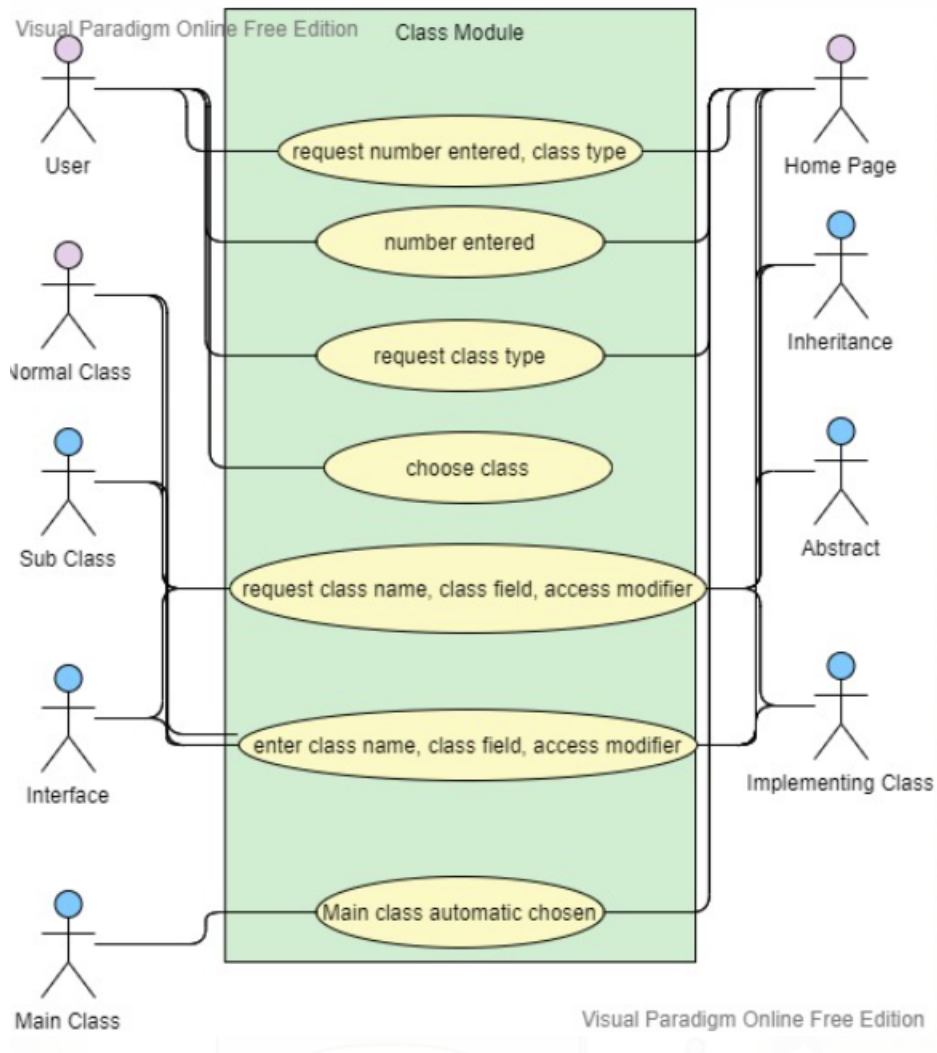


Fig: Class module sequence and use case diagram

Class module activity diagram

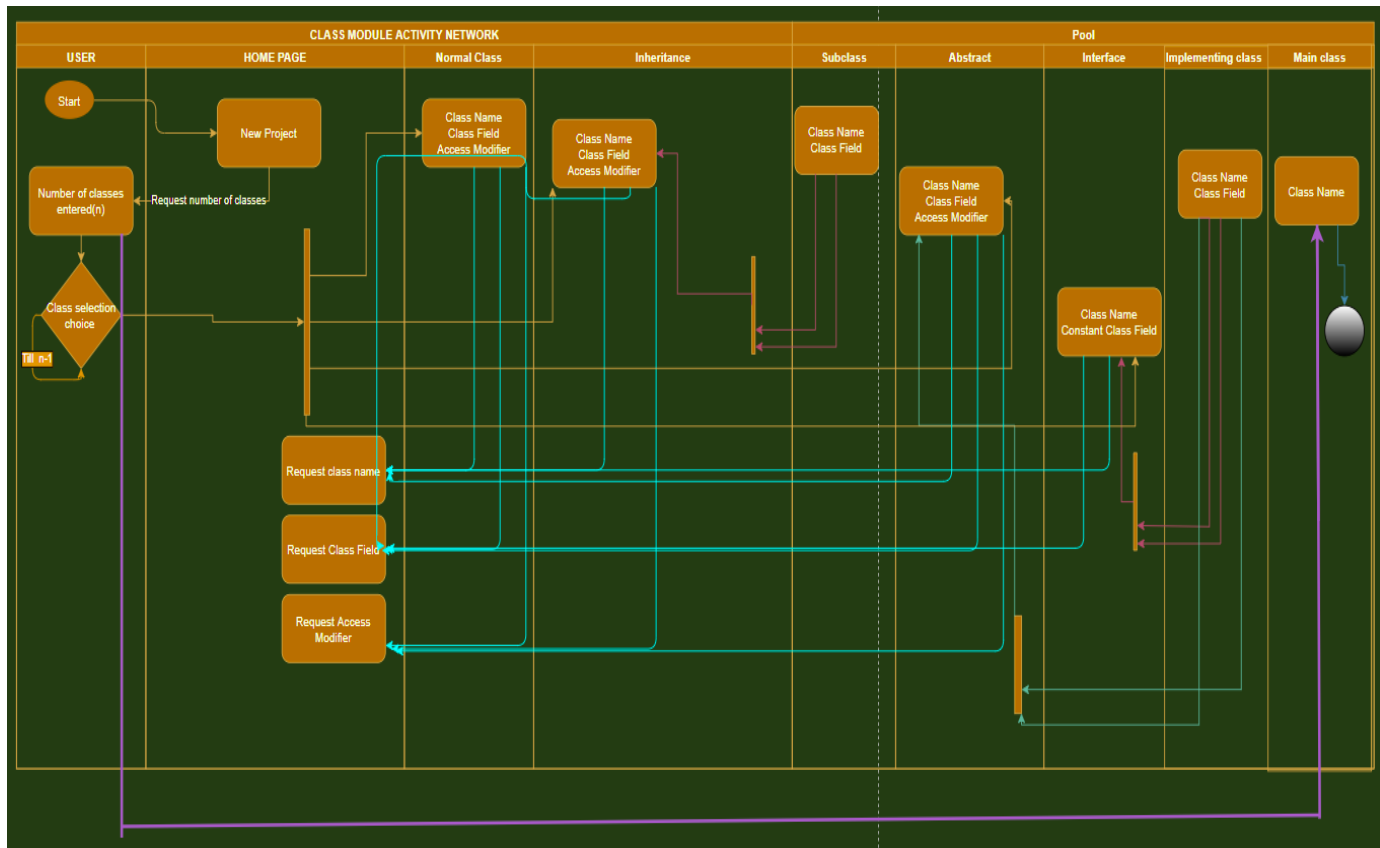
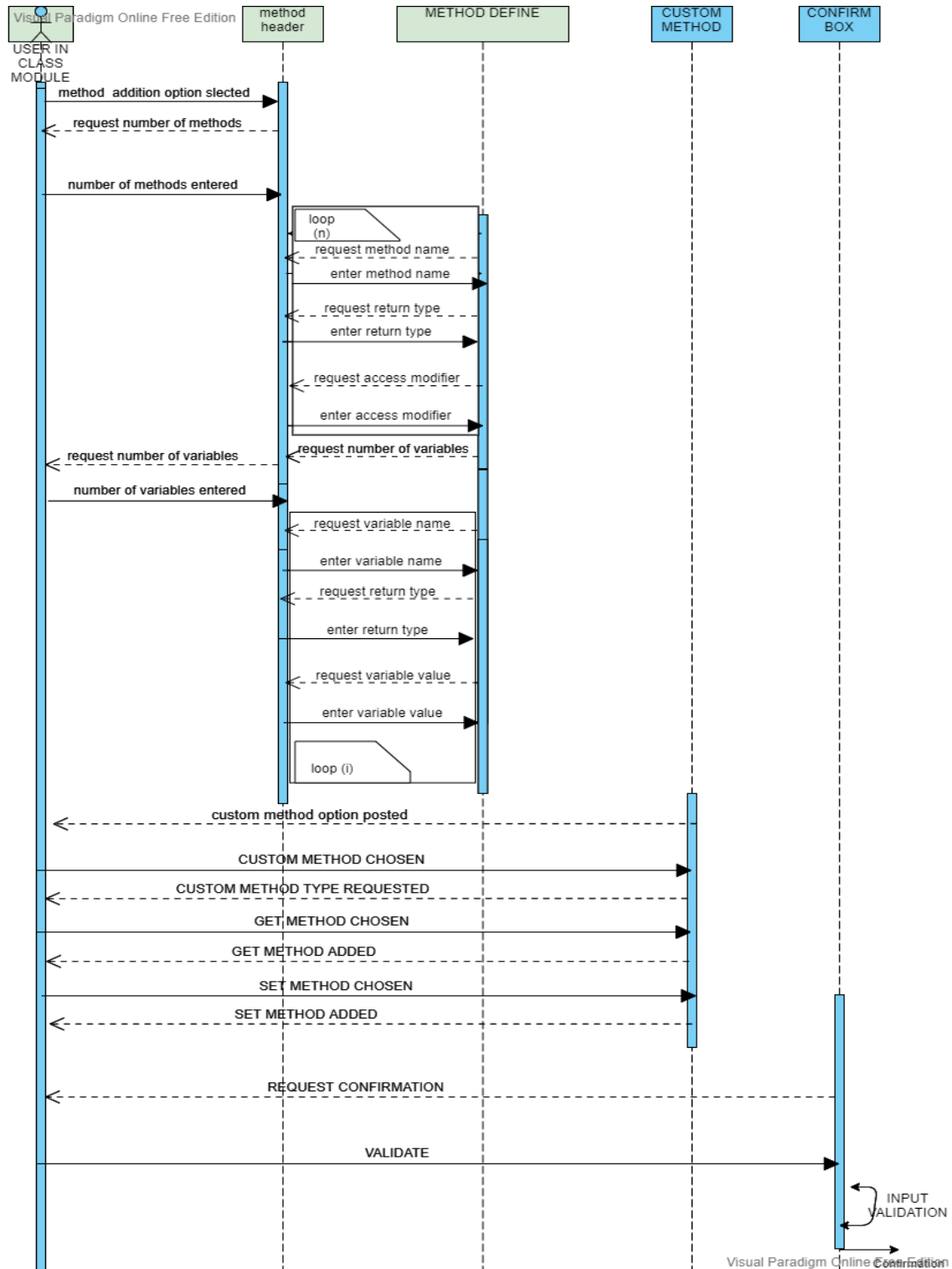


Fig: Class module activity diagram

Method module

Design of sequence and use case diagram



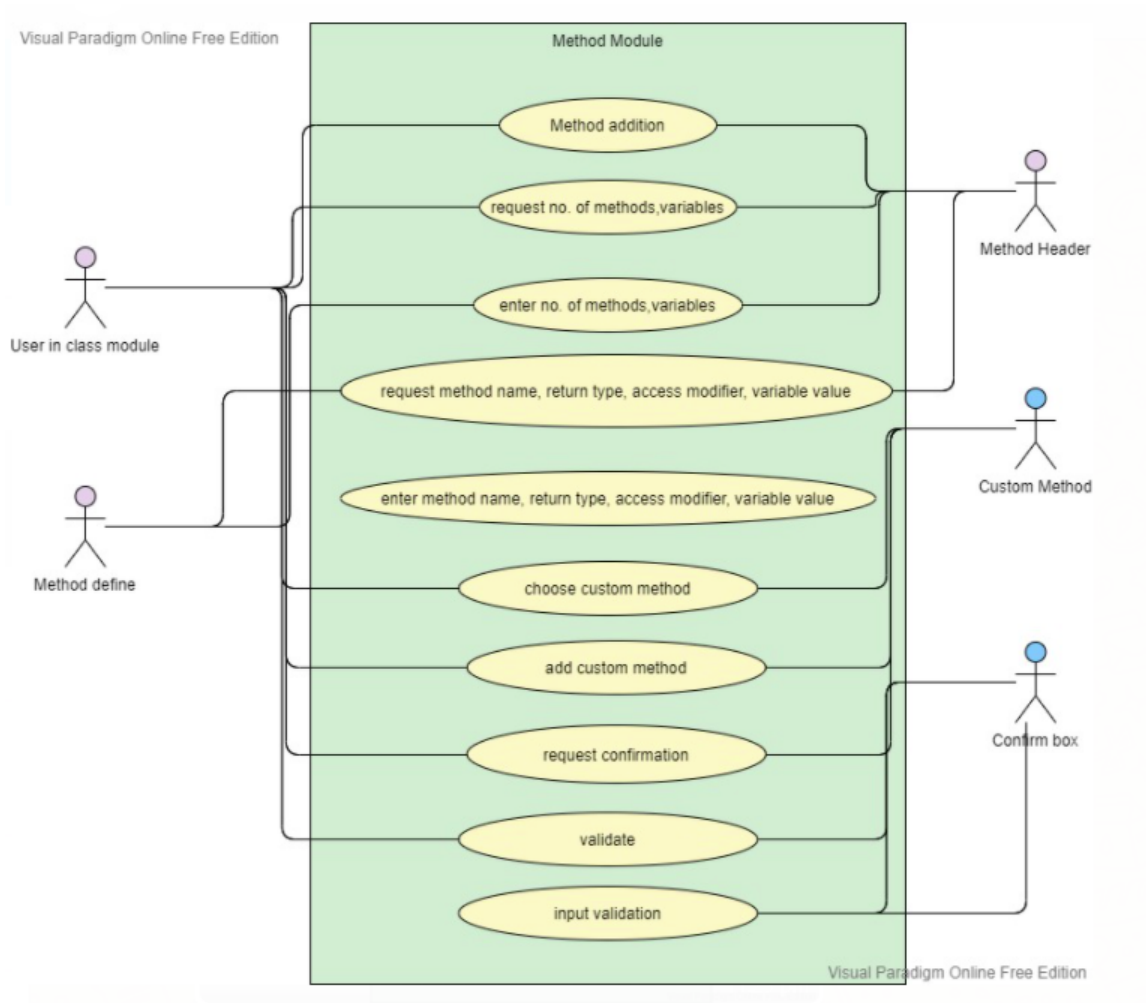


Fig: Method module sequence and use case diagram

Design

Method module activity diagram

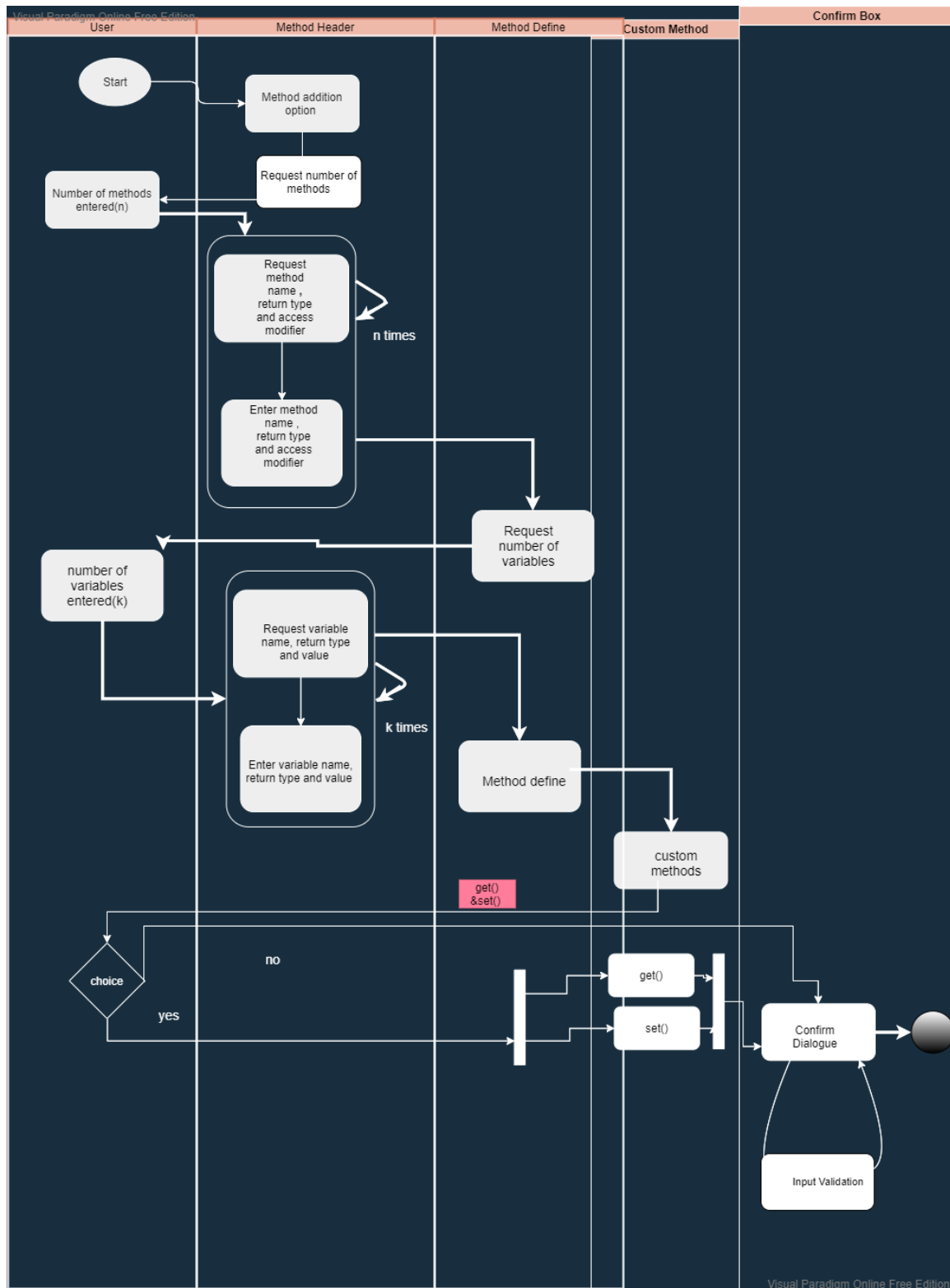
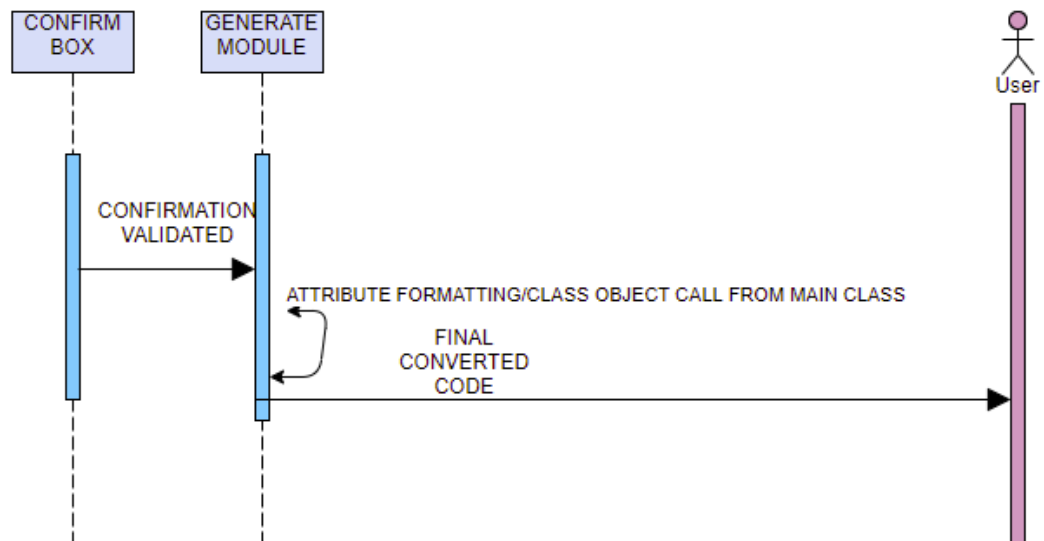


