UML2JAVA

Submitted in partial fulfillment of the requirements for the degree of

Bachelor of Technology

in

Computer Science Engineering

by

Debalay Dasgupta **19BCE2423**

CSE3001_SOFTWARE ENGINEERING

DECLARATION

June, 2020

We hereby declare that the thesis entitled **UML2JAVA** 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 Swathi JN**.

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: 02.06.2021

Debalay
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 **Ms. Swathi JN, 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.

I*n addition, a thank you to Professor Dr. Swathi J.N., who introduced us to the Methodology of work, and whose passion for the "underlying structures" had lasting effect.*

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

Debalay Dasgupta

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 get generated.There are 4 types of classes included in the project namely; Normal,Inheritance,Abstract and Implement . This will help in covering almost all type 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 write huge lines of code or creating code templates in JAVA sharing 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**

List of Figures

List of Abbreviations

UML	Unified Modelling Language
GUI	Graphical User Interface

1 INTRODUCTION

1.1 Objective

The main goals of the our software are:

1. This software allows the user to choose data types of method/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. User 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 a 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 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 using which user 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 covert 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 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 concept 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 specifier.	
	(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

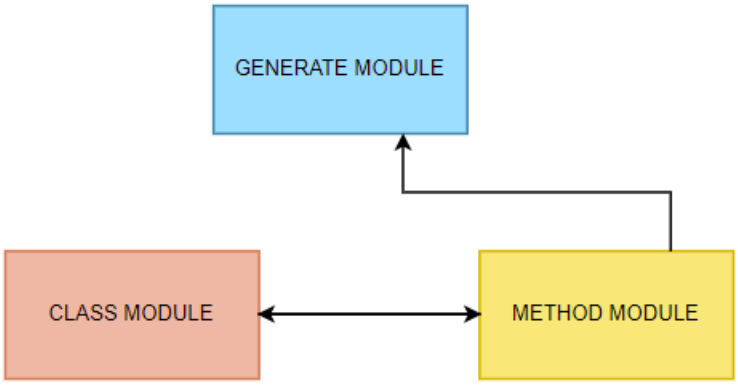


Fig: Data Flow Type Architectural Design

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

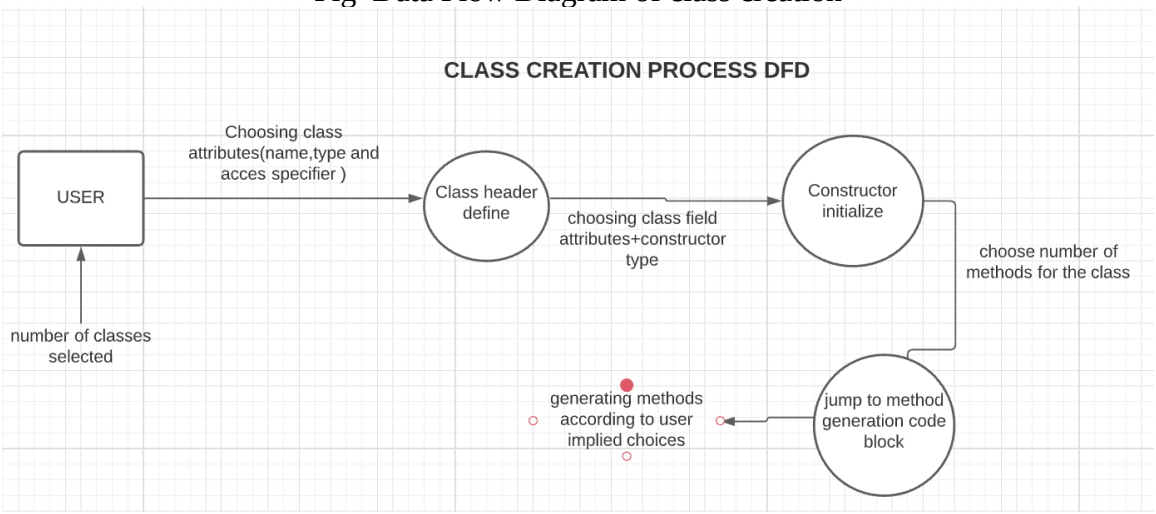


Fig: Data Flow Diagram of method creation

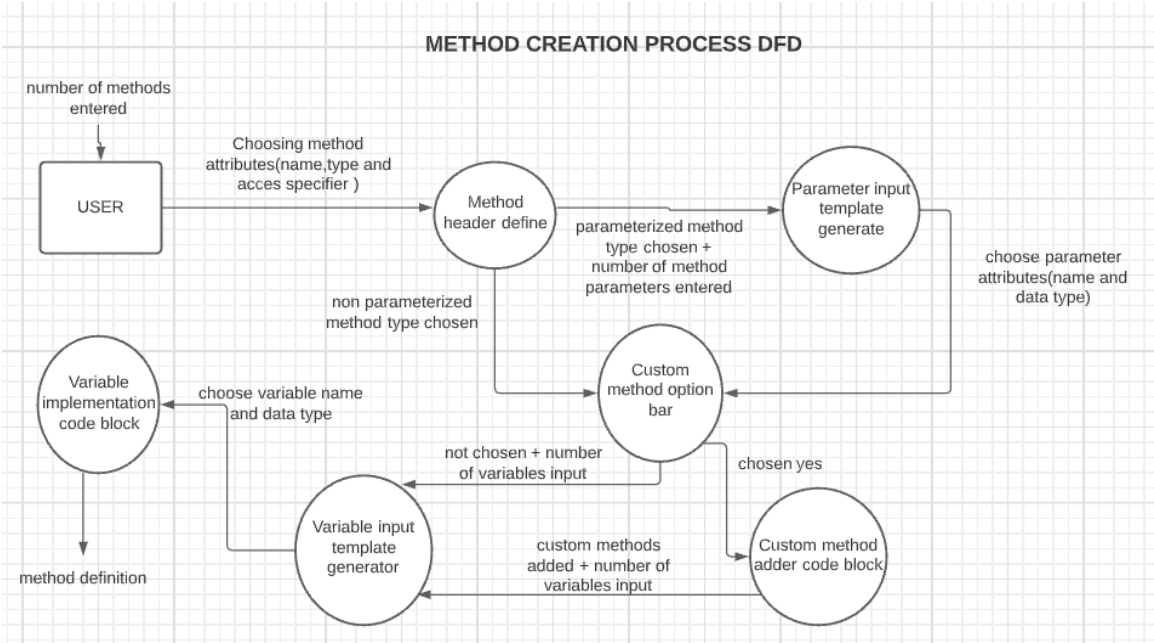


Fig: Data Flow Diagram of defining main class

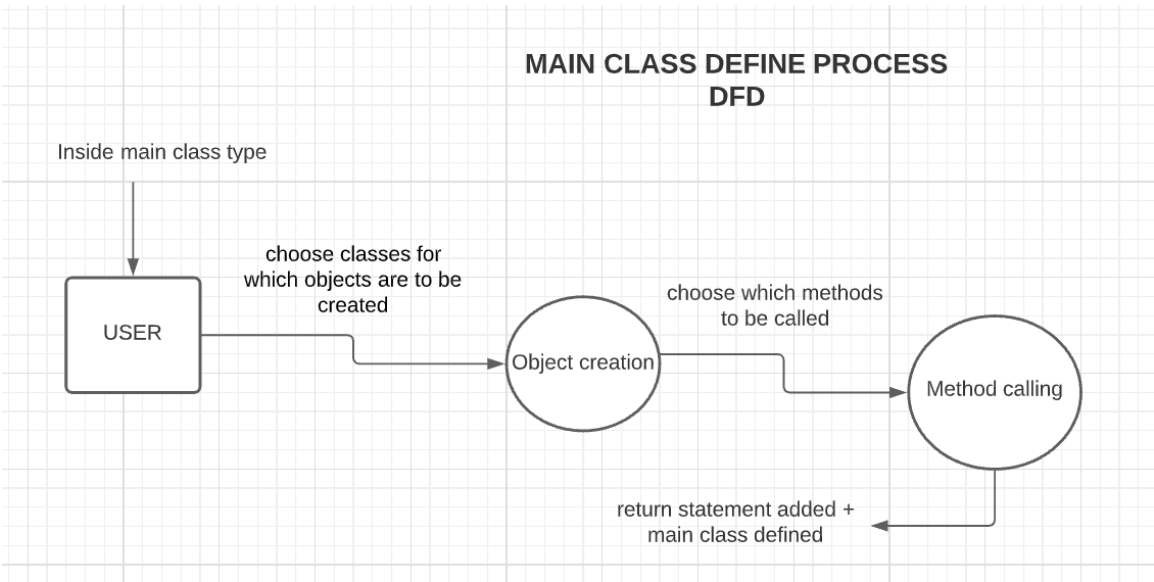


Fig : Data Flow Diagram of custom method selection

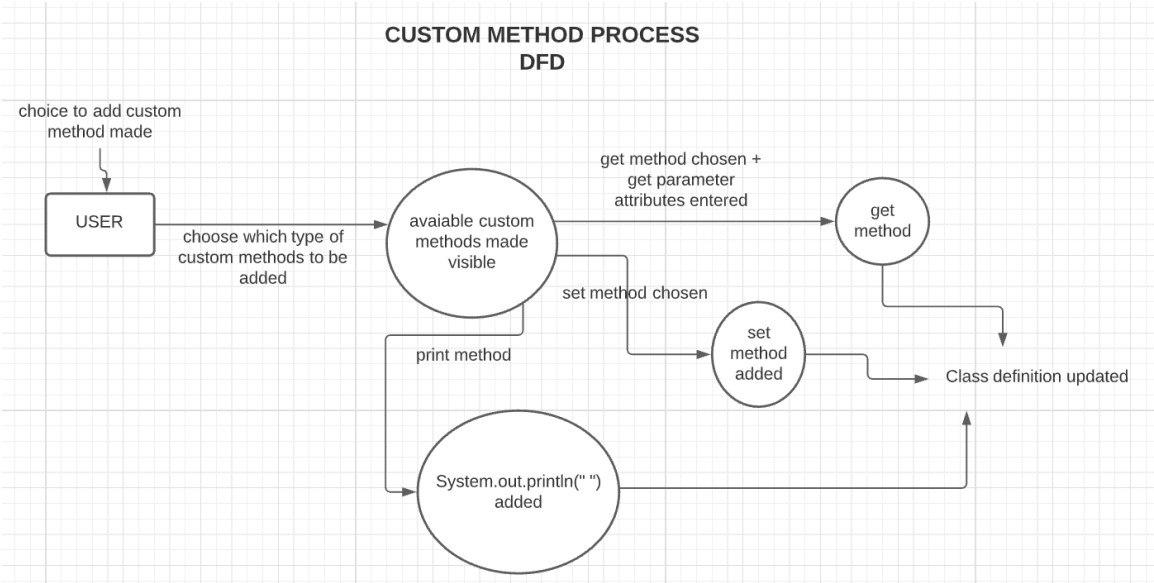


Fig: Data Flow Diagram of code generation

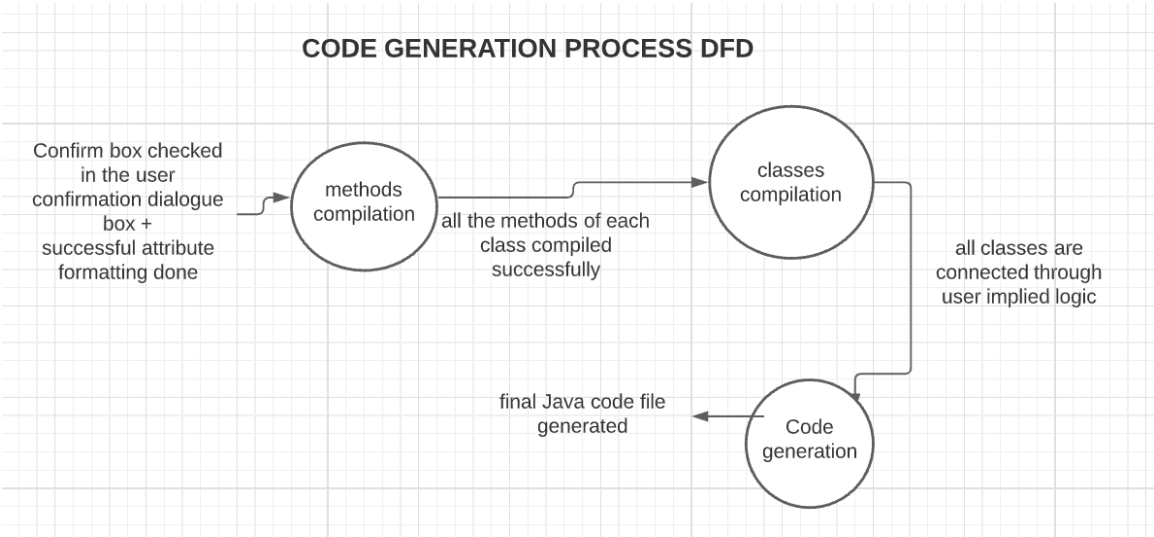
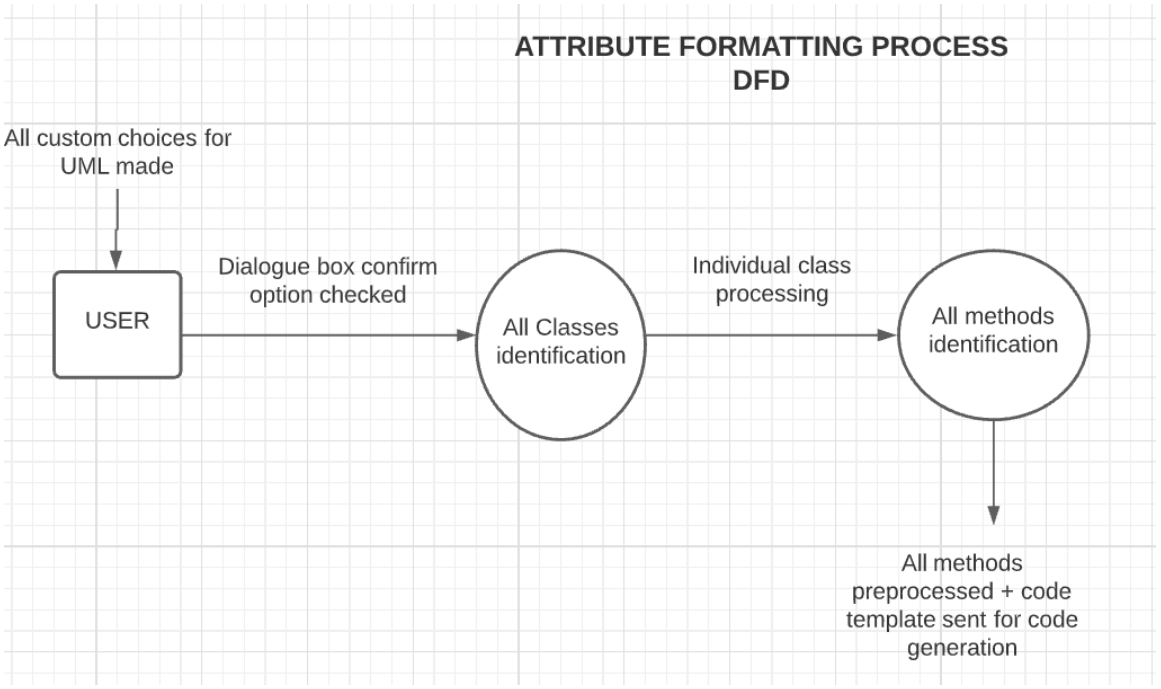
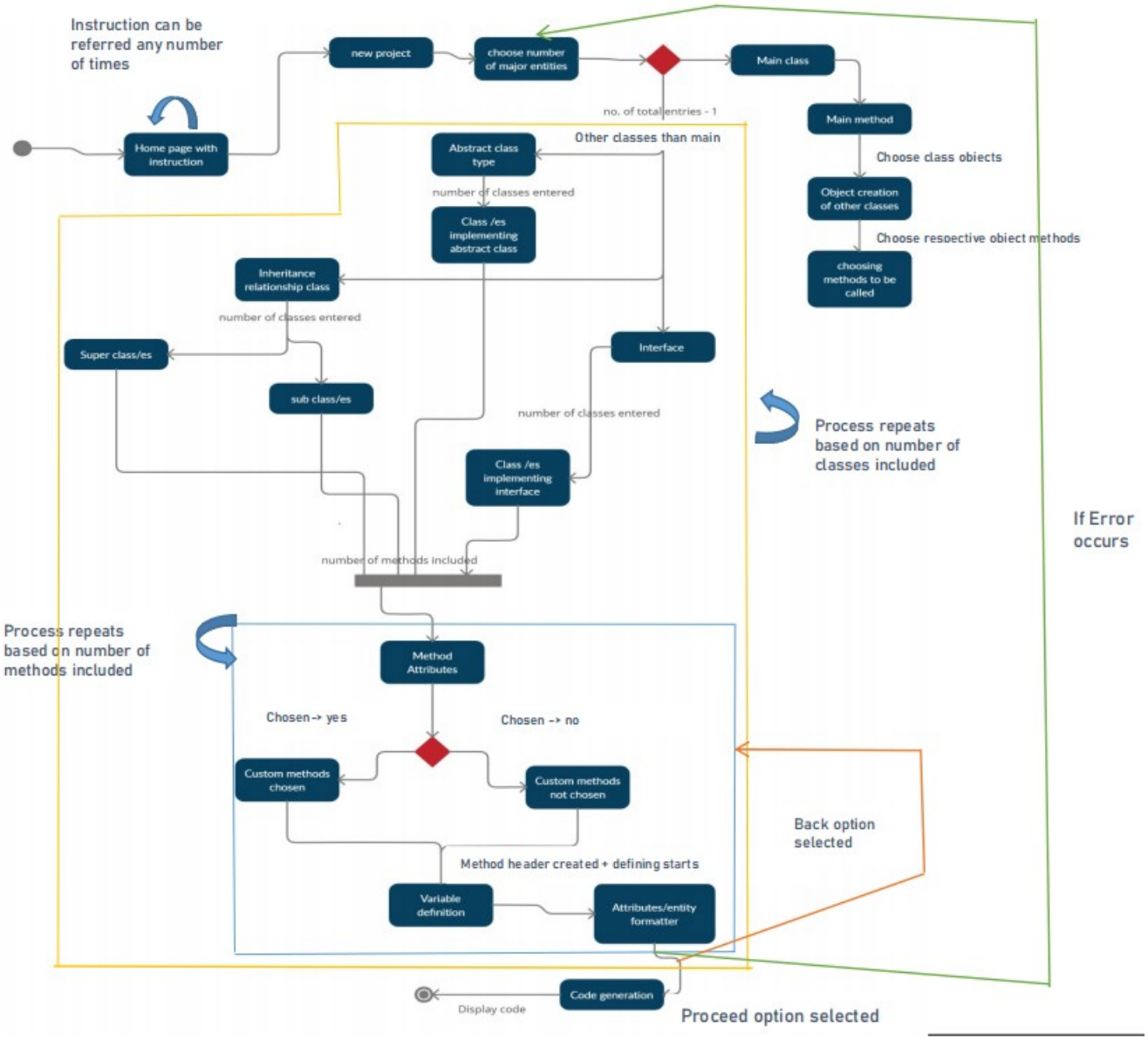


Fig: Data Flow Diagram of attribute formatting


ATTRIBUTE FORMATTING PROCESS DFD
All custom choices for UML made
USER
Dialogue box confirm option checked
All Classes identification
Individual class processing
All methods identification
All methods preprocessed + code template sent for code generation