Fig: Method module activity diagram

Generate module

Design of sequence and use case diagram



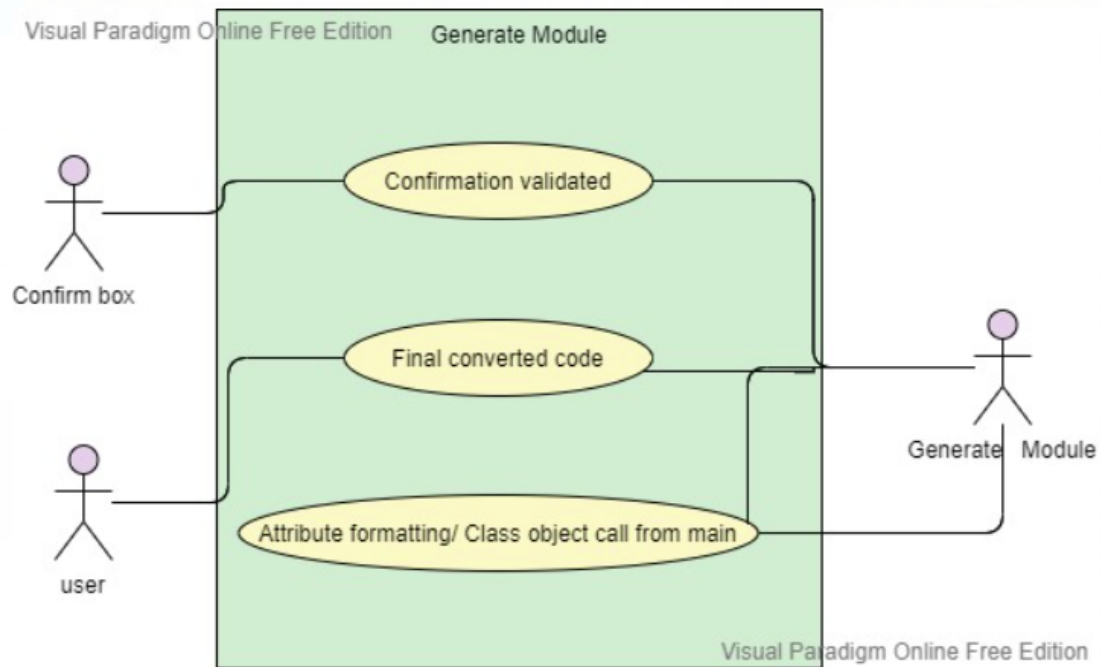


Fig: Generate module sequence and use case diagram

Generate module activity diagram

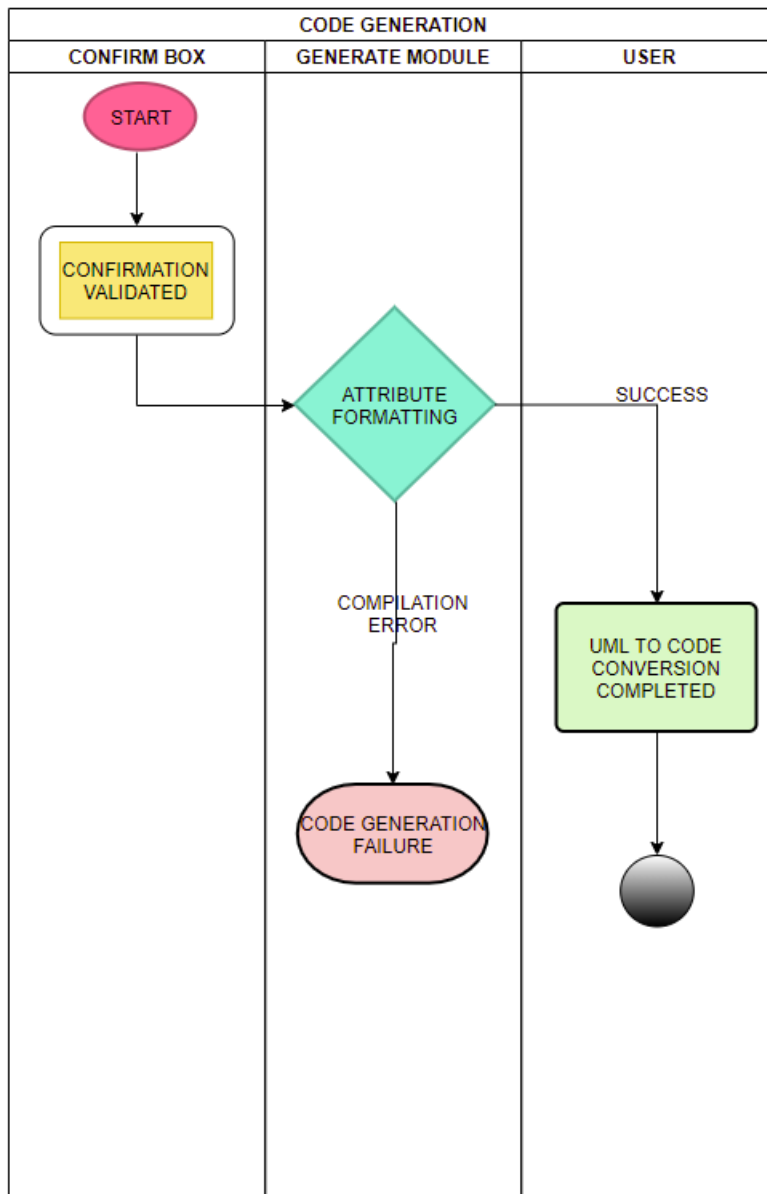
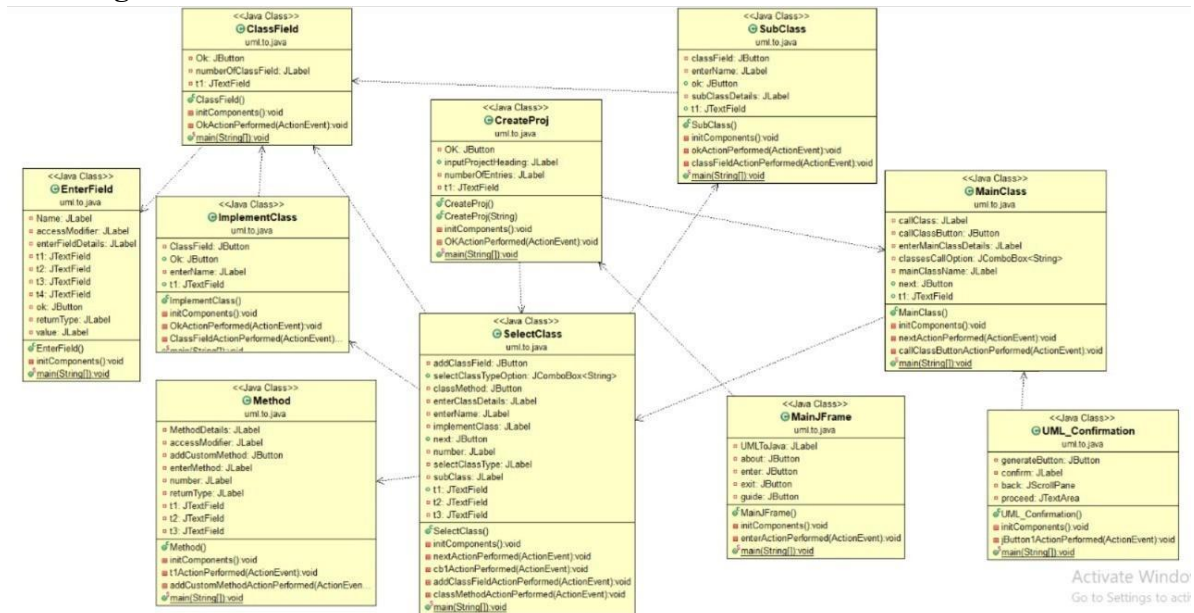


Fig: Generate module activity diagram

Class Diagram



4.1 Codes and Standards

The software has been developed using Java

4.1.1 Packages used :

- ☐ Swing
- ☐ WindowsBuilder

4.1.2 Software versions used :

- ❖ JDK 1.8
- ❖ JAVA EE 8

4.1.3 Environment :

- ❖ Eclipse IDE

4.1.4 Standards followed:

- Global variables have a limited use
- Standard headers for different modules have been mentioned
- All naming conventions for local , Global variables have been followed
- Functions , dependencies and Libraries have been clearly stated and named.

- Proper Indentation has been followed throughout the codes.
- Exception and error handling measures have been taken
- No identifier has a multiple usage
- GOTO statements are not used ,
- Codes are well documented
- High Cohesion , low coupling used
- Modularity maintained by reusing functions to create multiple cards and info pages.

4.2 Code Snippets :

```

6 package uml.to.java;
7 import javax.swing.JOptionPane;
11
12 /**
13  *
14  * @author User
15  */
16 public class MainJFrame extends javax.swing.JFrame {
17
18     /**
19      * Creates new form MainJFrame
20      */
21     public MainJFrame() {
22         initComponents();
23     }
24
25     /**
26      * This method is called from within the constructor to initialize the form.
27      * WARNING: Do NOT modify this code. The content of this method is always
28      * regenerated by the Form Editor.
29      */
30     @SuppressWarnings("unchecked")
31     // <editor-fold defaultstate="collapsed" desc="Generated Code"> //GEN-BEGIN: initComponents
32     private void initComponents() {
33
34         UMLToJava = new javax.swing.JLabel();
35         guide = new javax.swing.JButton();
36         guide.setFont(new Font("Century Schoolbook", Font.PLAIN, 17));
37         about = new javax.swing.JButton();
38         about.setFont(new Font("Century Schoolbook", Font.PLAIN, 17));
39         enter = new javax.swing.JButton();
40         enter.setFont(new Font("Century Schoolbook", Font.PLAIN, 17));
41         exit = new javax.swing.JButton();
42         exit.setFont(new Font("Century Schoolbook", Font.PLAIN, 17));
43         exit.addActionListener(new ActionListener() {
44             public void actionPerformed(ActionEvent arg0) {
45                 dispose();
46             }
47         });
48     }

```

ClassModule, Home Page

```

/**
 * 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"> //GEN-BEGIN: initComponents
private void initComponents() {

    enterName = new javax.swing.JLabel();
    enterName.setFont(new Font("Century Schoolbook", Font.PLAIN, 16));
    selectClassType = new javax.swing.JLabel();
    selectClassType.setFont(new Font("Century Schoolbook", Font.PLAIN, 16));
    cb1 = new javax.swing.JComboBox<>();
    cb1.setFont(new Font("Century Schoolbook", Font.PLAIN, 16));
    next = new javax.swing.JButton();
    next.setFont(new Font("Century Schoolbook", Font.PLAIN, 16));
    t1 = new javax.swing.JTextField();
    t1.setVisible(false);
    l2 = new javax.swing.JLabel();
    l1 = new javax.swing.JLabel();
    l1.setFont(new Font("Century Schoolbook", Font.PLAIN, 13));
    addClassField = new javax.swing.JButton();
    addClassField.setFont(new Font("Century Schoolbook", Font.PLAIN, 16));
    classMethod = new javax.swing.JButton();
    classMethod.setFont(new Font("Century Schoolbook", Font.PLAIN, 16));

    setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

    enterName.setText("Enter Class name (only if normal type chosen)");

    selectClassType.setText("Select Class Type :");

    cb1.setModel(new javax.swing.DefaultComboBoxModel<> (new String[] { "Interface", "Normal", "Abstract", "Inheritance" }));
    cb1.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {

```

Selecting Class


```

41@ @SuppressWarnings("unchecked")
42 // <editor-fold defaultstate="collapsed" desc="Generated Code">///GEN-BEGIN: initComponents
43 private void initComponents() {
44
45     jLabel1 = new javax.swing.JLabel();
46     numberOfClassField = new javax.swing.JLabel();
47     numberOfClassField.setFont(new Font("Century Schoolbook", Font.PLAIN, 16));
48     t1 = new javax.swing.JTextField();
49     Ok = new javax.swing.JButton();
50     Ok.setFont(new Font("Century Schoolbook", Font.PLAIN, 16));
51     l1 = new javax.swing.JLabel();
52     l1.setText("Blank");
53
54     setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
55
56     jLabel1.setFont(new Font("Century Schoolbook", Font.BOLD, 18)); // NOI18N
57     jLabel1.setText("Number of Class Fields");
58
59     numberOfClassField.setText("Number:");
60
61     Ok.setText("Confirm");
62@ Ok.addActionListener(new java.awt.event.ActionListener() {
63@     public void actionPerformed(java.awt.event.ActionEvent evt) {
64         OkActionPerformed(evt);
65     }
66 });
67
68     l1.setFont(new Font("Century Schoolbook", Font.BOLD, 14)); // NOI18N
69
70     JButton btnNewButton = new JButton("Return");
71     btnNewButton.setFont(new Font("Century Schoolbook", Font.PLAIN, 16));
72@ btnNewButton.addActionListener(new ActionListener() {
73@     public void actionPerformed(ActionEvent arg0) {
74         dispose();
75     }
76 });
77

```

Entering number of class fields

```

45@ @SuppressWarnings("unchecked")
46 // <editor-fold defaultstate="collapsed" desc="Generated Code">///GEN-BEGIN: initComponents
47 private void initComponents() {
48
49     Name = new javax.swing.JLabel();
50     Name.setFont(new Font("Century Schoolbook", Font.PLAIN, 16));
51     returnType = new javax.swing.JLabel();
52     returnType.setFont(new Font("Century Schoolbook", Font.PLAIN, 16));
53     accessModifier = new javax.swing.JLabel();
54     accessModifier.setFont(new Font("Century Schoolbook", Font.PLAIN, 16));
55     ok = new javax.swing.JButton();
56     ok.setFont(new Font("Century Schoolbook", Font.PLAIN, 15));
57     f1 = new javax.swing.JTextField();
58     f2 = new javax.swing.JTextField();
59     f3 = new javax.swing.JTextField();
60     enterFiledDetails = new javax.swing.JLabel();
61     jButton1 = new javax.swing.JButton();
62     jButton1.setFont(new Font("Century Schoolbook", Font.PLAIN, 16));
63
64     setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
65
66     Name.setText("Name");
67
68     returnType.setText("Return Type:");
69
70     accessModifier.setText("Acces Modifier: ");
71
72     ok.setText("OK & EXIT");
73     ok.setVisible(false);
74@ ok.addActionListener(new java.awt.event.ActionListener() {
75@     public void actionPerformed(java.awt.event.ActionEvent evt) {
76
77         okActionPerformed(evt);
78     }
79 });
80
81@ f2.addActionListener(new java.awt.event.ActionListener() {
82@     public void actionPerformed(java.awt.event.ActionEvent evt) {
83         iTextField2ActionPerformed(evt);

```

Entering Field details

```

32
33 /**
34  * This method is called from within the constructor to initialize the form.
35  * WARNING: Do NOT modify this code. The content of this method is always
36  * regenerated by the Form Editor.
37  */
38 @SuppressWarnings("unchecked")
39 // <editor-fold defaultstate="collapsed" desc="Generated Code"> // GEN-BEGIN: initComponents
40 private void initComponents() {
41
42     jLabel1 = new javax.swing.JLabel();
43     jLabel2 = new javax.swing.JLabel();
44     t1 = new javax.swing.JTextField();
45     jButton1 = new javax.swing.JButton();
46
47     setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
48
49     jLabel1.setFont(new java.awt.Font("Tahoma", 1, 18)); // NOI18N
50     jLabel1.setText("Number of Methods");
51
52     jLabel2.setText("Number:");
53
54     jButton1.setText("Confirm");
55     jButton1.addActionListener(new java.awt.event.ActionListener() {
56         public void actionPerformed(java.awt.event.ActionEvent evt) {
57             try {
58                 int l = Integer.parseInt(t1.getText());
59                 String[] arr = new String[l];
60                 int[] type = new int[l];
61                 dispose();
62                 for (int i = 0; i < l; i++) {
63                     Method frm = new Method(l1.getText());
64                     frm.setVisible(true);
65                 }
66             }

```

Entering number of methods

```

74     });
93
94     accessModifier.setText("Access Modifier");
95
96     JButton btnDone = new JButton("Done");
97     btnDone.setFont(new Font("Century Schoolbook", Font.PLAIN, 16));
98     btnDone.setVisible(false);
99
100     btnDone.addActionListener(new ActionListener() {
101         public void actionPerformed(ActionEvent e) {
102             String d = t3.getText() + " " + t2.getText() + " " + t1.getText() + "() {";
103             String l = printt + " ";
104             System.out.println(d);
105             System.out.println(l);
106             dispose();
107         }
108     });
109
110     JLabel lblNewLabel_1 = new JLabel("[Blank]");
111
112     JButton btnAttributesConfirm = new JButton("Attributes confirm");
113     btnAttributesConfirm.setFont(new Font("Century Schoolbook", Font.PLAIN, 16));
114     btnAttributesConfirm.addActionListener(new ActionListener() {
115         public void actionPerformed(ActionEvent arg0) {
116             addCustomMethod.setVisible(true);
117             t4.setVisible(true);
118             lblNewLabel_1.setVisible(true);
119             btnDone.setVisible(true);
120             lblNewLabel_1.setText("Method Header defined!");
121             btnAttributesConfirm.setVisible(false);
122         }
123     });
124
125
126

```

Entering methods details(declaration)

```

31
32 public ParentInterface(String n ){
33     initComponents();
34     l1.setText(n+"");
35 }
36
37 /*public childInherit(String n,String i ){
38     initComponents();
39     // l2.setText(n+"");
40     l1.setText(i+".1");
41 }*/
42
43 public void getName () {
44
45
46 }
47
48 /**
49  * This method is called from within the constructor to initialize the form.
50  * WARNING: Do NOT modify this code. The content of this method is always
51  * regenerated by the Form Editor.
52  */
53 @SuppressWarnings("unchecked")
54 // <editor-fold defaultstate="collapsed" desc="Generated Code"> //GEN-BEGIN: initComponents
55 private void initComponents() {
56
57     enterName = new javax.swing.JLabel();
58     enterName.setFont(new Font("Century Schoolbook", Font.PLAIN, 16));
59     next = new javax.swing.JButton();
60     next.setFont(new Font("Century Schoolbook", Font.PLAIN, 15));
61     next.setVisible(false);
62     t1 = new javax.swing.JTextField();
63
64     l1 = new javax.swing.JLabel();
65     l1.setFont(new Font("Century Schoolbook", Font.PLAIN, 16));
66     addClassField = new javax.swing.JButton();
67     addClassField.setFont(new Font("Century Schoolbook", Font.PLAIN, 16));
68     classMethod = new javax.swing.JButton();
69     classMethod.setFont(new Font("Century Schoolbook", Font.PLAIN, 16));
70

```

Entering Interface details

```

18 ~/
19 public class interfaceField extends javax.swing.JFrame {
20
21 /**
22  * Creates new form ClassField
23  */
24 public interfaceField() {
25     initComponents();
26 }
27
28 public interfaceField(String a ){
29     initComponents();
30     String g = "fields for class no. ";
31     String j = g+a + SelectClass.jk;
32     l1.setText(j);
33 }
34
35 /**
36  * This method is called from within the constructor to initialize the form.
37  * WARNING: Do NOT modify this code. The content of this method is always
38  * regenerated by the Form Editor.
39  */
40 @SuppressWarnings("unchecked")
41 // <editor-fold defaultstate="collapsed" desc="Generated Code"> //GEN-BEGIN: initComponents
42 private void initComponents() {
43
44     jLabel1 = new javax.swing.JLabel();
45     numberOfClassField = new javax.swing.JLabel();
46     numberOfClassField.setFont(new Font("Century Schoolbook", Font.PLAIN, 16));
47     t1 = new javax.swing.JTextField();
48     Ok = new javax.swing.JButton();
49     Ok.setFont(new Font("Century Schoolbook", Font.PLAIN, 16));
50     l1 = new javax.swing.JLabel();
51     l1.setText("[Blank]");
52
53     setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
54
55     jLabel1.setFont(new Font("Century Schoolbook", Font.BOLD, 18)); // NOI18N

```

Setting Interface fields

```

14 public class interfaceMethodNumber extends javax.swing.JFrame {
15
16     //public int methodFlag;
17
18 /**
19  * Creates new form ClassField
20  */
21 public interfaceMethodNumber() {
22     initComponents();
23 }
24
25 public interfaceMethodNumber(String s) {
26     initComponents();
27     String g = "no. of methods for class no: ";
28     String j = g+s + SelectClass.jk;
29
30     l1.setText(s);
31 }
32
33
34 /**
35  * This method is called from within the constructor to initialize the form.
36  * WARNING: Do NOT modify this code. The content of this method is always
37  * regenerated by the Form Editor.
38  */
39 @SuppressWarnings("unchecked")
40 // <editor-fold defaultstate="collapsed" desc="Generated Code"> //GEN-BEGIN: initComponents
41 private void initComponents() {
42
43     jLabel1 = new javax.swing.JLabel();
44     jLabel2 = new javax.swing.JLabel();
45     jLabel2.setFont(new Font("Century Schoolbook", Font.PLAIN, 16));
46     t1 = new javax.swing.JTextField();
47     jButton1 = new javax.swing.JButton();
48     jButton1.setFont(new Font("Century Schoolbook", Font.PLAIN, 16));
49
50     setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

```

Setting Interface Methods

```
93 classMethod.setText("Class Method");
94 classMethod.setVisible(false);
95 classMethod.addActionListener(new java.awt.event.ActionListener() {
96     public void actionPerformed(java.awt.event.ActionEvent evt) {
97         classMethodActionPerformed(evt);
98     }
99 });
100
101 lblInheritanceChildClass = new JLabel("INTERFACE IMPLEMENT");
102 lblInheritanceChildClass.setFont(new Font("Century Schoolbook", Font.BOLD, 17));
103
104 btnNewButton = new JButton("Confirm Name");
105 btnNewButton.setFont(new Font("Century Schoolbook", Font.PLAIN, 16));
106 btnNewButton.addActionListener(new ActionListener() {
107     public void actionPerformed(ActionEvent e) {
108
109         next.setVisible(true);
110         addClassField.setVisible(true);
111         classMethod.setVisible(true);
112
113         lblNewLabel.setText("IMPLEMENT CLASS HEADER DEFINED ");
114         getClassName();
115         enterName.setVisible(false);
116         t1.setVisible(false);
117         btnNewButton.setVisible(false);
118     }
119 });
120
121 lblNewLabel = new JLabel("[Blank]");
122 lblNewLabel.setFont(new Font("Century Schoolbook", Font.PLAIN, 16));
123
124 javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
125 layout.setHorizontalGroup(
126     layout.createParallelGroup(Alignment.LEADING)
127         .addGroup(layout.createSequentialGroup()
128             .addContainerGap()
129             .addComponent(l1, GroupLayout.DEFAULT_SIZE, 595, Short.MAX_VALUE))
130         .addGroup(layout.createSequentialGroup()
131             .addGroup(layout.createParallelGroup(Alignment.LEADING)
```

Interface Child implementation

```

37  /*public childInherit(String n,String i ){
38      initComponents();
39      // l2.setText(n+"");
40      l1.setText(i+".1");
41  }*/
42
43  public void getCname () {
44
45  }
46
47
48  /**
49   * This method is called from within the constructor to initialize the form.
50   * WARNING: Do NOT modify this code. The content of this method is always
51   * regenerated by the Form Editor.
52   */
53  @SuppressWarnings("unchecked")
54  // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN: initComponents
55  private void initComponents() {
56
57      enterName = new javax.swing.JLabel();
58      enterName.setFont(new Font("Century Schoolbook", Font.PLAIN, 16));
59      next = new javax.swing.JButton();
60      next.setFont(new Font("Century Schoolbook", Font.PLAIN, 15));
61      next.setVisible(false);
62      t1 = new javax.swing.JTextField();
63      l1 = new javax.swing.JLabel();
64      l1.setFont(new Font("Century Schoolbook", Font.PLAIN, 16));
65      addClassField = new javax.swing.JButton();
66      addClassField.setFont(new Font("Century Schoolbook", Font.PLAIN, 16));
67      classMethod = new javax.swing.JButton();
68      classMethod.setFont(new Font("Century Schoolbook", Font.PLAIN, 16));
69

```

Parent Inheritance Implementation

```

21 public class childInherit extends javax.swing.JFrame {
22     static String jk = "";
23
24     /**
25      * Creates new form SelectClass
26      */
27     public childInherit() {
28         initComponents();
29     }
30
31     public childInherit(String n){
32         initComponents();
33         l1.setText(n+"");
34     }
35
36     /*public childInherit(String n,String i ){
37         initComponents();
38         // l2.setText(n+"");
39         l1.setText(i+".1");
40     }*/
41
42
43
44     /**
45      * This method is called from within the constructor to initialize the form.
46      * WARNING: Do NOT modify this code. The content of this method is always
47      * regenerated by the Form Editor.
48      */
49     @SuppressWarnings("unchecked")
50     // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN: initComponents
51     private void initComponents() {
52
53         enterName = new javax.swing.JLabel();
54         enterName.setFont(new Font("Century Schoolbook", Font.PLAIN, 16));
55         next = new javax.swing.JButton();
56         next.setFont(new Font("Century Schoolbook", Font.PLAIN, 15));
57         next.setVisible(false);
58         t1 = new javax.swing.JTextField();
59

```

Child Inheritance Implementation

```

21 public class parentAbstract extends javax.swing.JFrame {
22     static String jk = "";
23     static String parentinh = "";
24
25     /**
26      * Creates new form SelectClass
27      */
28     public parentAbstract() {
29         initComponents();
30     }
31
32     public parentAbstract(String n ){
33         initComponents();
34         l1.setText(n+"");
35     }
36
37     /*public childInherit(String n,String i ){
38         initComponents();
39         // l2.setText(n+"");
40         l1.setText(i+".1");
41     }*/
42
43     public void getCname () {
44
45
46     }
47
48     /**
49      * This method is called from within the constructor to initialize the form.
50      * WARNING: Do NOT modify this code. The content of this method is always
51      * regenerated by the Form Editor.
52      */
53     @SuppressWarnings("unchecked")
54     // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN: initComponents
55     private void initComponents() {
56
57         enterName = new javax.swing.JLabel();
58         enterName.setFont(new Font("Century Schoolbook", Font.PLAIN, 16));
59         next = new javax.swing.JButton();
60         next.setFont(new Font("Century Schoolbook", Font.PLAIN, 15));
61         next.setVisible(false);

```

Abstract Parent implementation

```

30import javax.swing.GroupLayout.Alignment;
13
14 public class Abstractmethodnumber extends javax.swing.JFrame {
15
16     //public int methodFlag;
17
18     /**
19      * Creates new form ClassField
20      */
21     public Abstractmethodnumber() {
22         initComponents();
23     }
24
25     public Abstractmethodnumber(String s) {
26         initComponents();
27         //String g = "no. of methods for class no: ";
28         //String j = g+s + SelectClass.jk;
29
30         l1.setText(s);
31     }
32
33
34     /**
35      * This method is called from within the constructor to initialize the form.
36      * WARNING: Do NOT modify this code. The content of this method is always
37      * regenerated by the Form Editor.
38      */
39     @SuppressWarnings("unchecked")
40     // <editor-fold defaultstate="collapsed" desc="Generated Code"> //GEN-BEGIN: initComponents
41     private void initComponents() {
42
43         jLabel1 = new javax.swing.JLabel();
44         jLabel2 = new javax.swing.JLabel();
45         jLabel2.setFont(new Font("Century Schoolbook", Font.PLAIN, 16));
46         t1 = new javax.swing.JTextField();
47         jButton1 = new javax.swing.JButton();
48         jButton1.setFont(new Font("Century Schoolbook", Font.PLAIN, 16));
49
50         setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

```

No. Of Abstract Methods

```

19 public class abstractMethod extends javax.swing.JFrame {
20     static String printt = "";
21     /**
22      * Creates new form Method
23      */
24     public abstractMethod() {
25         initComponents();
26     }
27
28     public abstractMethod(String n){
29         initComponents();
30         //l1.setText(n+".1");
31
32         String h = "method details of class ";
33         String g = h+n ;
34
35         l1.setText(g);
36     }
37
38     /**
39      * This method is called from within the constructor to initialize the form.
40      * WARNING: Do NOT modify this code. The content of this method is always
41      * regenerated by the Form Editor.
42      */
43     @SuppressWarnings("unchecked")
44     // <editor-fold defaultstate="collapsed" desc="Generated Code"> //GEN-BEGIN: initComponents
45     private void initComponents() {
46
47         MethodDetails = new javax.swing.JLabel();
48         l1 = new javax.swing.JLabel();
49         enterMethod = new javax.swing.JLabel();
50         enterMethod.setFont(new Font("Century Schoolbook", Font.PLAIN, 17));
51         t1 = new javax.swing.JTextField();
52         returnType = new javax.swing.JLabel();
53         returnType.setFont(new Font("Century Schoolbook", Font.PLAIN, 17));
54         t2 = new javax.swing.JTextField();
55
56         setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

```

Abstract Method details


```

21 public class childAbstract extends javax.swing.JFrame {
22     static String jk = "";
23
24     /**
25      * Creates new form SelectClass
26      */
27     public childAbstract() {
28         initComponents();
29     }
30
31     public childAbstract(String n ){
32         initComponents();
33         l1.setText(n+"");
34     }
35
36     /*public childInherit(String n,String i ){
37         initComponents();
38         // l2.setText(n+"");
39         l1.setText(i+".1");
40     }*/
41
42     public void getCname () {
43
44     }
45
46
47     /**
48      * This method is called from within the constructor to initialize the form.
49      * WARNING: Do NOT modify this code. The content of this method is always
50      * regenerated by the Form Editor.
51      */
52     @SuppressWarnings("unchecked")
53     // <editor-fold defaultstate="collapsed" desc="Generated Code"> //GEN-BEGIN: initComponents
54     private void initComponents() {
55
56         enterName = new javax.swing.JLabel();
57         enterName.setFont(new Font("Century Schoolbook", Font.PLAIN, 16));
58     }

```

Abstract child

4.3 Constraints, Alternatives and Tradeoffs

The following is a table of the design constraints that the system SHALL meet. The list of constraints was produced from the initial project documentation provided during requirement elicitation.