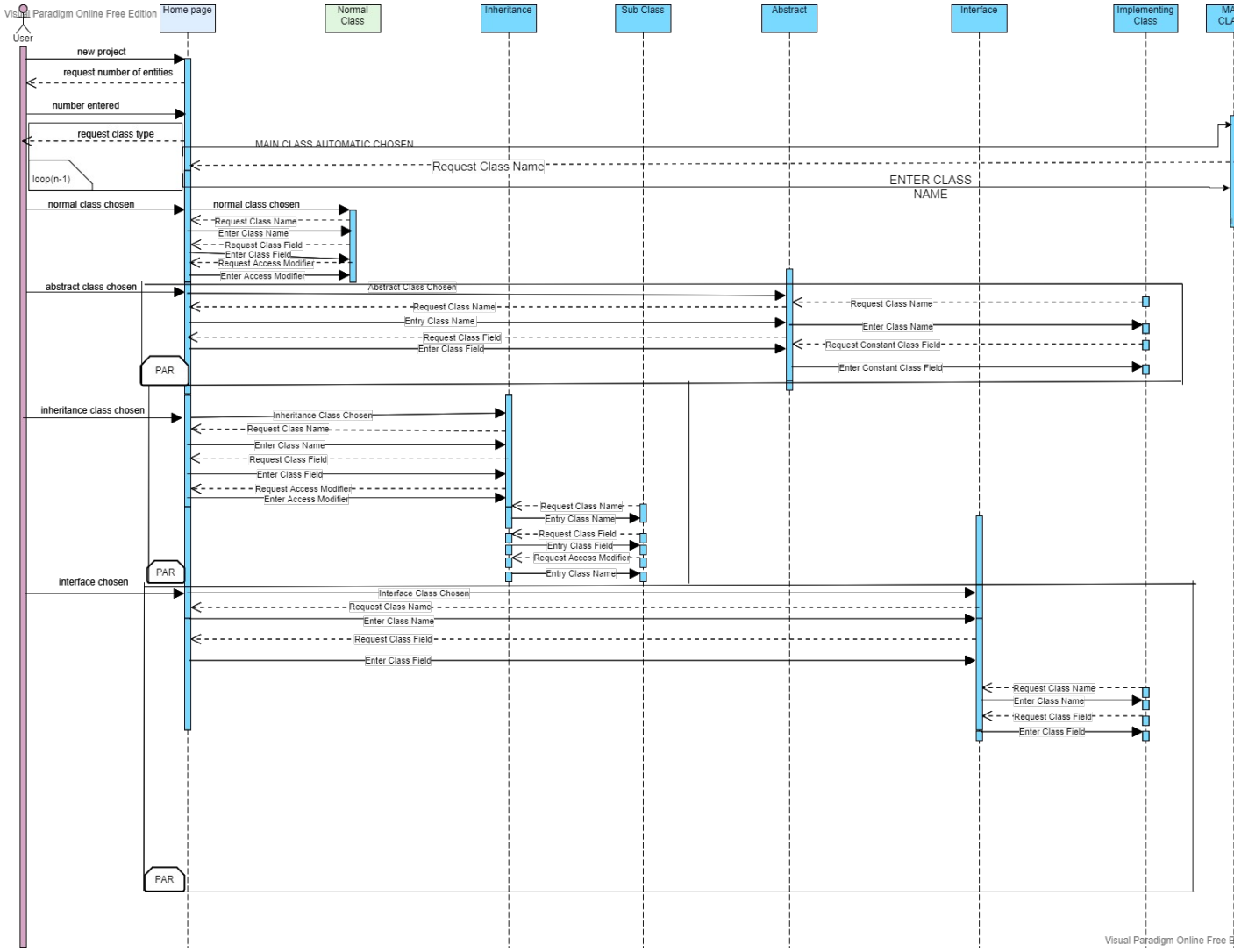
State Transition Diagram:



Module wise Detailed Designs

5.1.1Class module

Design of sequence and use case diagram



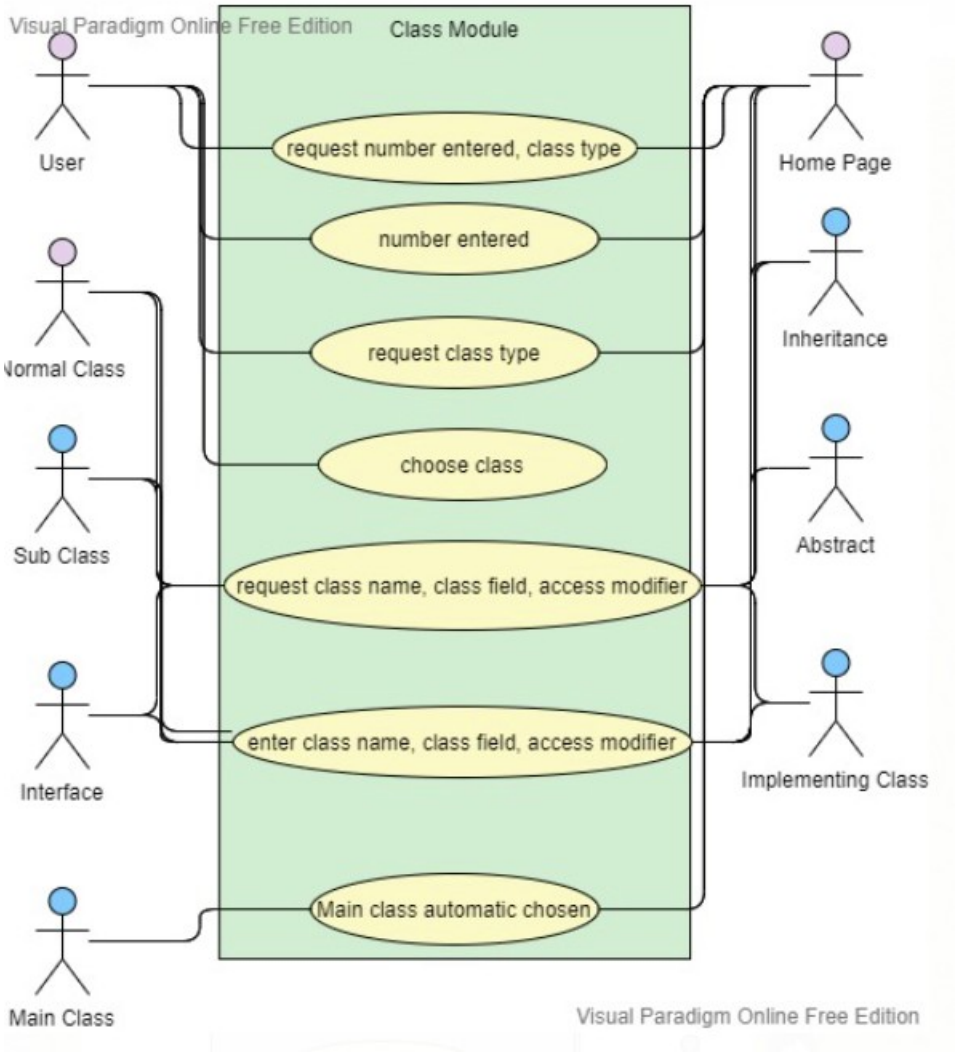


Fig: Class module sequence and use case diagram

Class module collaboration diagram

Design

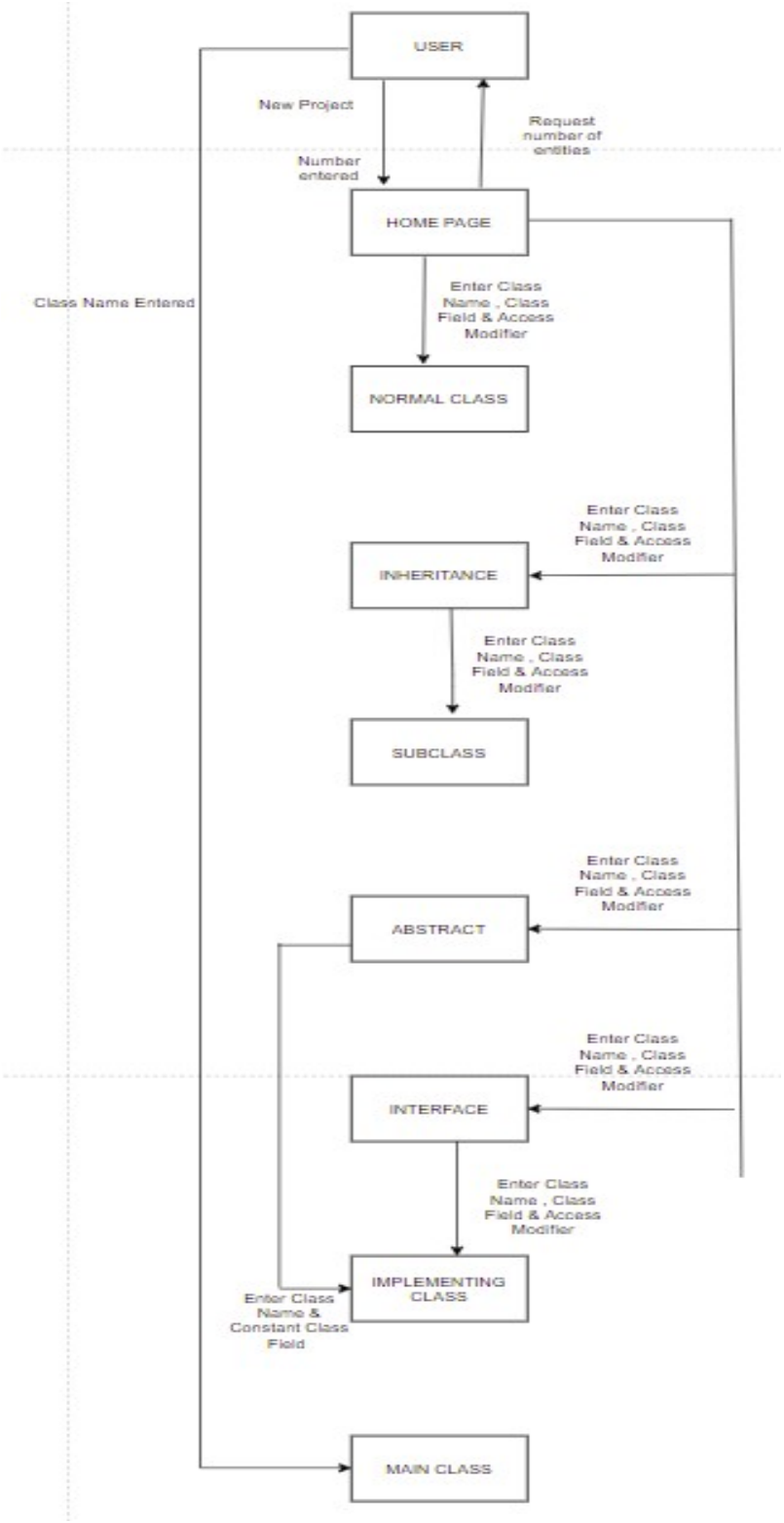


Fig: Class module collaboration diagram

Class module activity diagram

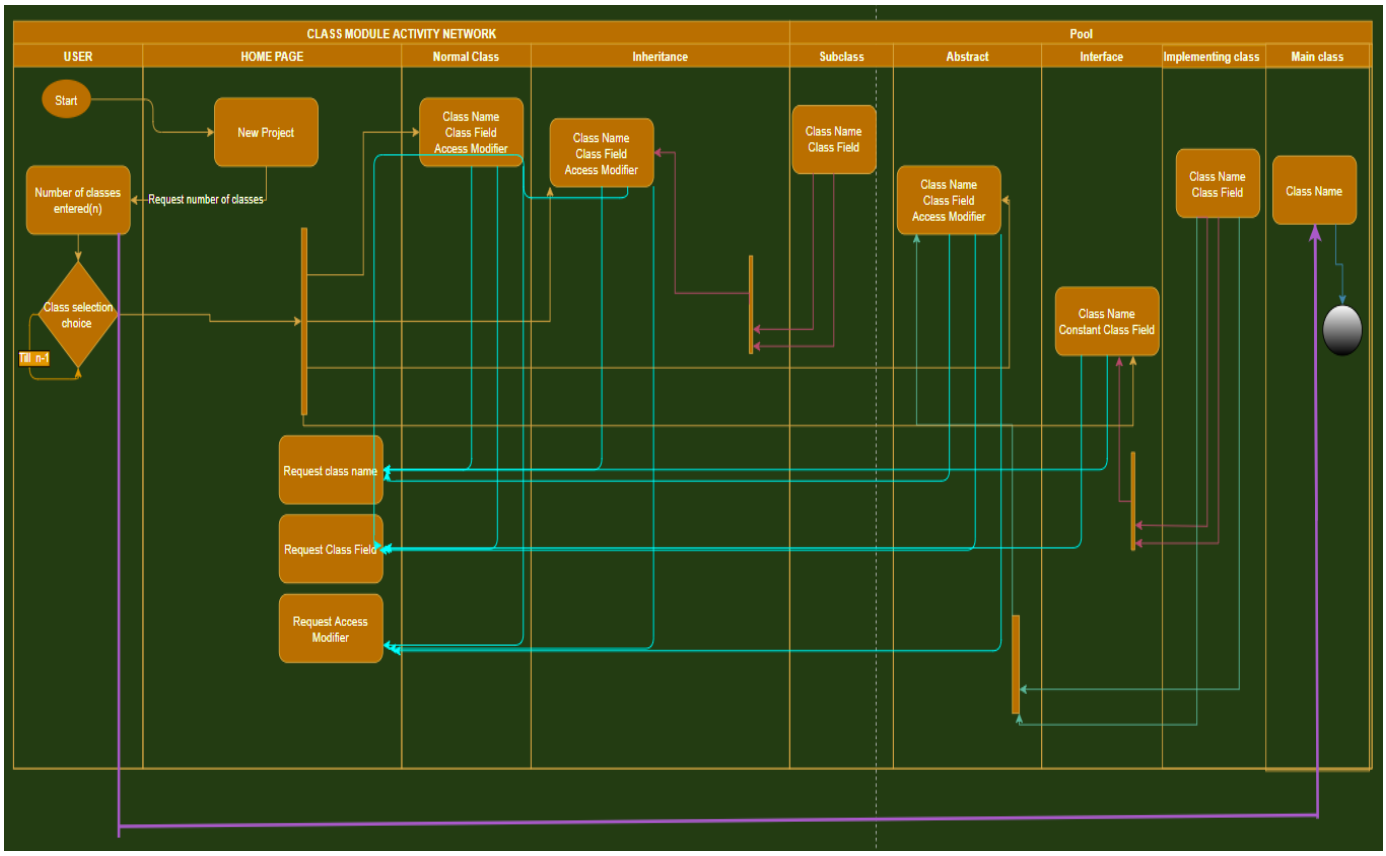
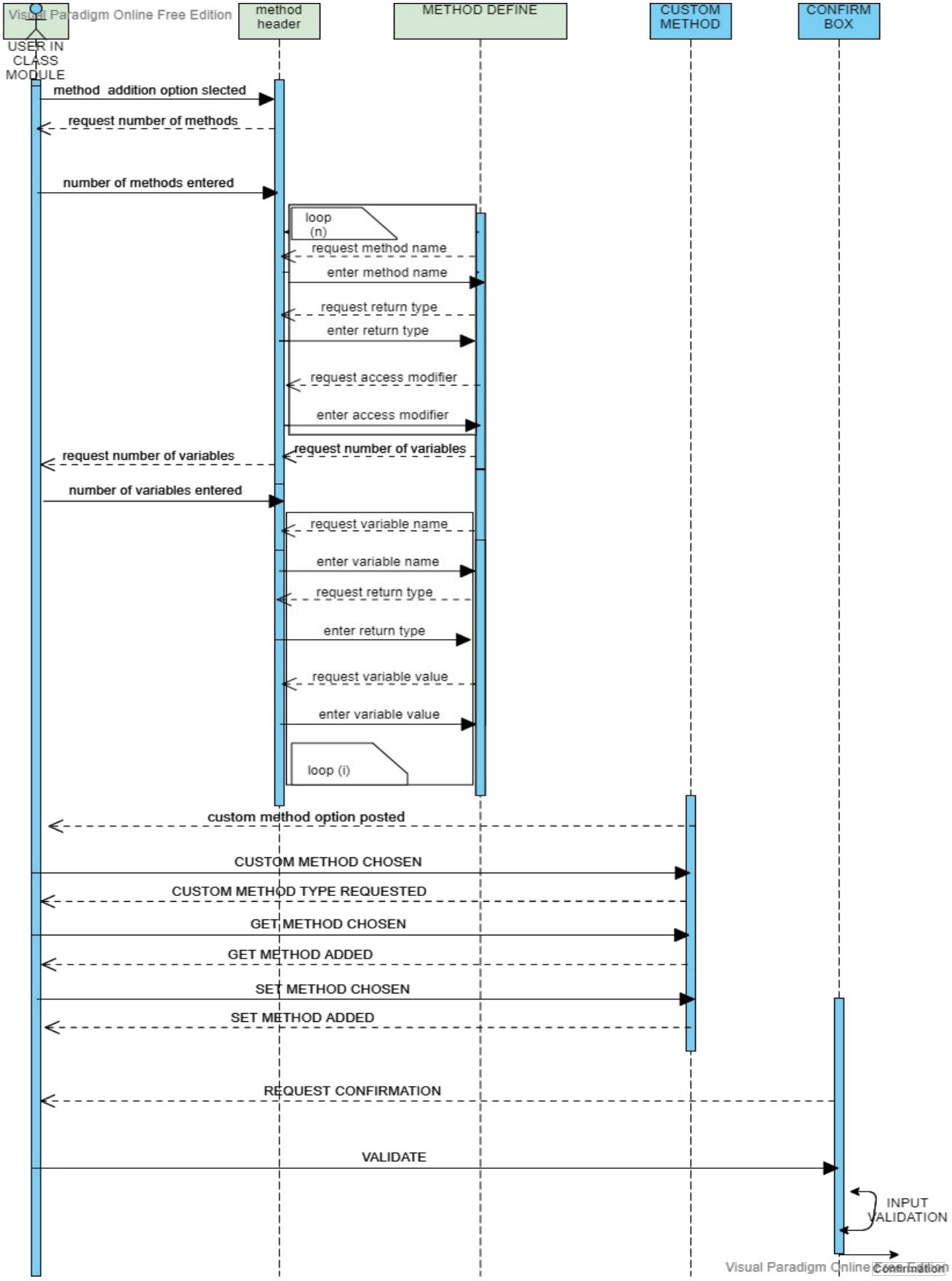


Fig: Class module activity diagram

Method module

Design of sequence and use case diagram


Visual Paradigm Online Free Edition
USER IN CLASS MODULE
method header
METHOD DEFINE
CUSTOM METHOD
CONFIRM BOX
method addition option slected
request number of methods
number of methods entered
loop (n)
request method name
enter method name
request return type
enter return type
request access modifier
enter access modifier
request number of variables
request number of variables
number of variables entered
request variable name
enter variable name
request return type
enter return type
request variable value
enter variable value
loop (i)
custom method option posted
CUSTOM METHOD CHOSEN
CUSTOM METHOD TYPE REQUESTED
GET METHOD CHOSEN
GET METHOD ADDED
SET METHOD CHOSEN
SET METHOD ADDED
REQUEST CONFIRMATION
VALIDATE
INPUT VALIDATION
Confirmation
Visual Paradigm Online Free Edition