Table of Design Constraints

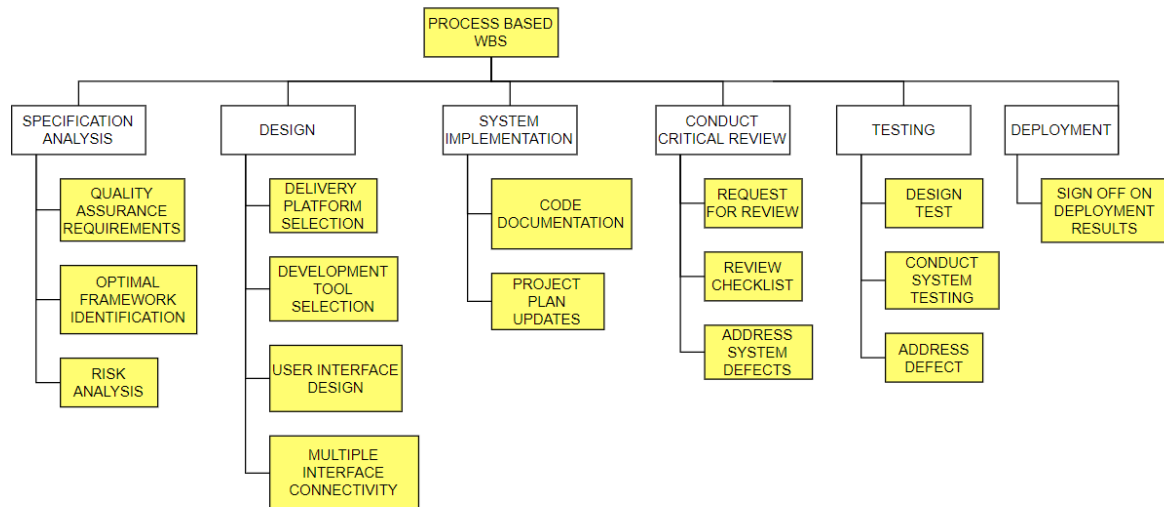
I D	Origin	Shall Requirement
1	Project Description Document	The system SHALL not be able to produce code that can be correctly compiled as the final code is drawn from user logic inputs
2	Project Description Document	The system SHALL not debug code errors other than few standard ones as its function is to provide closest user desired java code rather than system compiled code
3	Project Description Document	The system SHALL not rectify errors regarding scope of variables in a method as the resultant code is based on merely user inputs.
4	Project Description Document	The system SHALL not guarantee whether correct logical relationship between classes are drawn or not as it does not check

	<i>Document</i>	<i>user logic.</i>
5	<i>Project Description Document</i>	<i>The system SHALL not rectify if user has chosen wrong access specifier for the chosen class/method which might lead to accessibility errors as it does not limit user input</i>
6	<i>Project Description Document</i>	<i>The system SHALL not guarantee correct method overridden when it comes to Interface/abstract classes as it is user dependent</i>
7	<i>Project Description Document</i>	<i>The system SHALL not guarantee correct implementation of multiple subclasses of a single super class</i>

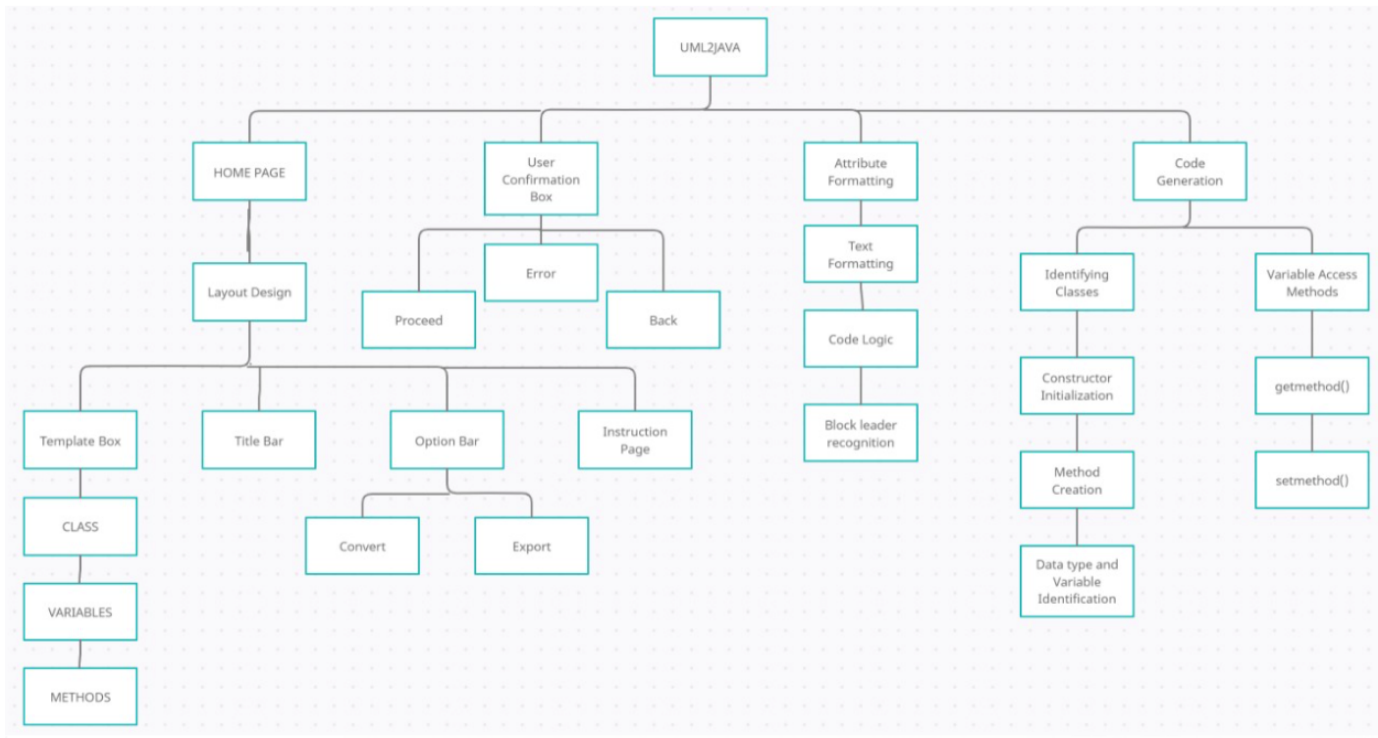
5 SCHEDULES AND TASKS.

5.1 Work Breakdown

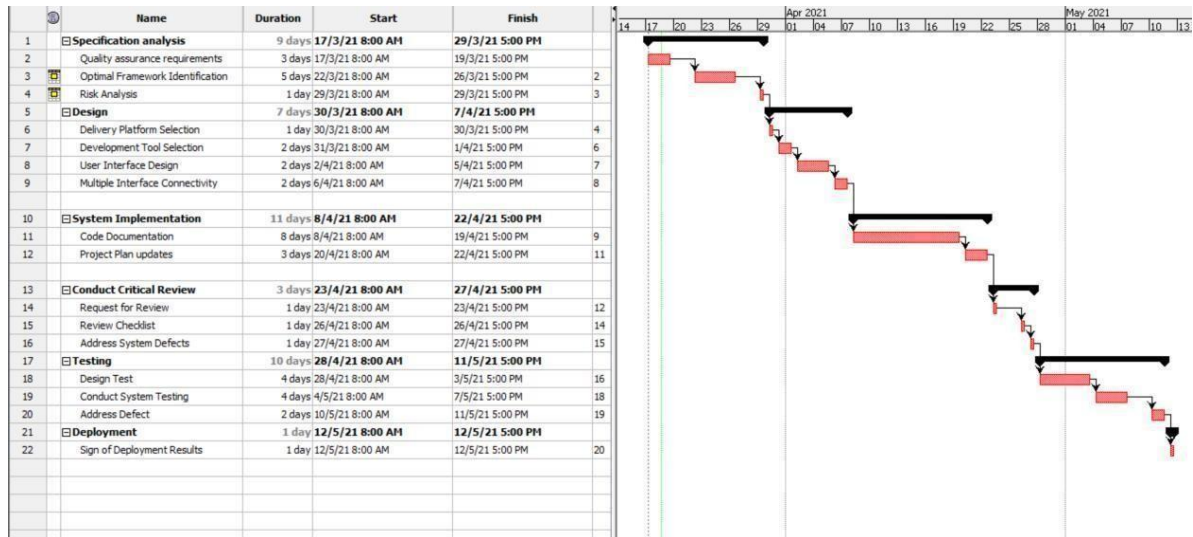
Process based:



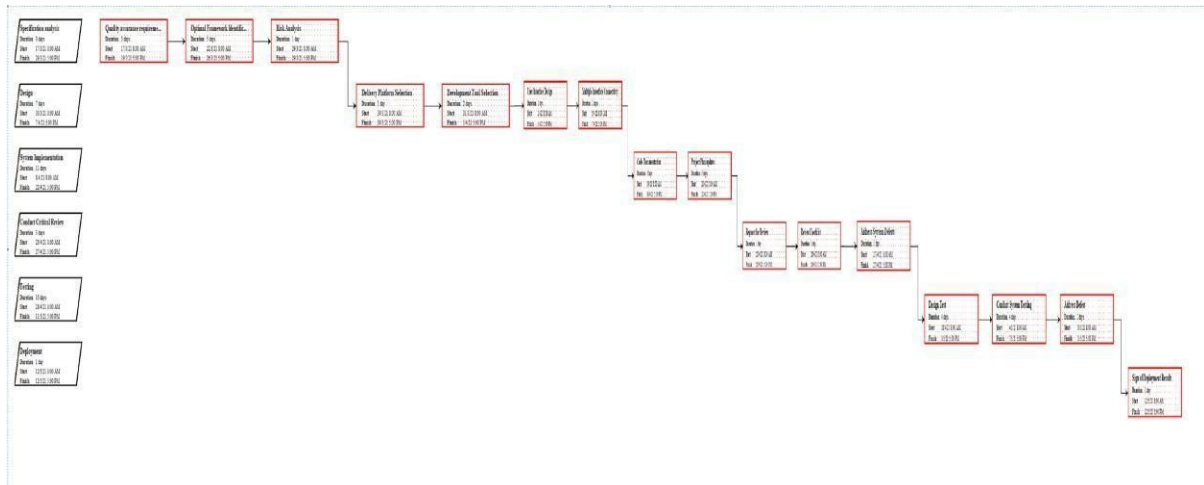
Product Based:



5.3 Gantt Chart



5.4 PERT Diagram



6 PROJECT DEMONSTRATION

a) GUI

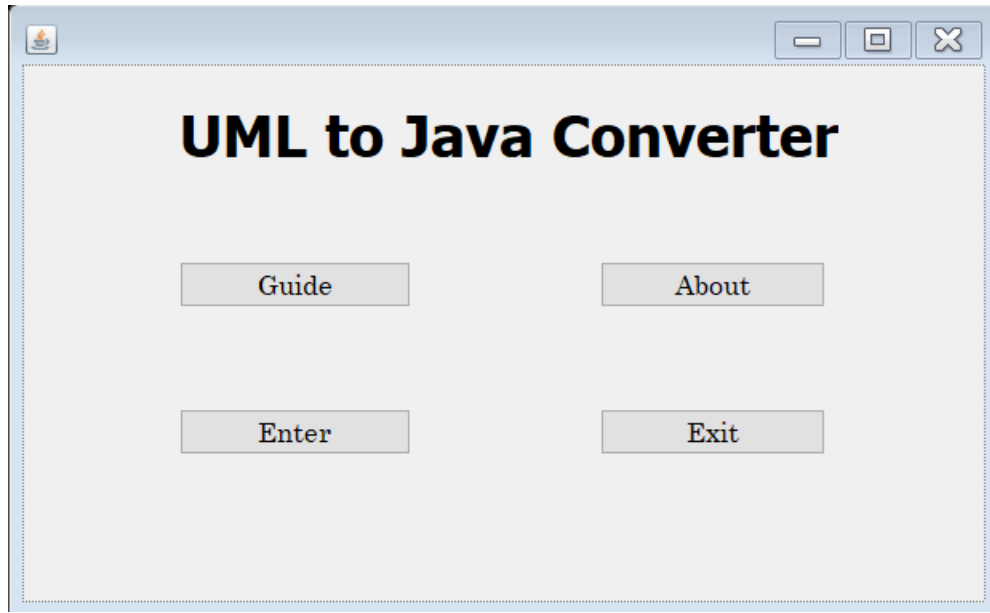


Figure 1:- Home Page

In figure 1, the home gives the user four buttons to press. On clicking on guide the user gets redirected to the documentation and tutorial page when they can read up on how to use the software to get the output code by entering the respective UML. The about page redirects the user to a page with the developer details where they can check the frequently asked questions and contact the developers for any queries. The exit button does as suggested and exits the program.

Finally if the user wishes to proceed to enter the UML they can click on the enter button.

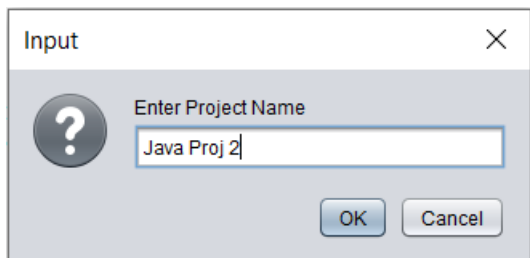


Figure 2:- Project name input box

The enter button redirects the user to the project name entry page where they can enter the project name as shown in figure 2.

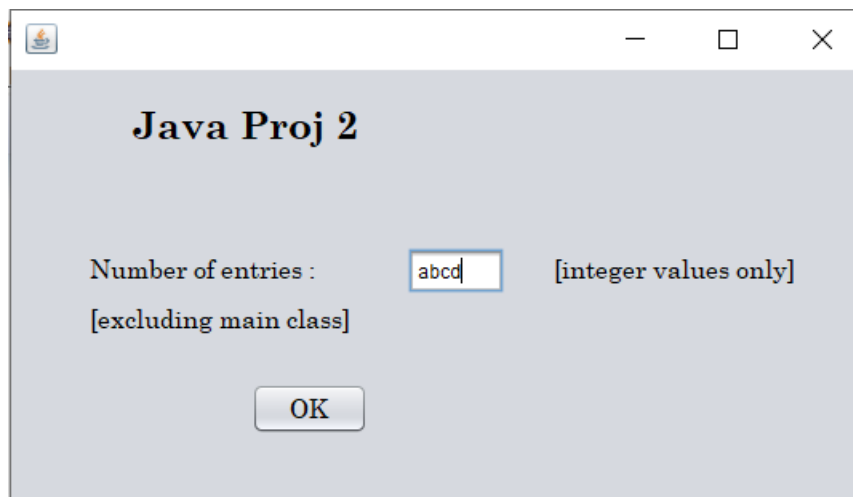


Figure 3:- Number of entries intake window with illegal entry

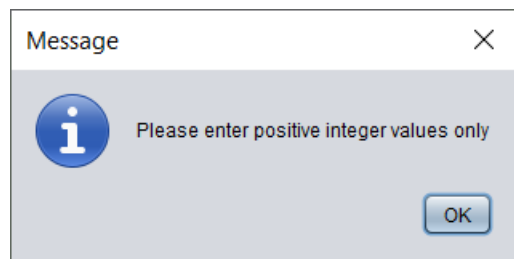


Figure 4:- Illegal entry pop up window

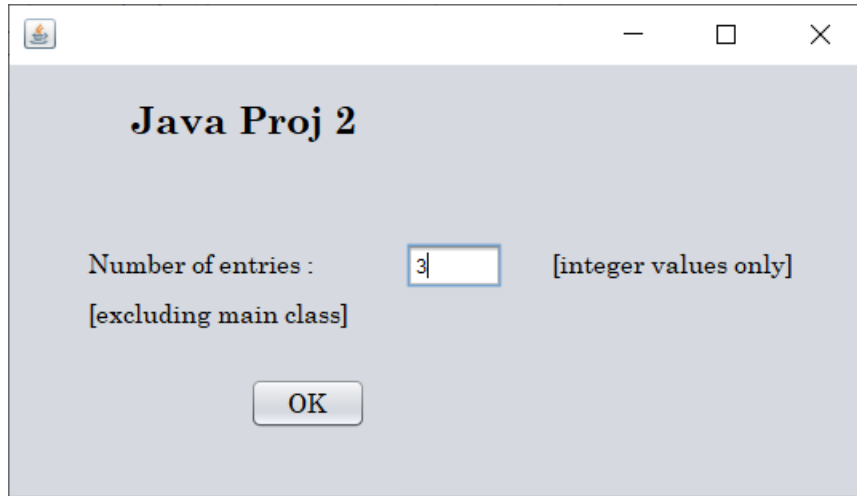


Figure 5:- Number of entries window

After entering the project name, the number of entries box opens up and the project name gets displayed on top as shown in figure 5. On entering the wrong values as shown in figure 3, a pop up appears warning the user to enter only positive values as shown in figure 4.

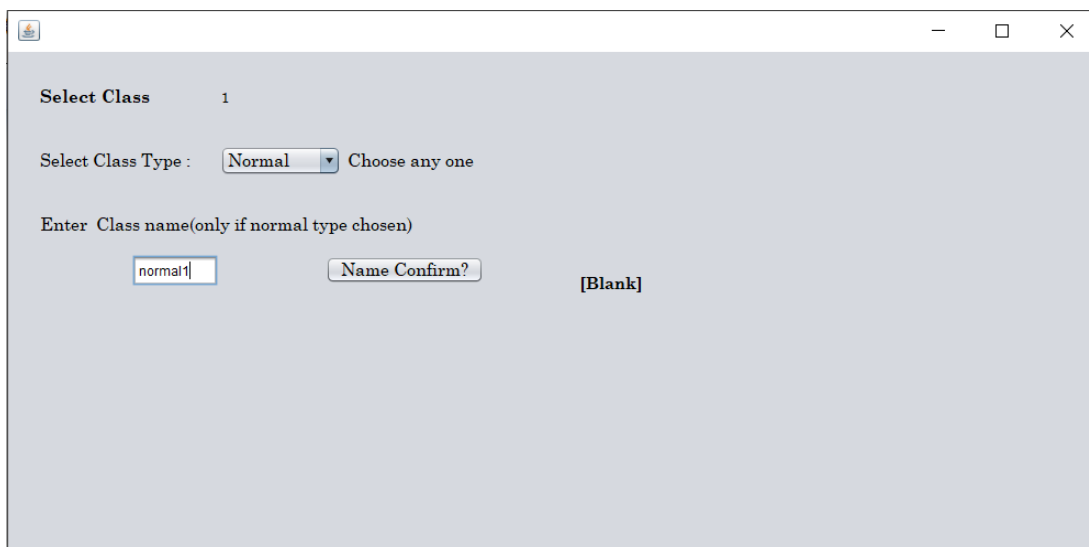
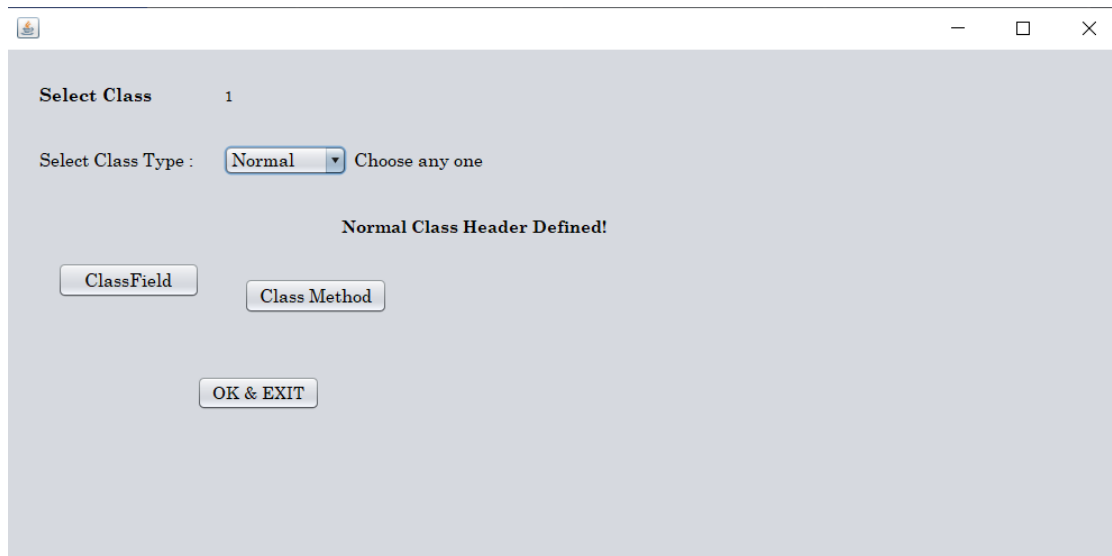


Figure 6:- Class entry window

In the previous window, on clicking OK, the class entry window opens up(figure 6) and the class number is displayed on the upper left corner for user convenience. Here the user can select the class type in a drop down box and enter the class name and then

go on to confirm these entries. As done before, the class number is shown in the top left corner of the form.



Select Class 1

Select Class Type : Normal Choose any one

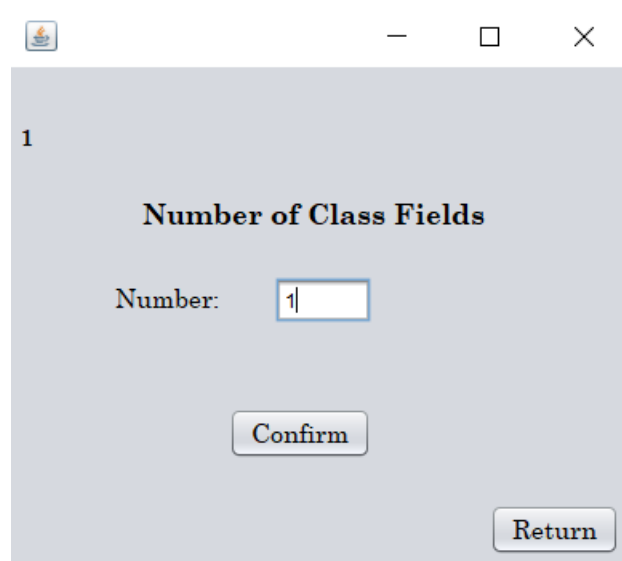
Normal Class Header Defined!

ClassField Class Method

OK & EXIT

Figure 7:- Class entry window after name confirmation

After clicking on the name confirm button, the name confirm button is disabled and a message shows up on the screen displaying a message saying '[CLASS TYPE] defined!' Other 3 buttons for class field entry, class method entry and ok & exit get enabled. As shown in figure 7.



1

Number of Class Fields

Number: 1

Confirm

Return

Figure 8:- Number of class fields entry window.

In figure 7, on clicking on the enter class method, the form in figure 8 opens up and the user can enter the number of class fields and proceed to confirm the entry. The form number is displayed in the top left corner.

1

Enter Field

Name: normalField

Return Type: integer

Access Modifier: public

Attributes Confirm

[Blank]

Figure 9:- Field entry window

After confirming the number of entries the respective number of field entry windows open up as shown in figure 9, here the user can enter the name, return type and the access modifier type and then proceed to confirm these entered values.

1

Enter Field

Name: normalField

Return Type: integer

Access Modifier: public

Add get/set?

OK & EXIT

[Blank]

Figure 10:- Field entry window after attribute confirmation

In figure 9 after confirmation, the add get/set methods button gets enabled along with the ok & exit button as shown in figure 10.

1

Number of Methods

Number:

Figure 11:- Number of methods intake window

In figure 7, on clicking on the class method button the form in figure 11 opens up for entering the number of methods to be entered.

1

Method

Name:

Return Type

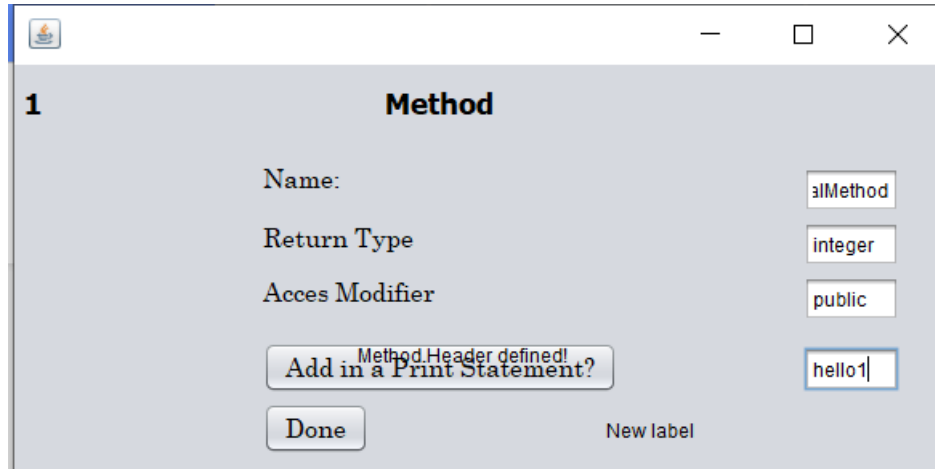
Access Modifier

[Blank]

New label

Figure 12:- Method entry box

The method entry box in figure 12 allows the user to enter the name return type and the access modifier of the method and confirm the entries. The method number is displayed in the top left of the form.



1 Method

Name:

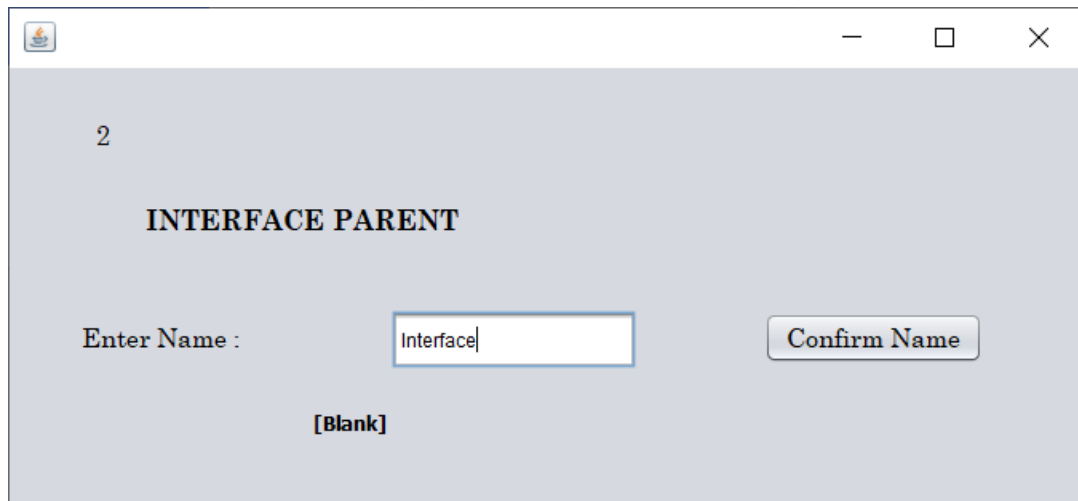
Return Type:

Access Modifier:

Method Header defined!
Add in a Print Statement?

Figure13:- Method entry box after attribute confirmation

In the figure 12 after confirming the method entries the add print statement button and the 'done' button get enabled. The first one allows the user to add a print method to the program while the second one allows the user to proceed further.



2

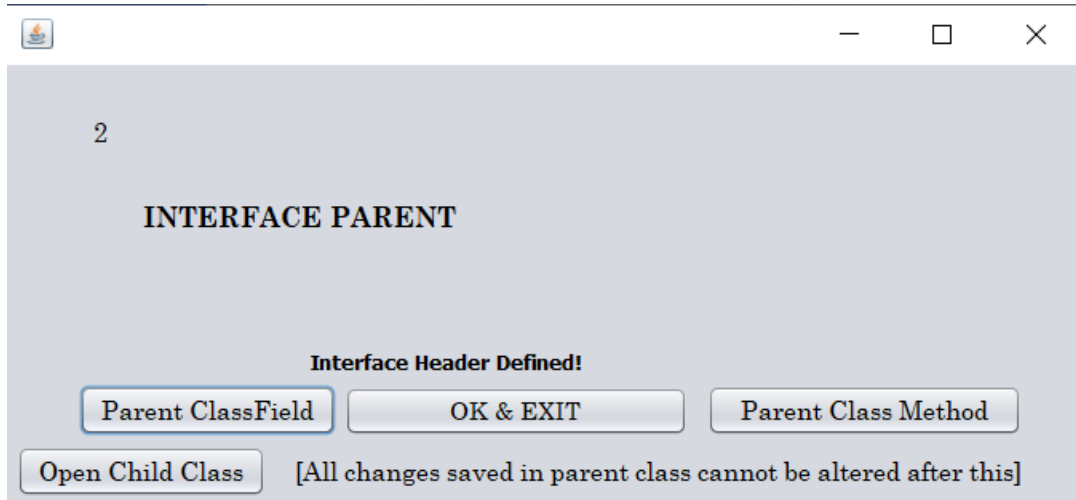
INTERFACE PARENT

Enter Name :

[Blank]

Figure 14:- Interface entry box

Once the user clicks in done in the previous form they can enter the parent interface name in figure 14 and go on to confirm the interface name.



2

INTERFACE PARENT

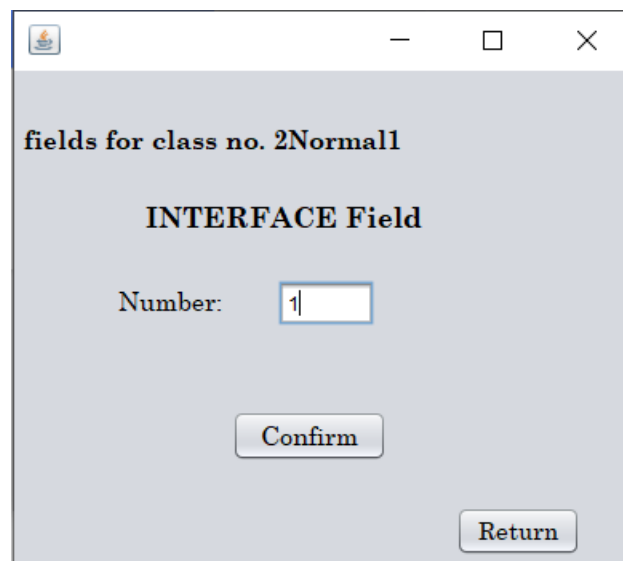
Interface Header Defined!

Parent ClassField OK & EXIT Parent Class Method

Open Child Class [All changes saved in parent class cannot be altered after this]

Figure 15:- Interface entry box after name confirmation

After the interface confirmation, a message saying interface defined is displayed in the form, the buttons for adding parent classified, parent class method and child class are enabled as shown in figure 15.



fields for class no. 2Normal1

INTERFACE Field

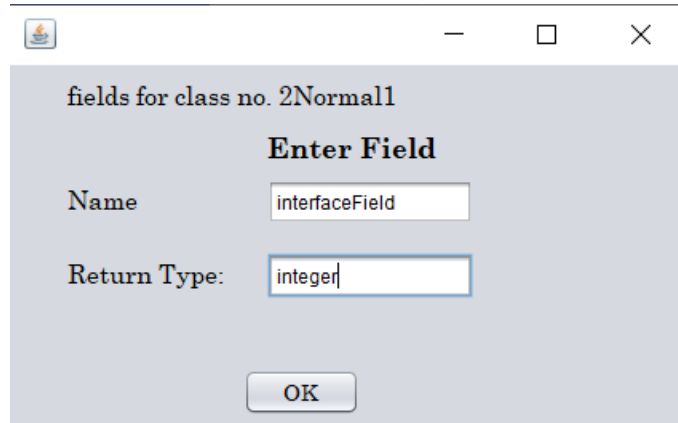
Number: 1

Confirm

Return

Figure 16:- Number of fields entry box for interface

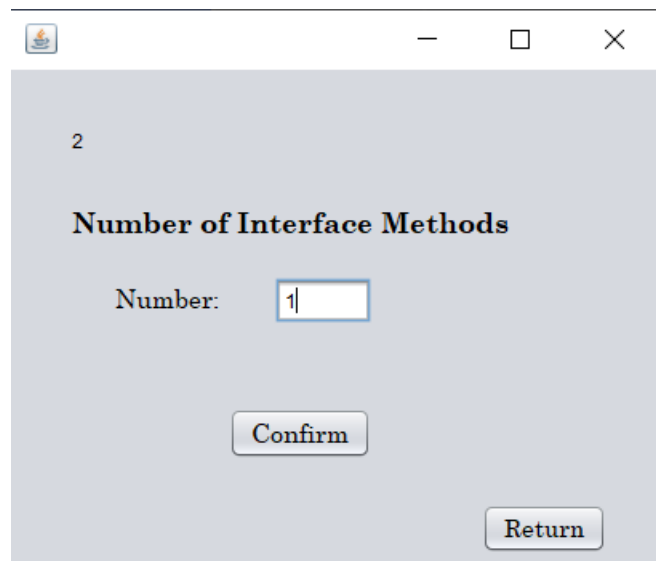
On clicking on the Parent classified button in figure 15, the interface field form opens up as shown in figure 16. Here the user can enter the number of fields and confirm the entry. The return button allows the user to return to figure 15.



A Java Swing window titled "fields for class no. 2Normal1". It contains a title "Enter Field". Below the title, there are two labels: "Name" and "Return Type:". The "Name" label is followed by a text input field containing "interfaceField". The "Return Type:" label is followed by a text input field containing "integer". At the bottom center of the window is an "OK" button.

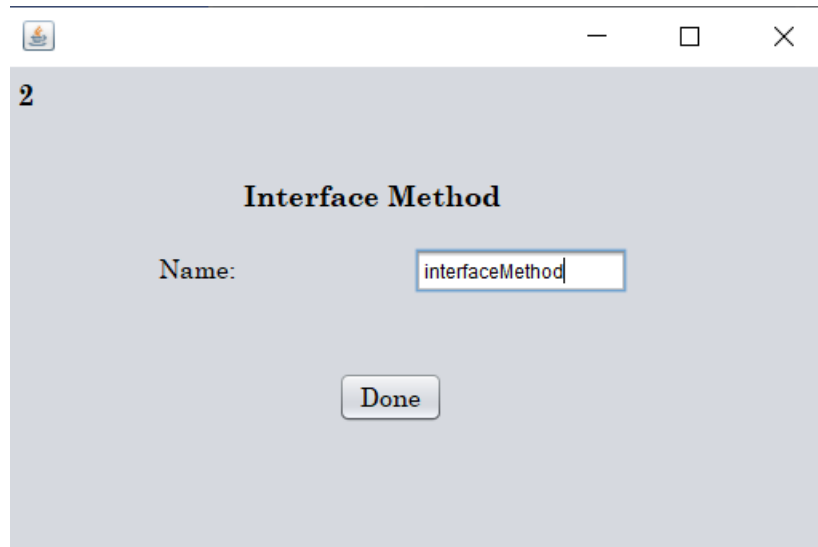
Figure 17:- Field entry box for interface

In Figure 16, after confirmation, the field entry box as shown in figure 17 is displayed on the user's screen where they can enter the name and return type of the parent classified. As shown in figure 17.



A Java Swing window with a title bar. Inside, the number "2" is displayed at the top left. Below it is the title "Number of Interface Methods". Under the title, there is a label "Number:" followed by a text input field containing "1". At the bottom center is a "Confirm" button, and at the bottom right is a "Return" button.

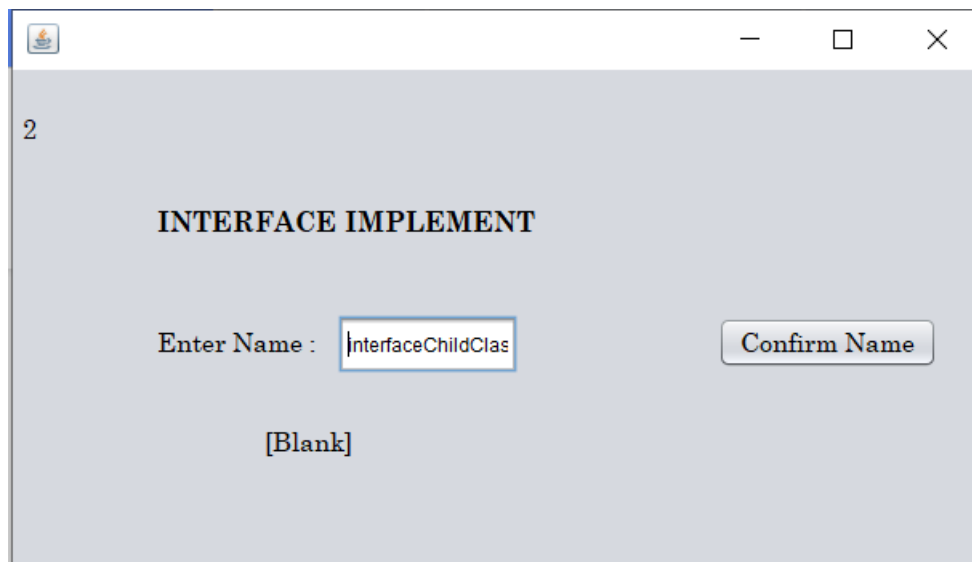
Figure 18:- Number of interface methods entry box



A screenshot of a software window titled "Interface Method". The window has a standard Windows-style title bar with a minimize button, a maximize button, and a close button. In the top-left corner of the window's content area, there is a small icon of a notepad and the number "2". The main content area has a light gray background. It features the title "Interface Method" in a bold, black, serif font. Below the title, there is a label "Name:" followed by a text input field containing the text "interfaceMethod". At the bottom center of the window, there is a button labeled "Done".