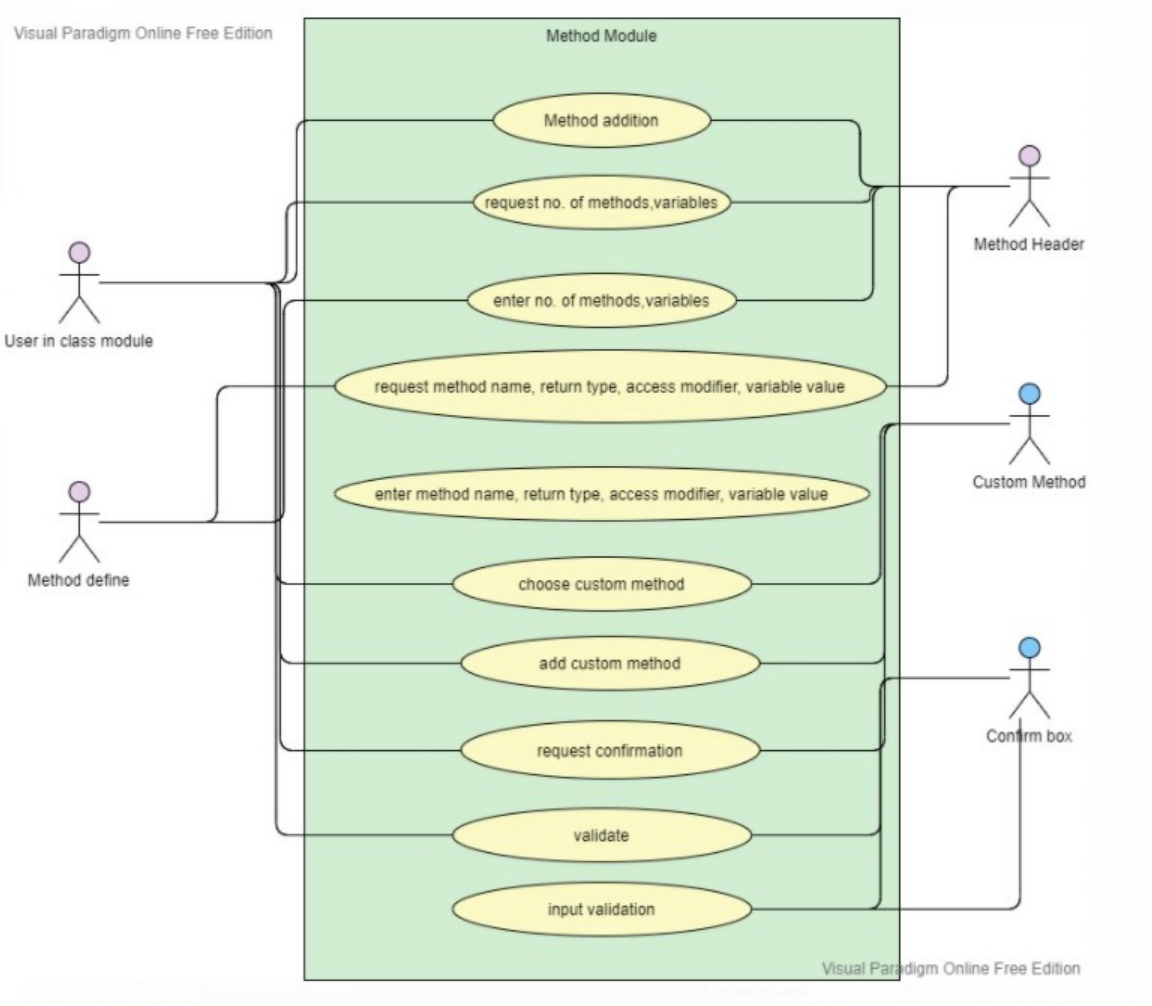


Fig: Method module sequence and use case diagram

Method module collaboration diagram

Design

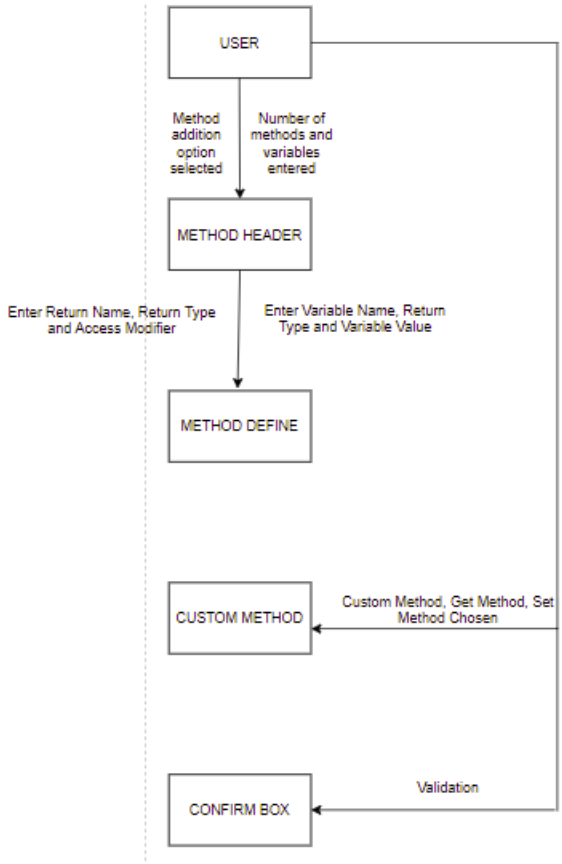


Fig: Method module collaboration diagram

Method module activity diagram

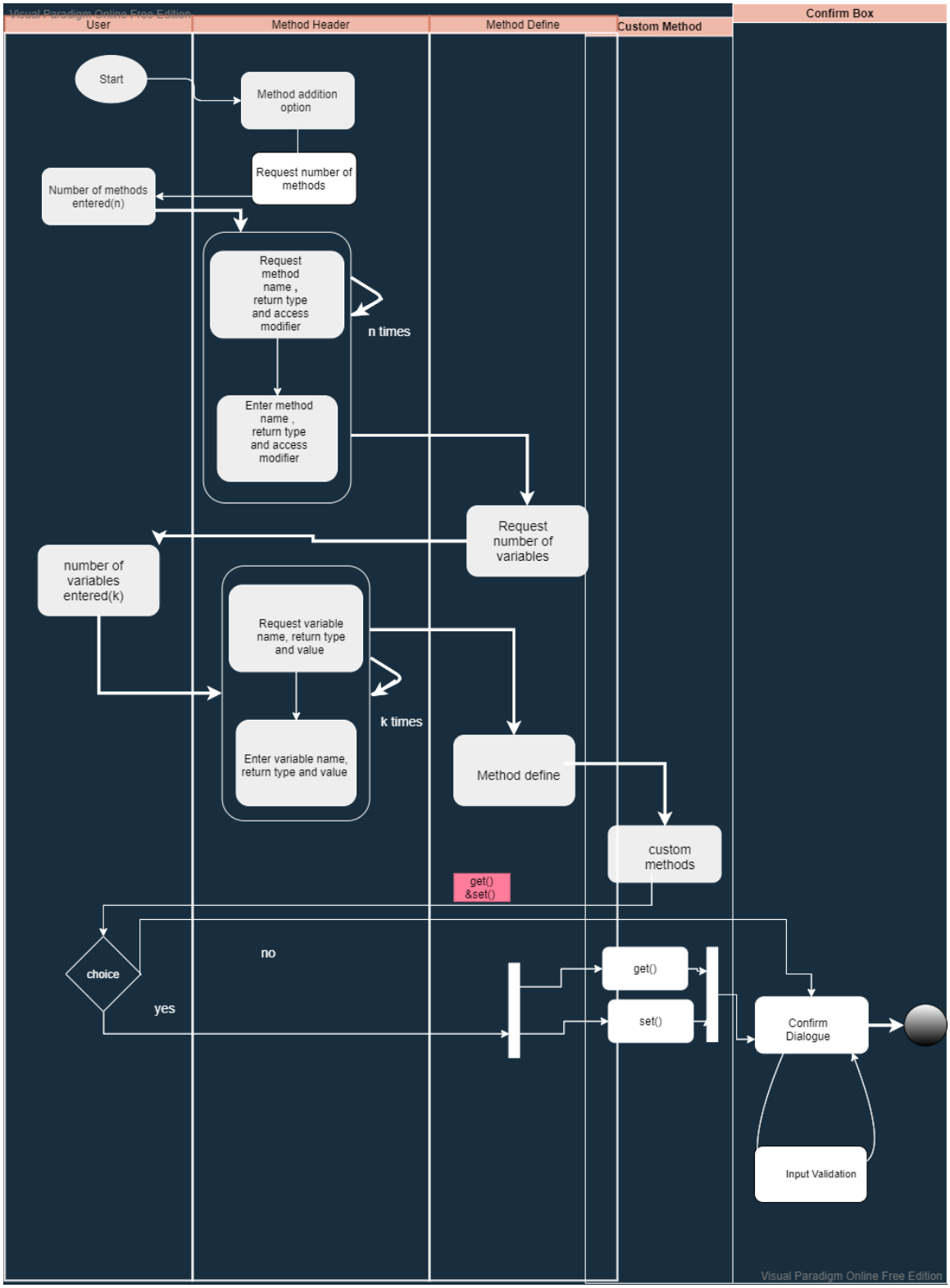
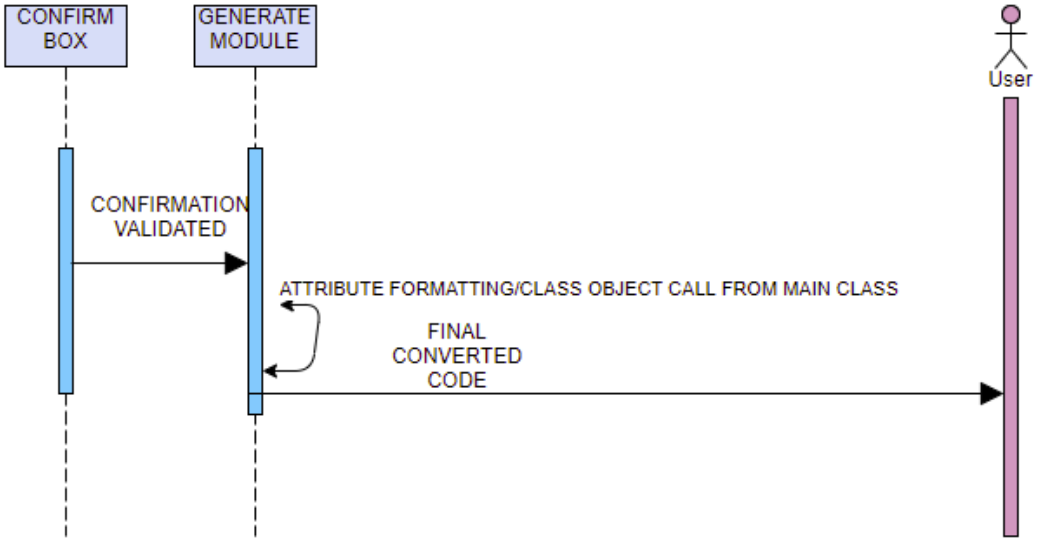