Figure 19:- Interface method entry box.

Figure 18 shows the form which opens when the user clicks on the add parent method button in figure 15. Which proceeds to work as the previous method forms.



A screenshot of a software window titled "INTERFACE IMPLEMENT". The window has a standard Windows-style title bar with a minimize button, a maximize button, and a close button. In the top-left corner of the window's content area, there is a small icon of a notepad and the number "2". The main content area has a light gray background. It features the title "INTERFACE IMPLEMENT" in a bold, black, serif font. Below the title, there is a label "Enter Name :" followed by a text input field containing the text "InterfaceChildClas". To the right of the input field, there is a button labeled "Confirm Name". Below the input field and button, there is a label "[Blank]".

Figure 20:- Implemented interface entry box

After the user is done with all the entries in figure 15, the implemented class form opens up as shown in figure 20.

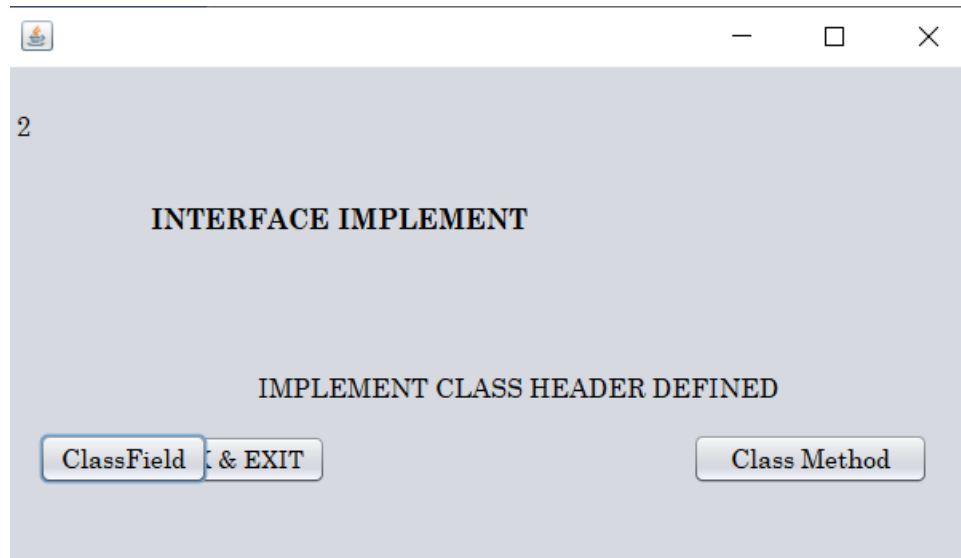


Figure 21:- Implemented interface entry box after confirmation

Figure 21 opens by clicking on confirm in the enter implemented class form. Here the user can proceed to add class methods and fields which work as explained above in Figure 8 and Figure 11 respectively.

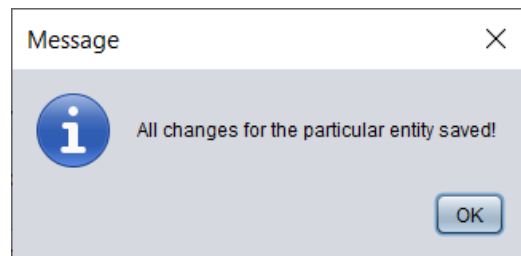


Figure 22:- All changes saved dialog box

Figure 22 is a pop up window which displays when the user clicks on save and exit in figure 21. It shows when all the changes get saved.

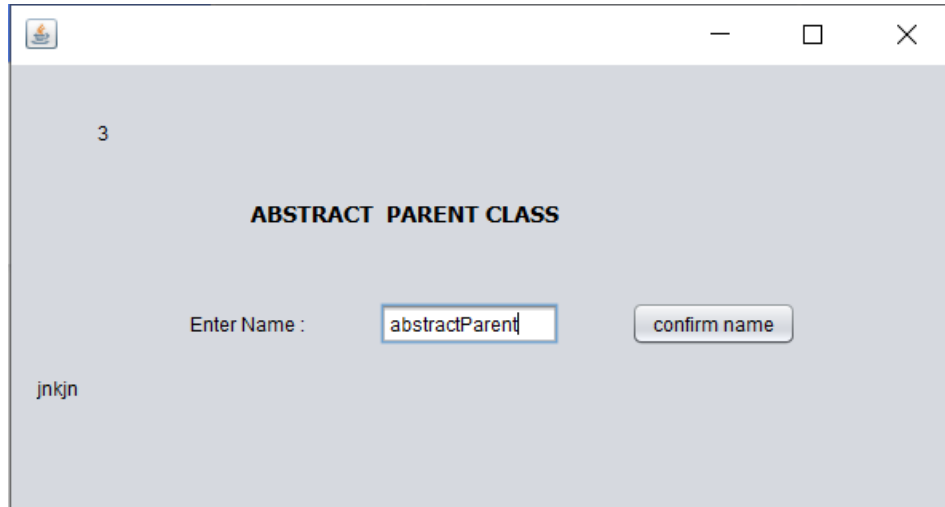


Figure 23:- Abstract parent entry box

If the user selects the class type as abstract in figure 6, the abstract parent entry box pops up as shown in figure 23. Here the user can enter the name and confirm it.

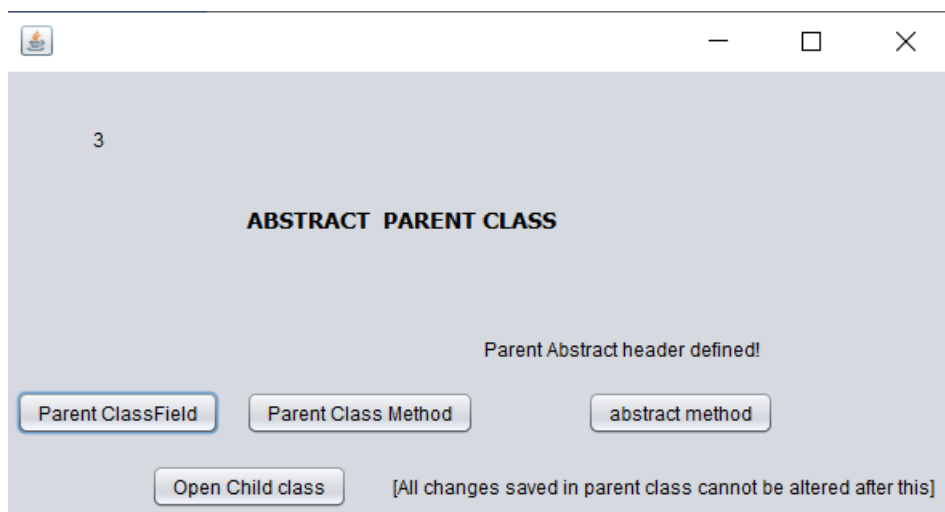
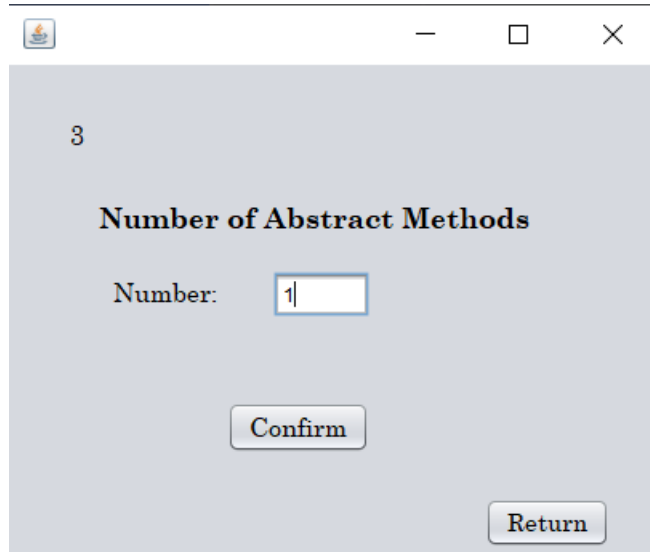


Figure 24:- Abstract parent entry box after name confirmation

On confirming the name in figure 23, the parent classified, parent method ,abstract class and open child class buttons become available to the user.



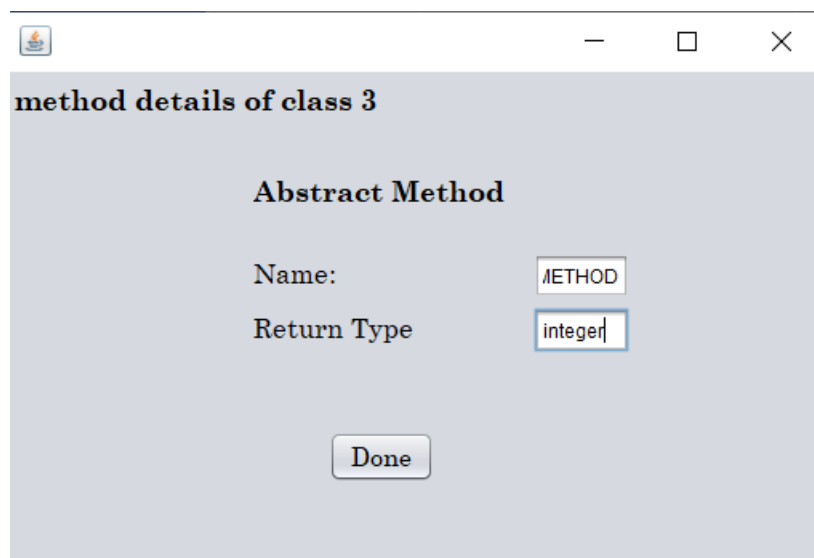
3

Number of Abstract Methods

Number:

Figure 25:- Number of abstract methods entry box

The form in figure 25 opens up on pressing the abstract method button in figure 24. Here the user can enter the number of abstract methods to be entered.



method details of class 3

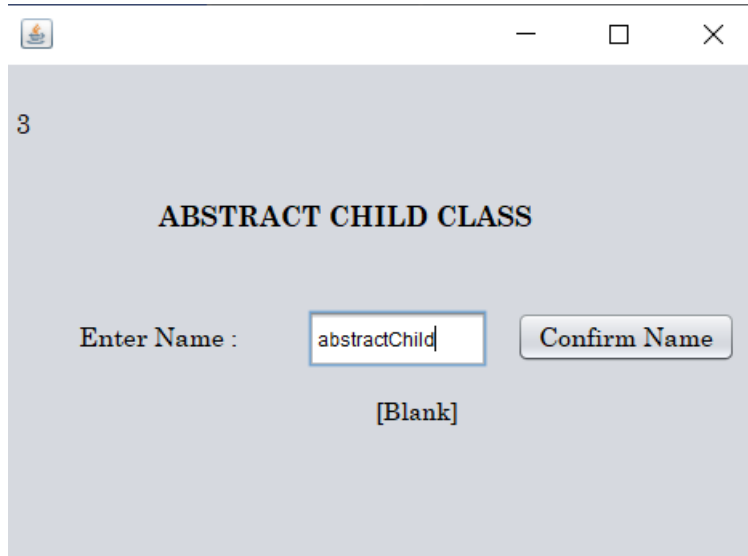
Abstract Method

Name:

Return Type

Figure 26:- Abstract method entry box

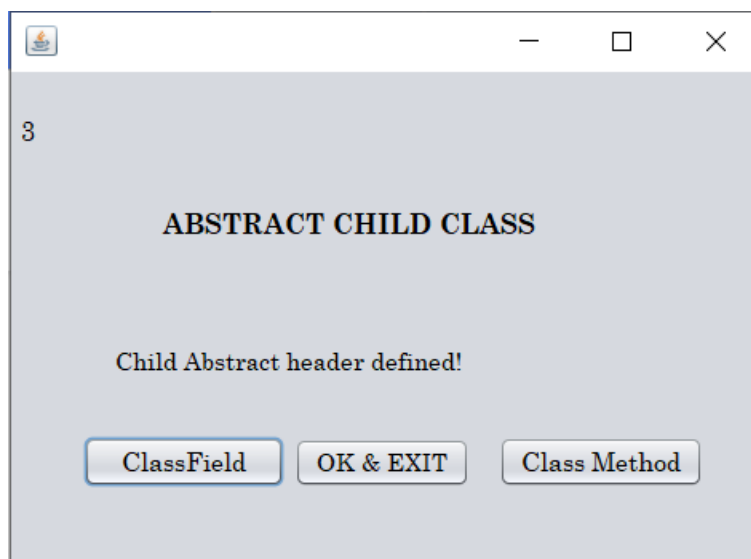
After entering the number of abstract methods, the form in figure 26 opens up which allows the user to enter the name and return type of the abstract method.



A screenshot of a Java Swing window titled '3'. The window has a light gray background and a standard title bar with minimize, maximize, and close buttons. In the center, the text 'ABSTRACT CHILD CLASS' is displayed in bold. Below this, there is a label 'Enter Name :'. To the right of the label is a text input field containing the text 'abstractChild'. To the right of the input field is a button labeled 'Confirm Name'. Below the input field, the text '[Blank]' is displayed.

Figure 27:- Abstract child entry box

On clicking the abstract child class in figure 24, the form in figure 27 opens up which allows the user to enter the abstract child class name and confirm it.



A screenshot of the same Java Swing window titled '3'. The window has a light gray background and a standard title bar. In the center, the text 'ABSTRACT CHILD CLASS' is displayed in bold. Below this, the text 'Child Abstract header defined!' is displayed. At the bottom, there are three buttons: 'ClassField', 'OK & EXIT', and 'Class Method'.

Figure 28:- Abstract class entry box after name confirmation

After entering the abstract child class name in figure 27 the enter class field and class method buttons become available to the user as shown in figure 28. The add ClassField and ClassMethod buttons work as shown in figures 8 and 11 respectively.

<terminated> MainJFrame [Java Application] C:\Program Files\Java\jre1.8.0_291\bin\javaw.exe (17-May-2021, 2:13:50 AM)

```
class Normal1 {
public integer normalField ;
public integer getnormalField() {
return normalField; }
public void setnormalField(integer newAssign ) {
this.normalField = newAssign ; }
public integer normalMethod() {
}
}
interface Interface {
public static final integer interfaceField;
public void interfaceMethod() ;
}
class interfaceChildClas implements Interface {
public integer interfaceChildField ;
public integer interfaceChildMethod() {
}
}
abstract class abstractParent {
public integer abstractField ;
public integer abstractMethod() {
}
}
public abstract integer METHOD() ;
}
class abstractChild extends abstractParent {
public integer abstractChildField ;
public integer abstractChildMethod() {
}
}
}
```

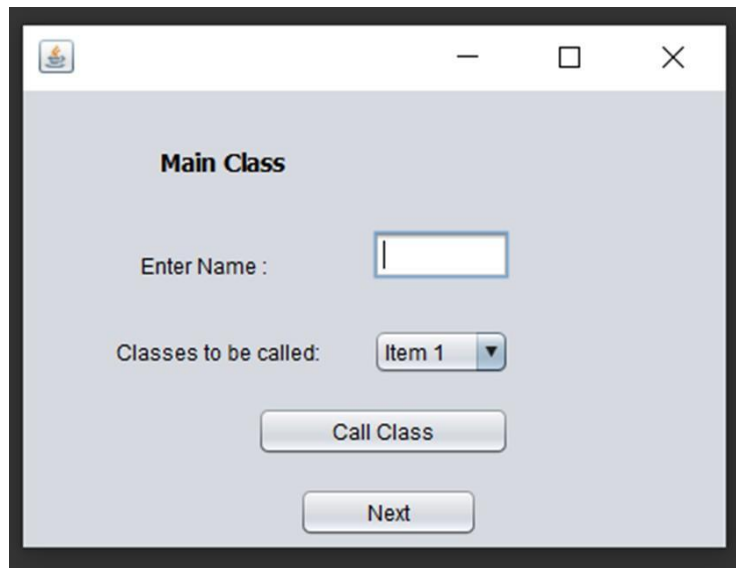
OUTPUT:

Figure 29:- Sample code output

Finally figure 29 shows us the output code the user receives in the form of a text file once they have completely entered the whole UML into the program.


b) Testing

In UML 2 JAVA program, we can see that from the test cases that the program fails in the completeness check. This is because of the failure of the main method which is used to define the main method and use methods from other classes. This error is clearly shown in the test case below.



The screenshot shows a Java application window titled "Main Class". Inside the window, there is a text input field labeled "Enter Name :". Below it, there is a dropdown menu labeled "Classes to be called:" with "Item 1" selected. At the bottom, there are two buttons: "Call Class" and "Next".

Test Objective	Test Data	Expected Results	Actual Results	Test Pass/Fail
To check if Main class can call objects from other class	committing all input values after final attribute filling confirmation of last class	Psvm method calling objects	Error output with code failing logic	Fail



1

INHERITANCE PARENT CLASS

Parent super class header defined!