Fig: Method module activity diagram

Generate module

Design of sequence and use case diagram



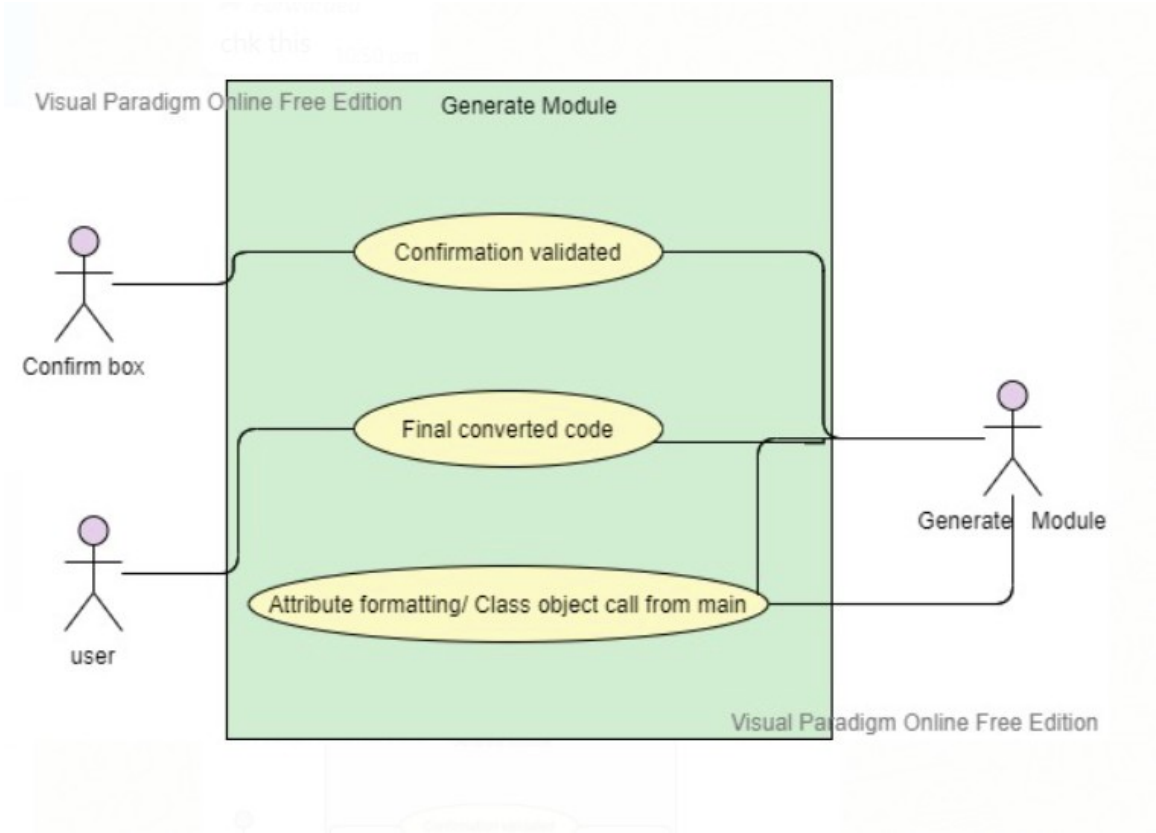


Fig: Generate module sequence and use case diagram

.

Generate module collaboration diagram

Design

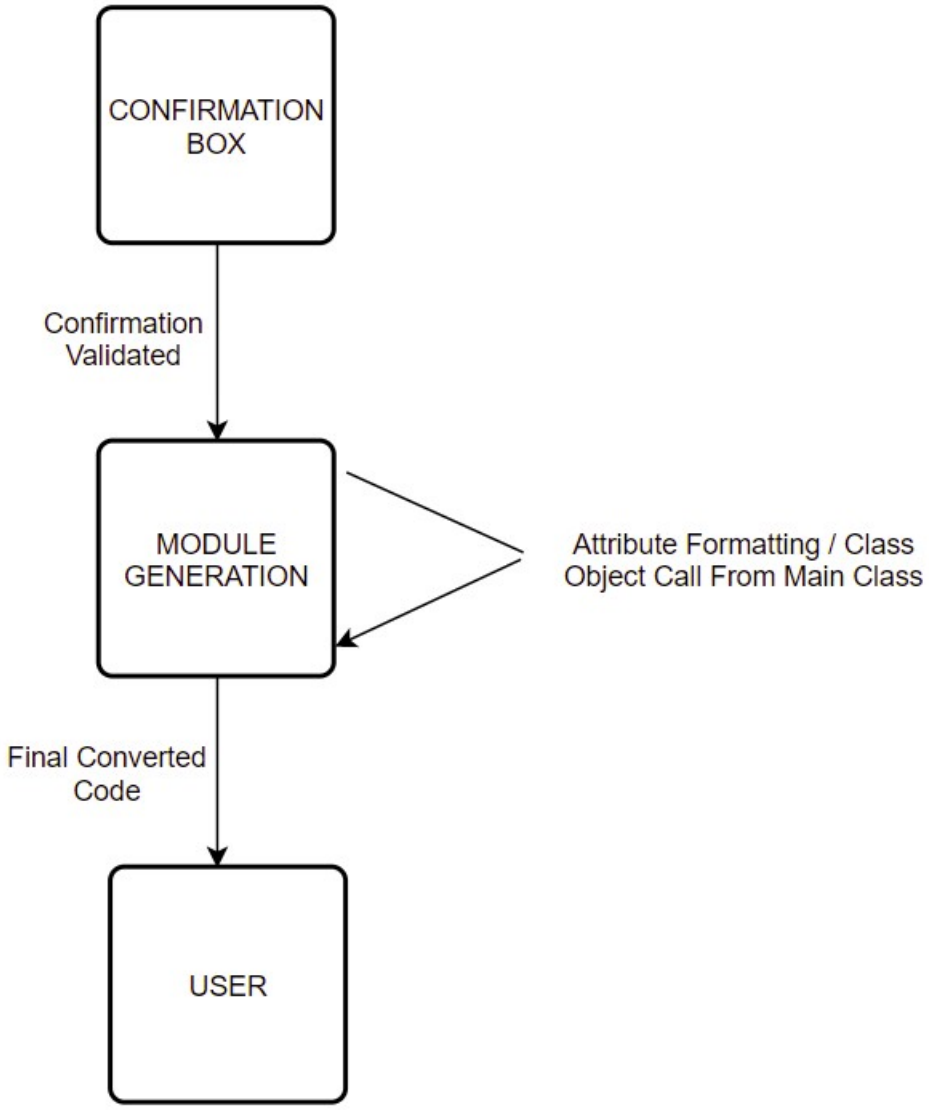


Fig: Generate module collaboration diagram

Generate module activity diagram

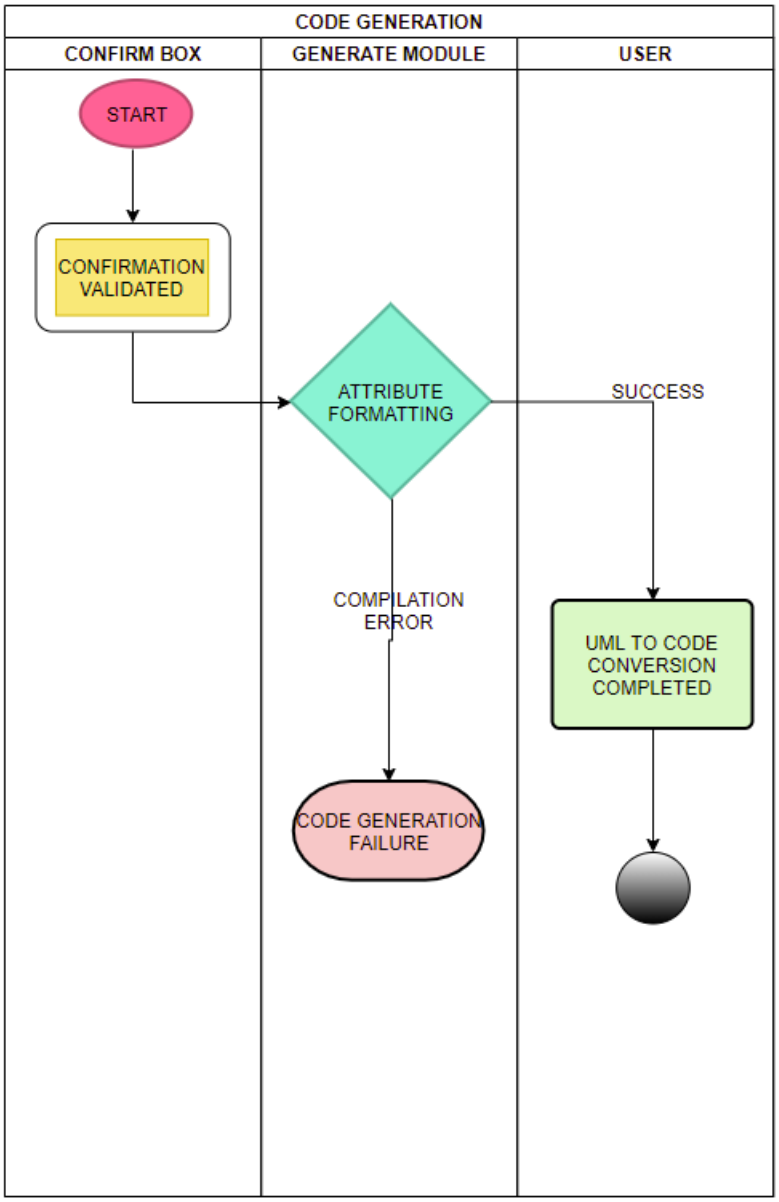
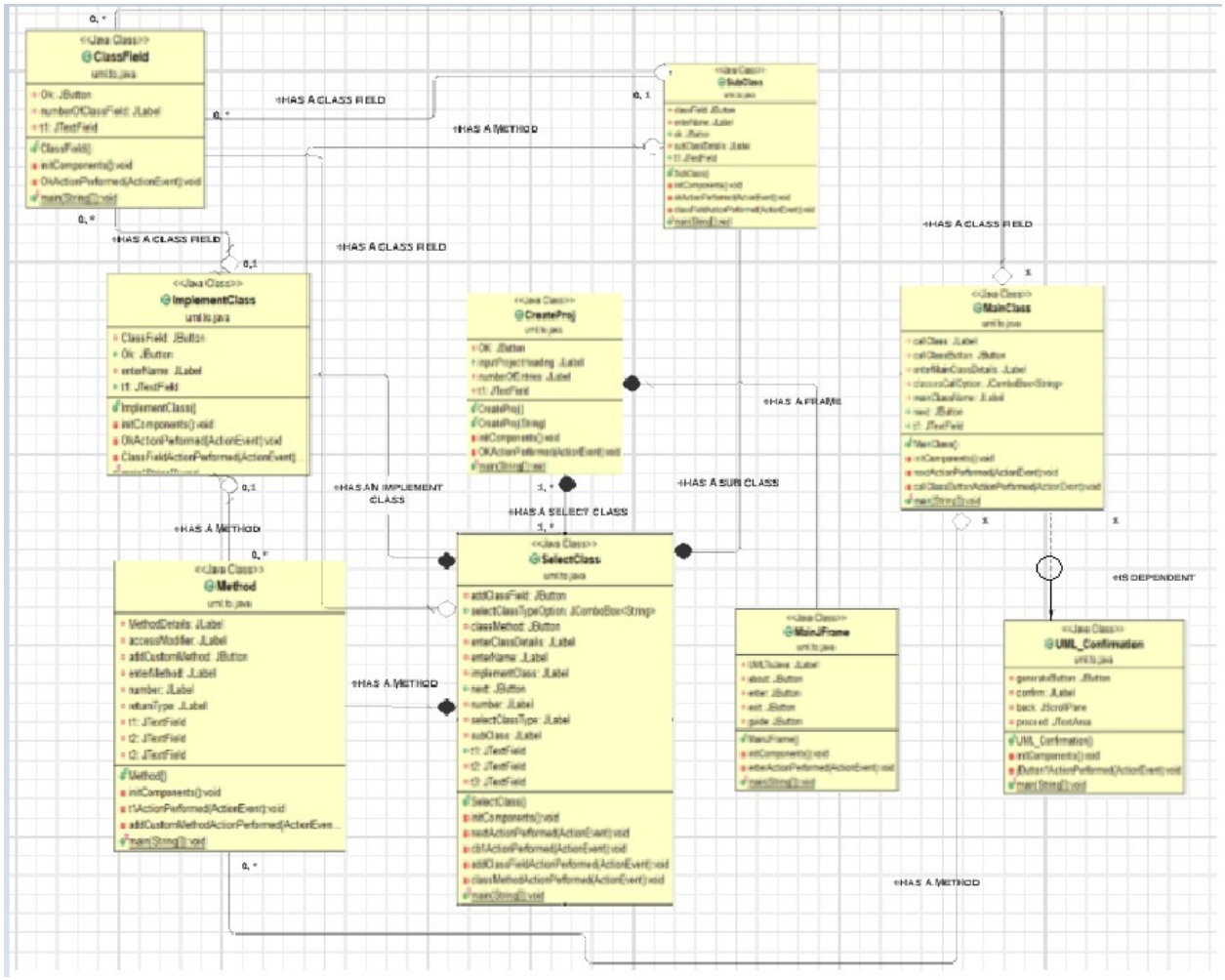


Fig: Generate module activity diagram

UML 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 :

```java
package uml.to.java;
import javax.swing.JOptionPane;

/**
 *
 * @author User
 */
public class MainJFrame extends javax.swing.JFrame {

    /**
     * Creates new form MainJFrame
     */
    public MainJFrame() {
        initComponents();
    }

    /**
     * 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() {

        UMLToJava = new javax.swing.JLabel();
        guide = new javax.swing.JButton();
        guide.setFont(new Font("Century Schoolbook", Font.PLAIN, 17));
        about = new javax.swing.JButton();
        about.setFont(new Font("Century Schoolbook", Font.PLAIN, 17));
        enter = new javax.swing.JButton();
        enter.setFont(new Font("Century Schoolbook", Font.PLAIN, 17));
        exit = new javax.swing.JButton();
        exit.setFont(new Font("Century Schoolbook", Font.PLAIN, 17));
        exit.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent arg0) {
                dispose();

            }
        });
```

ClassModule, Home Page

```java
/**
 * 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

```java
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        });
```

Entering number of class fields

```java
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                jTextField2ActionPerformed(evt);
```

Entering Field details

```java

    /**
     * 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() {

        jLabel1 = new javax.swing.JLabel();
        jLabel2 = new javax.swing.JLabel();
        t1 = new javax.swing.JTextField();
        jButton1 = new javax.swing.JButton();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

        jLabel1.setFont(new java.awt.Font("Tahoma", 1, 18)); // NOI18N
        jLabel1.setText("Number of Methods");

        jLabel2.setText("Number:");

        jButton1.setText("Confirm");
        jButton1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                 try{int l = Integer.parseInt(t1.getText());
                 String[] arr = new String[l];
                 int[] type = new int[l];
                 dispose();
                 for (int i = 0; i < l; i++) {
                 Method frm = new Method(l1.getText());
                 frm.setVisible(true);


            }}
```

Entering number of methods

```java
            }
        });

        accessModifier.setText("Acces Modifier");

        JButton btnDone = new JButton("Done");
        btnDone.setFont(new Font("Century Schoolbook", Font.PLAIN, 16));
        btnDone.setVisible(false);

        btnDone.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                String d= t3.getText()+" "+ t2.getText()+ " "+ t1.getText()+ "() {";
                String l = printt+" }";
                System.out.println(d);
                System.out.println(l);
                dispose();


            }
        });
        JLabel lblNewLabel_1 = new JLabel("[Blank]");


        JButton btnAttributesConfirm = new JButton("Attributes confirm");
        btnAttributesConfirm.setFont(new Font("Century Schoolbook", Font.PLAIN, 16));
        btnAttributesConfirm.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent arg0) {
                addCustomMethod.setVisible(true);
                t4.setVisible(true);
                lblNewLabel.setVisible(true);
                btnDone.setVisible(true);
                lblNewLabel_1.setText("Method Header defined!");
                btnAttributesConfirm.setVisible(false);

```

Entering methods details(declaration)

```java

    public ParentInterface(String n ){
        initComponents();
        l1.setText(n+"");
    }

    /*public childInherit(String n,String i ){
        initComponents();
       // l2.setText(n+"");
        l1.setText(i+".1");
    }*/

    public void getCname () {


    }

    /**
     * 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));
        next = new javax.swing.JButton();
        next.setFont(new Font("Century Schoolbook", Font.PLAIN, 15));
        next.setVisible(false);
        t1 = new javax.swing.JTextField();

        l1 = new javax.swing.JLabel();
        l1.setFont(new Font("Century Schoolbook", Font.PLAIN, 16));
        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));
```

Entering Interface details

```java
 */
public class interfaceField extends javax.swing.JFrame {

    /**
     * Creates new form ClassField
     */
    public interfaceField() {
        initComponents();
    }

    public interfaceField(String a ){
        initComponents();
        String g = "fields for class no. ";
        String j = g+a + SelectClass.jk;
        l1.setText(j);
    }

    /**
     * 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() {

        jLabel1 = new javax.swing.JLabel();
        numberOfClassField = new javax.swing.JLabel();
        numberOfClassField.setFont(new Font("Century Schoolbook", Font.PLAIN, 16));
        t1 = new javax.swing.JTextField();
        Ok = new javax.swing.JButton();
        Ok.setFont(new Font("Century Schoolbook", Font.PLAIN, 16));
        l1 = new javax.swing.JLabel();
        l1.setText("[Blank]");

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

        jLabel1.setFont(new Font("Century Schoolbook", Font.BOLD, 18)); // NOI18N
```

Setting Interface fields

```java
public class interfaceMethodNumber extends javax.swing.JFrame {

    //public int methodFlag;

    /**
     * Creates new form ClassField
     */
    public interfaceMethodNumber() {
        initComponents();
    }

    public interfaceMethodNumber(String s) {
         initComponents();
        String g = "no. of methods for class no: ";
        String j = g+s + SelectClass.jk;

        l1.setText(s);

    }

    /**
     * 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() {

        jLabel1 = new javax.swing.JLabel();
        jLabel2 = new javax.swing.JLabel();
        jLabel2.setFont(new Font("Century Schoolbook", Font.PLAIN, 16));
        t1 = new javax.swing.JTextField();
        jButton1 = new javax.swing.JButton();
        jButton1.setFont(new Font("Century Schoolbook", Font.PLAIN, 16));

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
```

Setting Interface Methods

```java
        classMethod.setText("Class Method");
        classMethod.setVisible(false);
        classMethod.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                classMethodActionPerformed(evt);
            }
        });

        lblInheritanceChildClass = new JLabel("INTERFACE IMPLEMENT");
        lblInheritanceChildClass.setFont(new Font("Century Schoolbook", Font.BOLD, 17));

        btnNewButton = new JButton("Confirm Name");
        btnNewButton.setFont(new Font("Century Schoolbook", Font.PLAIN, 16));
        btnNewButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {

                next.setVisible(true);
                addClassField.setVisible(true);
                classMethod.setVisible(true);

                lblNewLabel.setText("IMPLEMENT CLASS HEADER DEFINED ");
                getClassName();
                enterName.setVisible(false);
                t1.setVisible(false);
                btnNewButton.setVisible(false);
            }
        });

        lblNewLabel = new JLabel("[Blank]");
        lblNewLabel.setFont(new Font("Century Schoolbook", Font.PLAIN, 16));