Parent ClassField

OK & EXIT

Parent Class Method

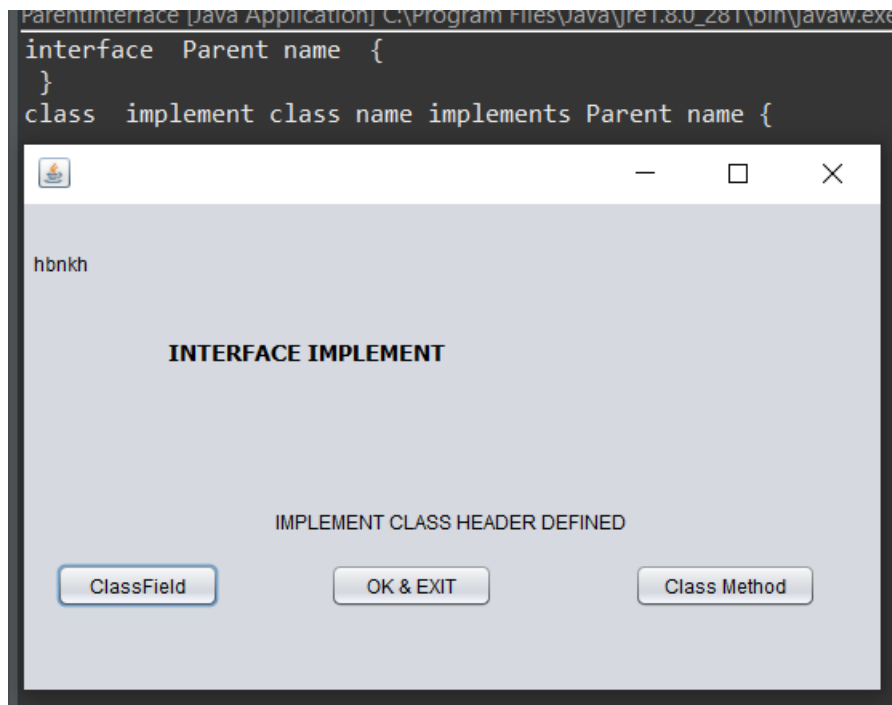
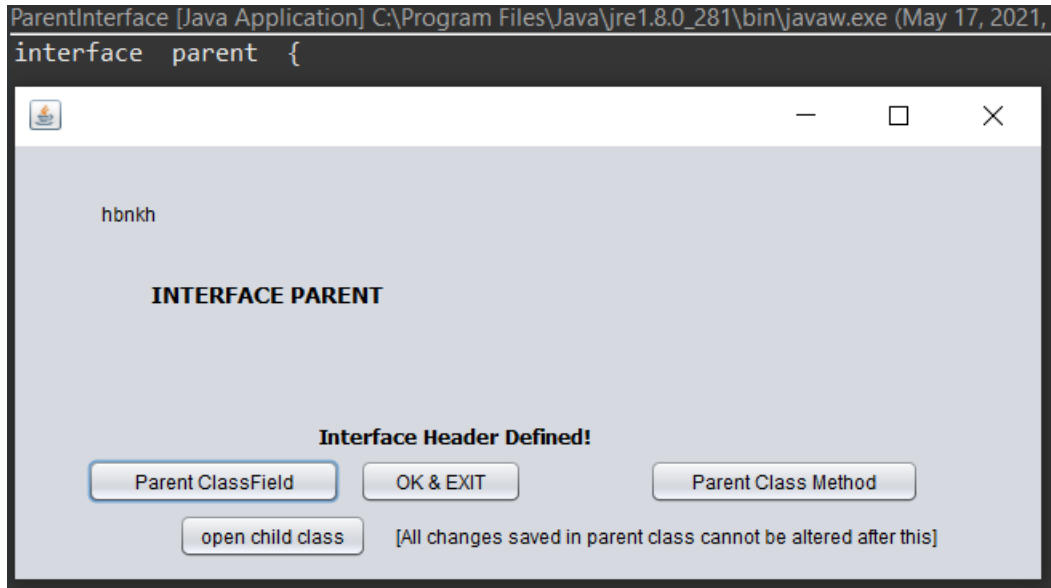
Open child class

[All changes saved in parent class cannot be altered after this]

Test Objective	Test Data	Expected Results	Actual Results	Test Pass/Fail
To check whether All subsequent form labels should get numbered automatically when there are multiple classes to avoid confusion	Number of entries set to 4	Respective Class field and class methods get numbered according to the the Class number	Respective Class field and class methods got numbered according to the the Class number	Pass

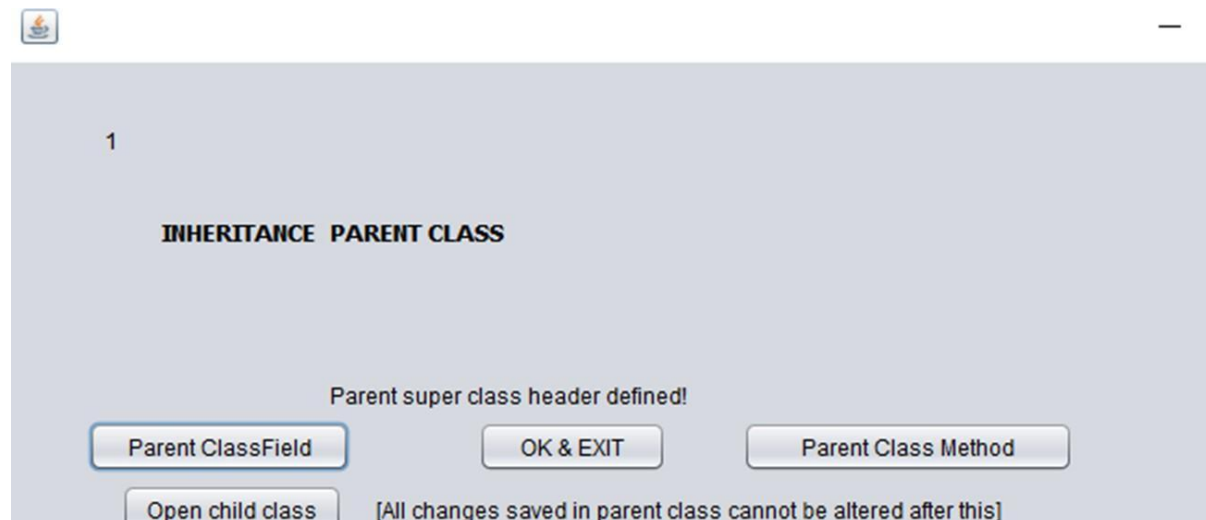
Consistency check

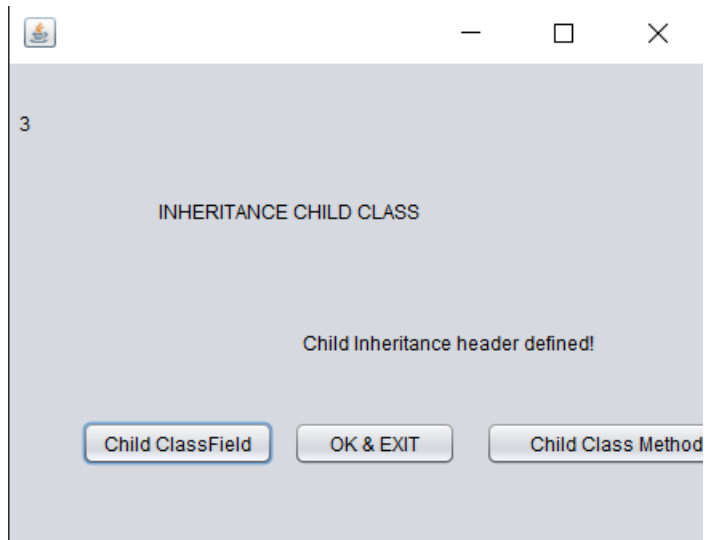
When the consistency of the program was checked using test cases, there has been a problem noticed in the child. As per the test cases, the child class had a contradiction during creation when the parent classes were not created.



Test Objective	Test Data	Expected Results	Actual Results	Test Pass/Fail
To check whether Implement class can be added or not on opting for Interface Class type	Choose Interface class type from Select class type form	Should open parent form and give choice for implement class	Opened parent form and gave choice for implement class	Pass
To check whether Child class can be added or not on opting for	Choose Inheritance class type from Select class type form	Opens parent form and should give choice for Child class	Opened parent form and gave choice for Child class	Pass

Inheritance Class type				
An option whether to add Child class or not on opting for Abstract Class type	Abstract class type chosen from Select class type form	Opens parent form and should give choice for Child class	Opened parent form and gave choice for Child class	Pass





Test Objective	Test Data	Expected Results	Actual Results	Test Pass/Fail
To check whether Child Class header gets confirmed after confirm button selection to populate class code further with fields and methods	Select “ Child class ” after selecting “ parent class ” and enter “ class name ”	“ Child class Header Defined ” should show up in label	“Child Class Header Defined” shows up in label	Pass

Ambiguity check

The requirements in this program are expressed using terms which are clearly defined. The functionalities are straightforward, and the requests sent to user are understood by any user who knows the parts of a uml diagram.



Test Objective	Test Data	Expected Results	Actual Results	Test Pass/Fail
To check whether “Enter Project Name” JOptionPane pops up after selecting Enter JButton.	Select Enter JButton from MainJFrame	“Enter Project Name” JOptionPane should pop up	“Enter Project Name” JOptionPane should popped up	Pass

Verification check

When the consistency of the program was checked using test cases, there has been a problem noticed in the child. The child class cannot be created anytime unless the child class has the parent class already created. This error also happens when number of class fields have been entered as 'zero'.

Select Class 1

Select Class Type : Normal ▼ Choose any one

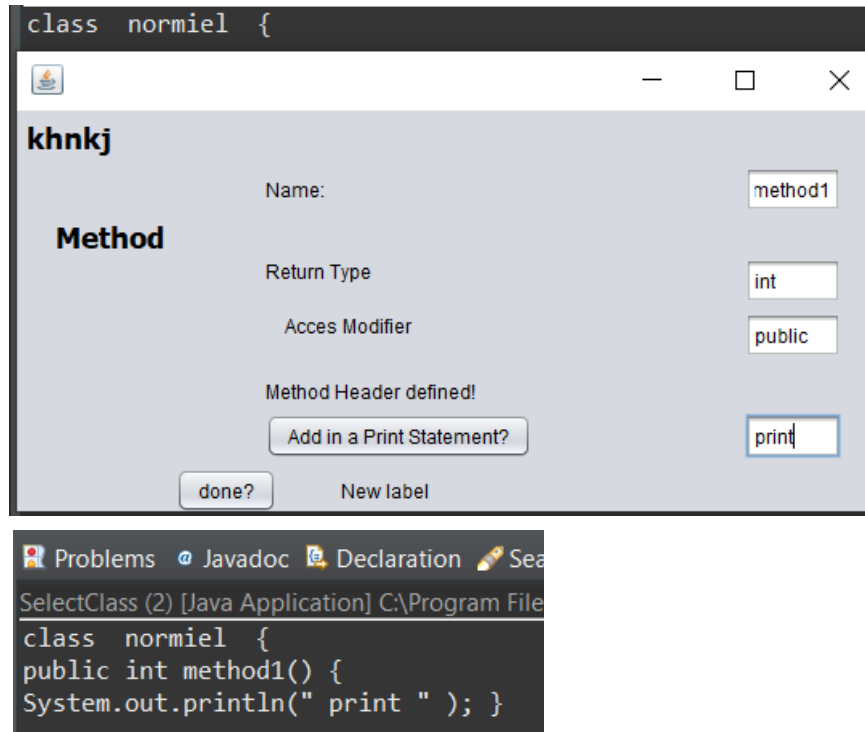
Normal Class Header Defined!

ClassField Class Method

OK & EXIT

Test Objective	Test Data	Expected Results	Actual Results	Test Pass/Fail
To check whether Form disposes on selecting "OK&EXIT" button	"OK&EXIT" Button clicked	The particular form should dispose with all input values being saved	Form disposed successfully with all changes saved	Pass
Number of class fields entered as zero	Entered "0"	"number of class fields cannot be	Form disposes without affecting	Fail

		zero” message should pop up	code similar to return button	
Number of class methods entered as zero	Entered “0”	“number of class methods cannot be zero” message should pop up	Form disposes without affecting code similar to return button	Fail



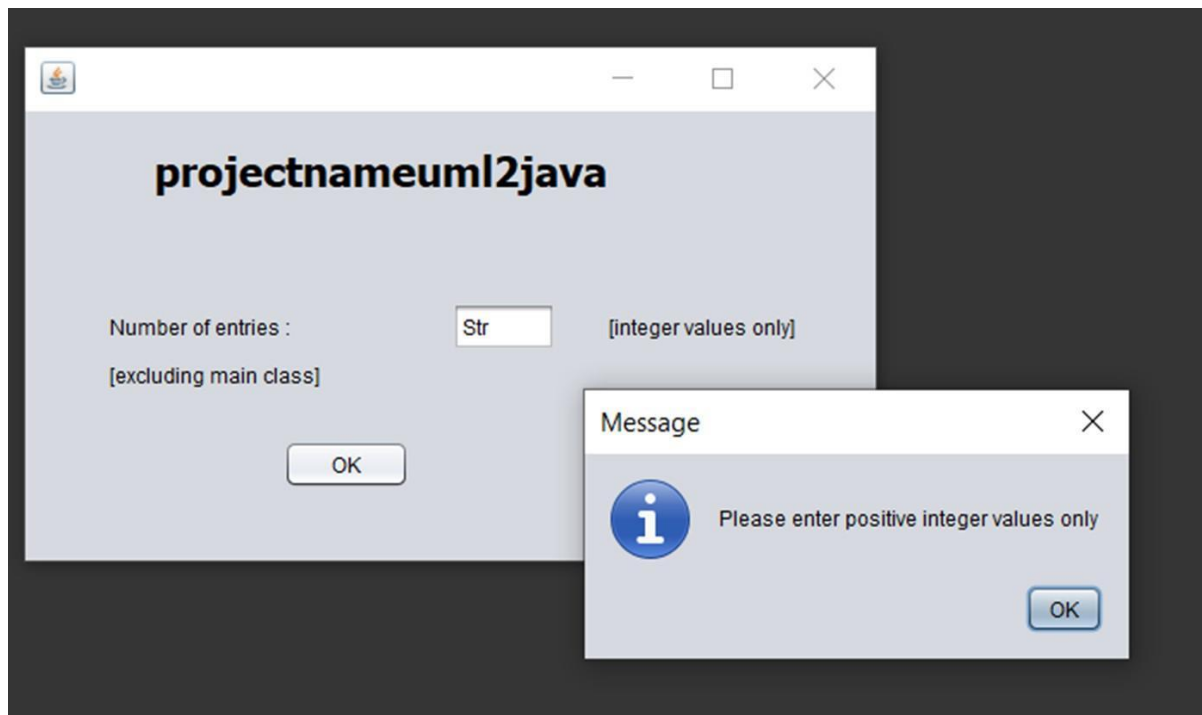
Test Objective	Test Data	Expected Results	Actual Results	Test Pass/Fail
To check whether changes in method form gets saved after selecting “done?” button	Select “done?” button	Output code file should get updated with method details	Method of the particular class gets updated with the input info	Pass
To check whether Multiple custom add print statements can be added	Select “Add in a Print Statement?” button	Multiple inputs should get allowed	Button disappears after single usage after confirming addition of first print method	Fail

Realism check

For the UML 2 JAVA program, the main objective was to “To generate the java code from uml diagram”. This is fulfilled and can be used in any desktop system which contains Windows, GNU/Linux or OS X as their operating system. The schedule was also followed according to the Gantt Chart given in the SRS Document.

Validity check

As we can see from the test cases below, we can try different test cases into the program. This means that these tests can demonstrate the system when given the input as an error. It will notify the user for all the inputs entered into the program



Test Objective	Test Data	Expected Results	Actual Results	Test Pass/Fail
To check “ No.of entries ” text field is accepting any String or not	“ ABC ”	“Enter only positive integers” message should pop up	“Enter only positive integers” message should popped up	Pass
To check “ No.of entries ” text field is accepting 0 or negative or any other numeric positive datatype	0 or any negative integer(= -3) or any double value(= 4.00)	“Enter only positive integers” message should pop up	“Enter only positive integers” message should popped up	Pass

Select Class 1

Select Class Type : Normal ▾ Choose any one

Normal Class Header Defined!

ClassField Class Method

OK & EXIT

2

INTERFACE PARENT

Interface Header Defined!

Parent ClassField OK & EXIT Parent Class Method

open child class [All changes saved in parent class cannot be altered after this]

Test Objective	Test Data	Expected Results	Actual Results	Test Pass/Fail
To check whether Normal Class header gets confirmed after confirm button selection to populate class code further with fields and methods	Select " Normal " and Enter " class name "(eg: Studentava)	"Normal Class Header Defined" should show up in label And Class name should get set to Studentava	"Normal Class Header Defined" shows up in label And Class name gets set to Studentava	Pass

To check whether Interface Class header gets confirmed after confirm button selection to populate class code further with fields and methods	Select “Interface” and enter “class name”(eg: Studentava)	“Interface Header Defined” should show up in label And Class name should get set to Studentava	“Interface Class Header Defined” shows up in label And Class name gets set to Studentava	Pass
To check whether Inheritance Class header gets confirmed after confirm button selection to populate class code further with fields and methods	Select “Inheritance” and enter “class name”(eg: Studentava)	“Parent super class header defined” should show up in label And Class name should get set to Studentava	“Normal Class Header Defined” shows up in label And Class name gets set to Studentava	Pass
To check whether Abstract Class header gets confirmed after confirm button selection to populate class code further with fields and methods	Select “Abstract” and enter “ class name”(eg: Studentava)	“Abstract Class Header Defined” should show up in label And Class name should get set to Studentava	“Abstract Class Header Defined” shows up in label And Class name gets set to Studentava	Pass

hnkn

Number of Class Fields

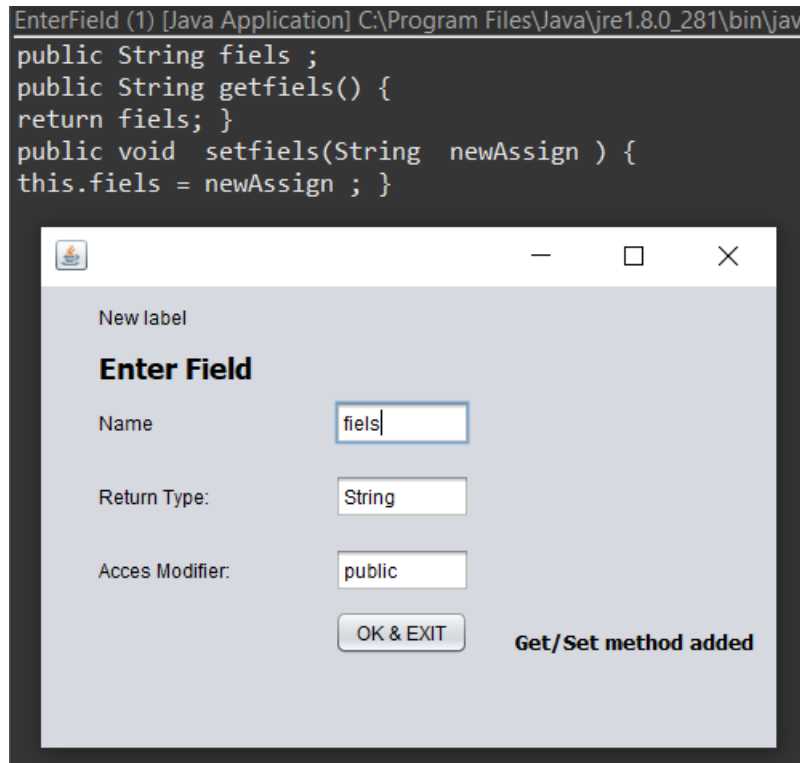
Number:

Confirm

Return

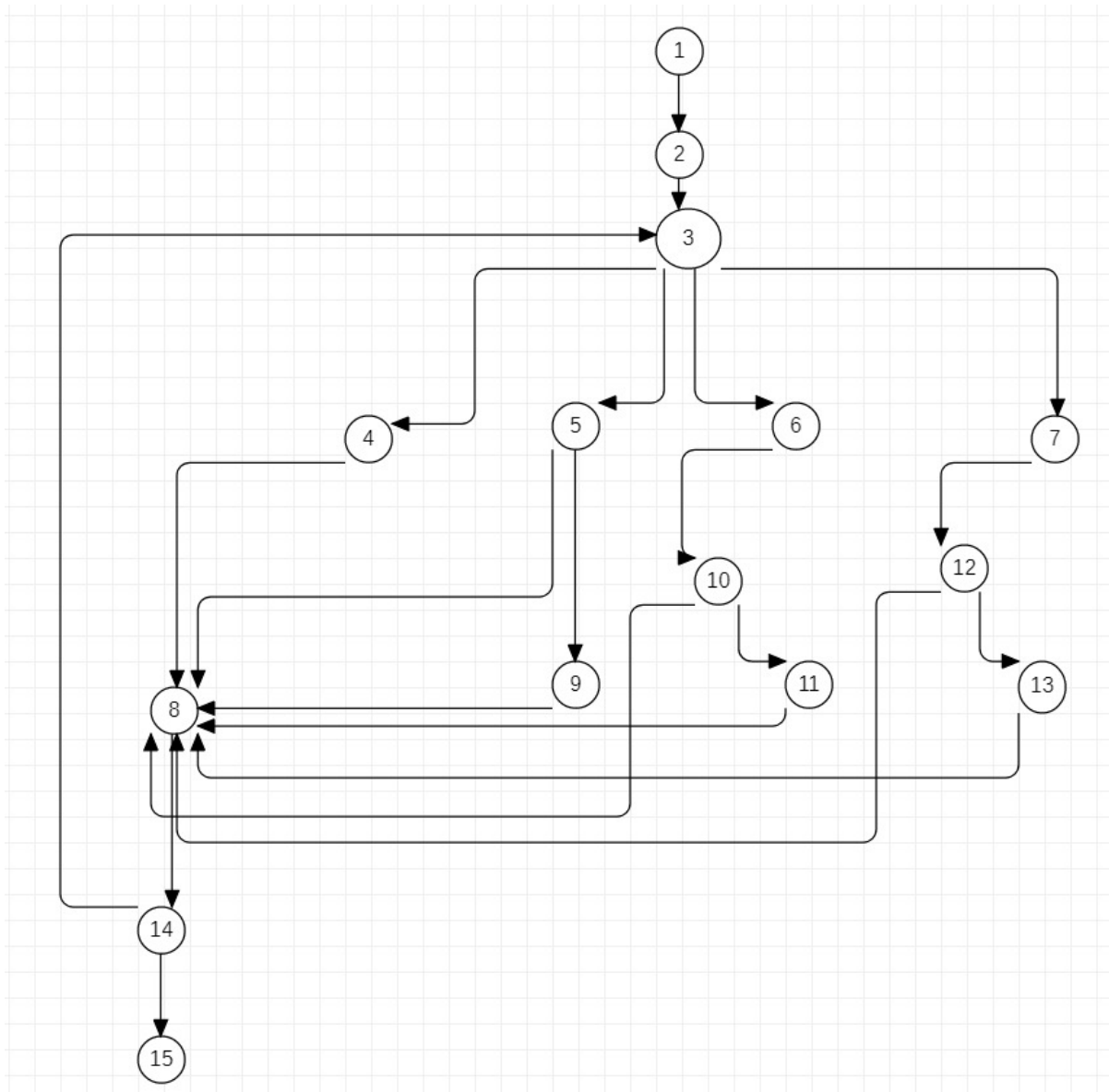
Test Objective	Test Data	Expected Results	Actual Results	Test Pass/Fail
To check whether “ Class field form ” gets disposed after	Select return button	Field Form should dispose without affecting code	Form disposed successfully without	Pass

selection of return button			affecting output code	
To check whether “ Class Method form ” gets disposed after selection of return button	Select return button	Method Form should dispose without affecting code	Form disposed successfully without affecting output code	Pass



Test Objective	Test Data	Expected Results	Actual Results	Test Pass/Fail
To check whether Custom Get/set method should appears only after confirming field attributes	Select Attributes confirm button	Add Get/set button should appear	Add Get/set button appeared	Pass
To check whether Access field members can be accessed using add get/set button	Select Add Get/set button	Get and set method should be added in the output code	Output code file populated with get and set method to access particular field members	Pass

1. Generate test cases for your system using basis path testing strategy.



- Path 1: 1,2,3,4,8,14,15
Path 2: 1,2,3,5,8,14,15
Path 3: 1,2,3,5,9,8,14,15
Path 4: 1,2,3,6,10,8,14,15
Path 5: 1,2,3,6,10,11,8,14,15
Path 6: 1,2,3,7,12,8,14,15
Path 7: 1,2,3,7,12,13,8,14,15
Path 8: 1,2,3,....,14,3, 14,15

By manual calculation the cyclomatic complexity was found out to be of medium complexity.

Test Case Id	Test Objective	Test Data	Expected Results	Actual Results	Test Pass/Fail
--------------	----------------	-----------	------------------	----------------	----------------

1.	To check whether "Enter Project Name" JOptionPane pops up after selecting Enter Jbutton.	Select Enter JButton from MainJFrame	"Enter Project Name" JOptionPane should pop up	"Enter Project Name" JOptionPane should popped up	Pass
----	-------------------------------------------------------------------------------------------------	--------------------------------------	----------------------------------------------------------	-------------------------------------------------------------	------

2.	To check whether Normal Class header gets confirmed after confirm button selection to populate class code further with fields and methods	Select "Normal" and Enter "class name"(eg: Studentava)	"Normal Class Header Defined" should show up in label And Class name should get set to Studentava	"Normal Class Header Defined" shows up in label And Class name gets set to Studentava	Pass
3.	To check whether Interface Class header gets confirmed after confirm button selection to populate class code further	Select "Interface" and enter "class name"(eg: Studentava)	"Interface Header Defined" should show up in label And Class name should get set to Studentava	"Interface Class Header Defined" shows up in label And Class name gets set to Studentava	Pass

	with fields and methods				
--	----------------------------	--	--	--	--

4.	To check whether Inheritance Class header gets confirmed after confirm button selection to populate class code further with fields and methods	Select "Inheritance" and enter "class name"(eg: Studentava)	"Parent super class header defined" should show up in label And Class name should get set to Studentava	"Normal Class Header Defined" shows up in label And Class name gets set to Studentava	Pass
5.	To check whether Abstract Class header gets confirmed after confirm button selection to populate class code further with fields and methods	Select "Abstract" and enter "class name"(eg: Studentava)	"Abstract Class Header Defined" should show up in label And Class name should get set to Studentava	"Abstract Class Header Defined" shows up in label And Class name gets set to Studentava	Pass

6.	To check whether Child Class header gets confirmed after confirm button selection to populate class code further with fields and methods	Select "Child class" after selecting "parent class" and enter "class name"	"Child class Header Defined" should show up in label	"Child Class Header Defined" shows up in label	Pass
----	-------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------	-------------------------------------------------------------	------------------------------------------------	------

7.	To check whether All subsequent form labels should get numbered automatically when there are multiple classes to avoid confusion	Number of entries set to 4	Respective Class field and class methods get numbered according to the the Class number	Respective Class field and class methods got numbered according to the the Class number	Pass
----	-----------------------------------------------------------------------------------------------------------------------------------------	-----------------------------------	------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------	------

8.	To check whether changes in method form gets saved after selecting " done? " button	Select " done? " button	Output code file should get updated with method details	Method of the particular class gets updated with the input info	Pass
----	--------------------------------------------------------------------------------------------	--------------------------------	---------------------------------------------------------	-----------------------------------------------------------------	------

9.	To check whether Implement class can be added or not on opting for Interface Class type	Choose Interface class type from Select class type form	Should open parent form and give choice for implement class	Opened parent form and gave choice for implement class	Pass
10.	To check whether Child class can be added or not on opting for Inheritance Class type	Choose Inheritance class type from Select class type form	Opens parent form and should give choice for Child class	Opened parent form and gave choice for Child class	Pass
11.	An option whether to add Child class or not on opting for Abstract Class type	Abstract class type chosen from Select class type form	Opens parent form and should give choice for Child class	Opened parent form and gave choice for Child class	Pass

12.	To check whether a choice is offered between abstract methods and normal methods in abstract parent class	Abstract class type chosen from Select class type form	"Abstract method" button should open different form other than normal method	"Abstract method" button opened different method form other than normal one where input fields were also different	Pass
-----	-----------------------------------------------------------------------------------------------------------	---------------------------------------------------------------	-------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------	------

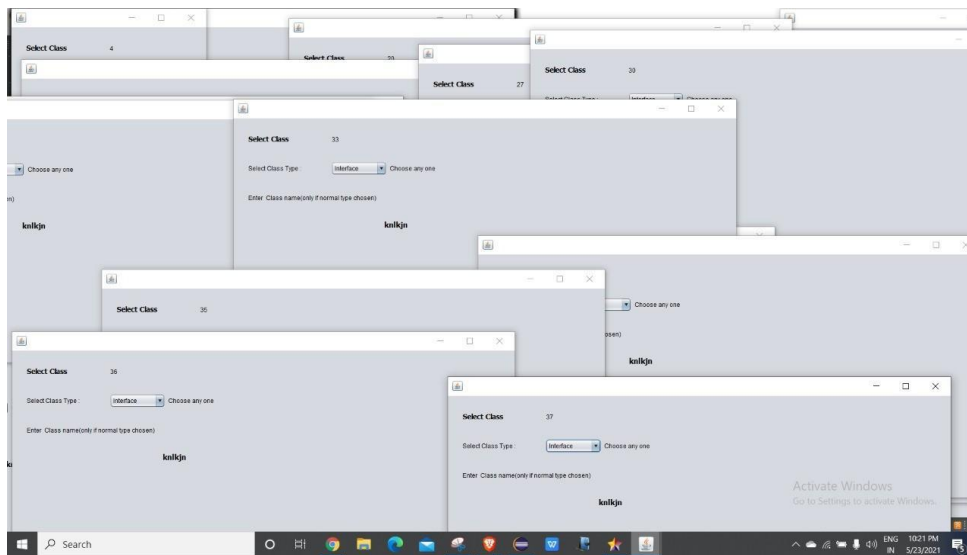
13.	To check Interface field if chosen should be declared separately	Select Parent Interface field button	Final static field should be declared in code	Final static field declared inside parent Interface	Pass
-----	-------------------------------------------------------------------------	---------------------------------------------	------------------------------------------------------	------------------------------------------------------------	------

2. Evaluate the performance of the system in terms of load, stress, security, deployment, usability and regression testing.

Load Testing:-

Load testing cannot be done for this application as it is meant to be used by a single user.

Stress test:-



The stress test we check if the software still works in extreme conditions. Here we requested the class entry form numerous times to see if the software still worked. The results obtained were

10 requests: Works fine

100 requests: 5 second delay

500 requests: 20 seconds approx. delay

1000 requests: 50 seconds approx. delay

1200 requests: system crashes

Security Testing:-

Since our project is a desktop application there aren't any encryption related security threats, but after a quality check using the help of SonarLint tool, it threw a security related bug and only one bug was detected.

Standard outputs should not be used directly to log anything (java:S106)

Code smell Major

When logging a message there are several important requirements which must be fulfilled:

- The user must be able to easily retrieve the logs
- The format of all logged message must be uniform to allow the user to easily read the log
- Logged data must actually be recorded
- Sensitive data must only be logged securely

If a program directly writes to the standard outputs, there is absolutely no way to comply with those requirements. That's why defining and using a dedicated logger is highly recommended.

Noncompliant Code Example

```
System.out.println("My Message"); // Noncompliant
```

Compliant Solution

```
logger.log("My Message");
```

See

- [CERT, ERR02-J](#), - Prevent exceptions while logging data

This issue was later resolved and now the software doesn't detect any security bugs in the code.

Usability test

1. Effectiveness : UML 2 JAVA program is very effective for students who are participating in hackathons and other competitive tests. As speed is a huge priority in such contests, this program helps in reducing the time deciphering and converting uml diagrams to java code.
2. Efficiency : UML 2 JAVA program uses only needed details which the user provides and converts that into java code. No extra information or space will be used in the process.
3. Engagement: This is a single user program where the student is the only personnel who interacts with the server side.
4. Error Tolerance : Usually the program runs into minute errors which is most probably because of the error in the user's input given to the program.
5. Ease of Learning : It is a user friendly program that can be used by any student knowing the components of uml diagrams.

7 a. COST ANALYSIS

Since this software is an app-based process and the only costing for the software would be app maintenance which could be recovered from advertising cost from advertisement on download link or app. The app would be provided free of cost to the users with unlimited usage.

7 b. RESULT & DISCUSSION

After several test cases the error is close to nil where syntactical error is zero but logic breakdown can occur from user(eg: in case of object data type)

For further developments a strong Image OCR tool should be developed which can directly extract text from any png/jpeg . The image can be of svg file after getting converted from a dot file that of an UML .It will take very less time to generate code . The major challenge is to segment symbols according to public,private or protected and to extract correct text. It all depends on the power of the OCR tool. The current best in free OCR tools is Tesseract which failed the test cases of our project, hence we used attribute formatting which requires more number of inputs from the user but it's accuracy is very much higher.

8.Conclusion

It is seen that a product as such can be beneficial for college students and new software developers alike. It can also help people in competitive environments by allowing them to write their code faster and provide them with a significant portion of the code. Since java is a platform independent language this also makes the project much more accessible and useful for people with different computing environments. The platform independence part also helps in cross platform sharing and running of the software.

9. References:

- [1] H. Chavez, W. Shen, R. France and B. A. Mechling, "An approach to testing Java implementation against its Unified Modeling Language", *Proc. ACM/IEEE Int. Conf. Model Driven Eng. Languages Syst.*, pp. 220-236, Oct. 2013.
- [2] W. Pires, J. Brunet, F. Ramalho and D. Serey, "Uml-based design test generation", *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing. Fortaleza*, pp. 735-740, 2008.
- [3] Akehurst, D., Howells, G., McDonald-Maier, K.: Implementing associations: UML 2.0 to Java 5. 0 to Java 5. *Software & Systems Modeling* 6(1), 3–35 (2007)
- [4] Foundation, T.: Eclipse Modeling - MDT - Home (2013)
- [5] Darga, P., Boyapati, C.: Efficient Software Model Checking of Data Structure Properties. In: *Proceedings of the 21st Annual ACM SIGPLAN Conference on Object-oriented Programming Systems, Languages, and Applications*, New York, NY, USA, pp. 363–382 (2006)
- [6] Charfi, A., Mraidha, C., Gérard, S., Terrier, F., Boulet, P.: Toward optimized code generation through model-based optimization. In: *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 1313–1316 (2010)