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        layout.setHorizontalGroup(
            layout.createParallelGroup(Alignment.LEADING)
                .addGroup(layout.createSequentialGroup()
                    .addContainerGap()
                    .addComponent(l1, GroupLayout.DEFAULT_SIZE, 595, Short.MAX_VALUE))
                .addGroup(layout.createSequentialGroup()
                    .addGap(157)
```

Interface Child implementation

```java
    /*public childInherit(String n,String i ){
        initComponents();
       // l2.setText(n+"");
        l1.setText(i+".1");
    }*/

    public void getCname () {


    }

    /**
     * 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));
        next = new javax.swing.JButton();
        next.setFont(new Font("Century Schoolbook", Font.PLAIN, 15));
        next.setVisible(false);
        t1 = new javax.swing.JTextField();
        l1 = new javax.swing.JLabel();
        l1.setFont(new Font("Century Schoolbook", Font.PLAIN, 16));
        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));
```

Parent Inheritance Implementation

```java
public class childInherit extends javax.swing.JFrame {
    static String jk = "";

    /**
     * Creates new form SelectClass
     */
    public childInherit() {
        initComponents();
    }

    public childInherit(String n ){
        initComponents();
        l1.setText(n+"");
    }

    /*public childInherit(String n,String i ){
        initComponents();
       // l2.setText(n+"");
        l1.setText(i+".1");
    }*/



    /**
     * 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));
        next = new javax.swing.JButton();
        next.setFont(new Font("Century Schoolbook", Font.PLAIN, 15));
        next.setVisible(false);
```

Child Inheritance Implementation

```java
public class parentAbstract extends javax.swing.JFrame {
    static String jk = "";
    static String parentinh = "";

    /**
     * Creates new form SelectClass
     */
    public parentAbstract() {
        initComponents();
    }

    public parentAbstract(String n ){
        initComponents();
        l1.setText(n+"");
    }

    /*public childInherit(String n,String i ){
        initComponents();
       // l2.setText(n+"");
        l1.setText(i+".1");
    }*/

    public void getCname () {


    }

    /**
     * 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));
        next = new javax.swing.JButton();
        next.setFont(new Font("Century Schoolbook", Font.PLAIN, 15));
        next.setVisible(false);
```

Abstract Parent implementation

```java
import javax.swing.GroupLayout.Alignment;

public class Abstractmethodnumber extends javax.swing.JFrame {

    //public int methodFlag;

    /**
     * Creates new form ClassField
     */
    public Abstractmethodnumber() {
        initComponents();
    }

    public Abstractmethodnumber(String s) {
         initComponents();
        //String g = "no. of methods for class no: ";
        //String j = g+s + SelectClass.jk;

        l1.setText(s);

    }

    /**
     * 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() {

        jLabel1 = new javax.swing.JLabel();
        jLabel2 = new javax.swing.JLabel();
        jLabel2.setFont(new Font("Century Schoolbook", Font.PLAIN, 16));
        t1 = new javax.swing.JTextField();
        jButton1 = new javax.swing.JButton();
        jButton1.setFont(new Font("Century Schoolbook", Font.PLAIN, 16));

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
```

No. Of Abstract Methods

```java
public class abstractMethod extends javax.swing.JFrame {
static String printt ="";
    /**
     * Creates new form Method
     */
    public abstractMethod() {
        initComponents();
    }

    public abstractMethod(String n ){
        initComponents();
        //l1.setText(n+".1");

        String h = "method details of class ";
        String g = h+n ;

        l1.setText(g);
    }

    /**
     * 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() {

        MethodDetails = new javax.swing.JLabel();
        l1 = new javax.swing.JLabel();
        enterMethod = new javax.swing.JLabel();
        enterMethod.setFont(new Font("Century Schoolbook", Font.PLAIN, 17));
        t1 = new javax.swing.JTextField();
        returnType = new javax.swing.JLabel();
        returnType.setFont(new Font("Century Schoolbook", Font.PLAIN, 17));
        t2 = new javax.swing.JTextField();
```

Abstract Method details

```java
public class childAbstract extends javax.swing.JFrame {
    static String jk = "";

    /**
     * Creates new form SelectClass
     */
    public childAbstract() {
        initComponents();
    }

    public childAbstract(String n ){
        initComponents();
        l1.setText(n+"");
    }

    /*public childInherit(String n,String i ){
        initComponents();
       // l2.setText(n+"");
        l1.setText(i+".1");
    }*/

    public void getCname () {


    }

    /**
     * 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));
```

Abstract child

4.3 Constraints, Alternatives and Tradeoffs

The follow 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.

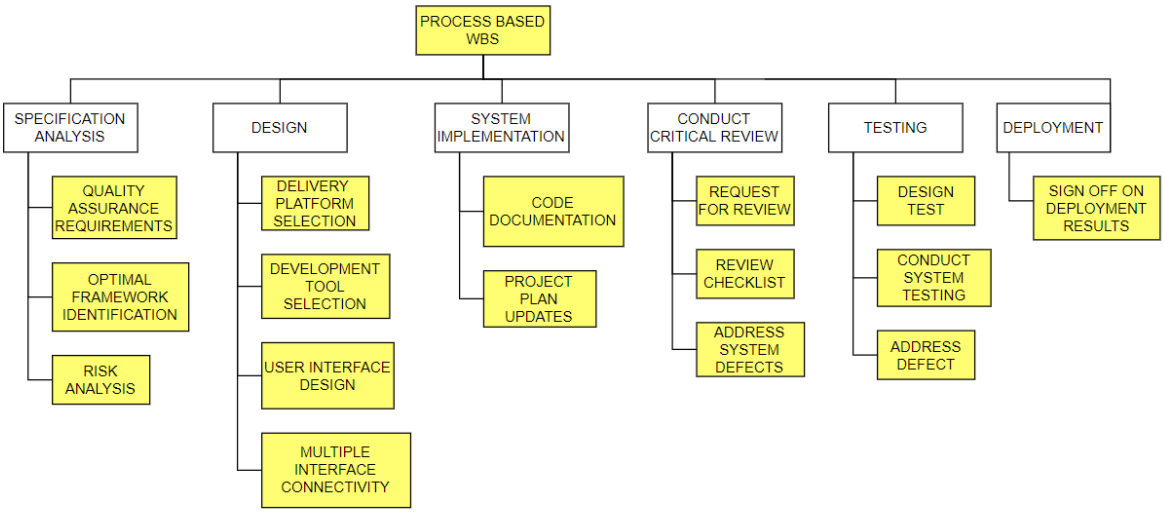
Table of Design Constraints

ID	**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*	*The system SHALL not rectify errors regarding scope of*

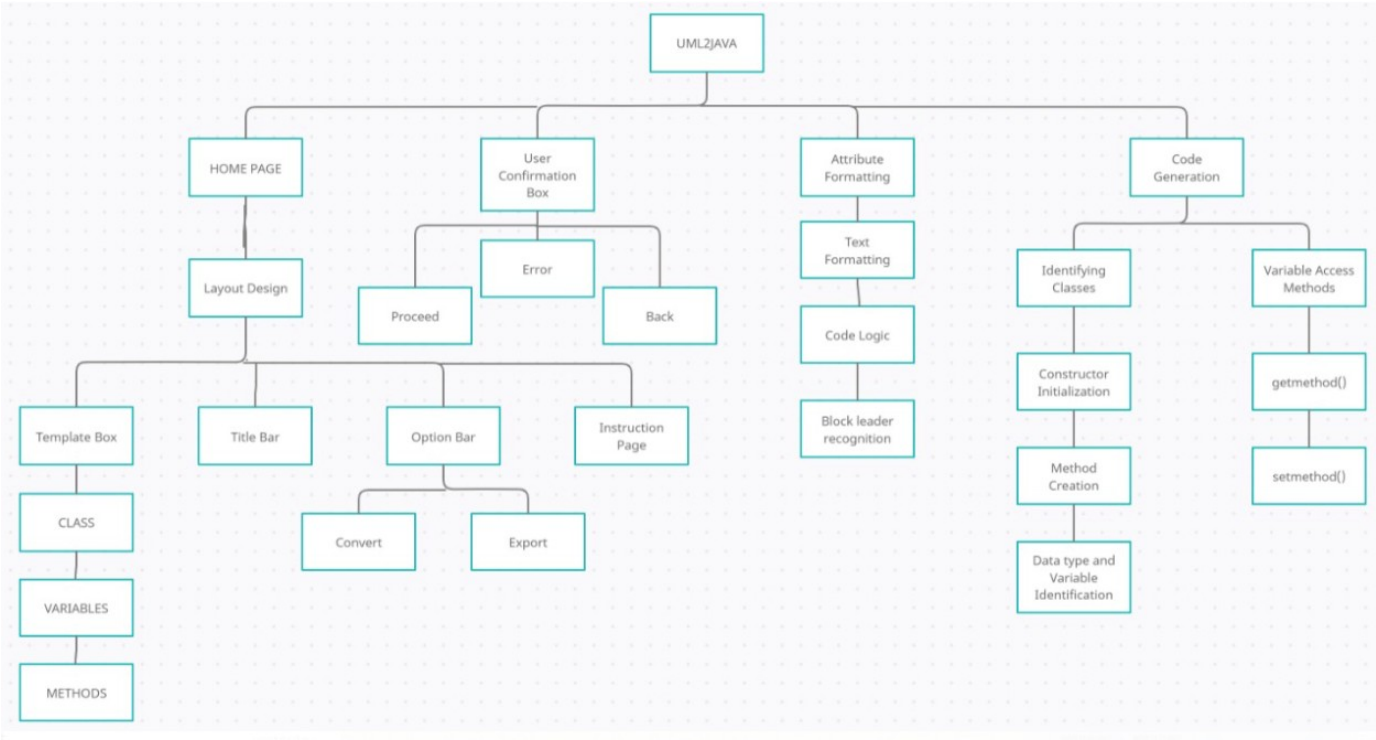
	Description Document	*variables in a method as the resultant code is based on merely user inputs.*
4	*Project Description Document*	*The system SHALL not guarantee whether correct logical relationship between classes are drawn or not as it does not check user logic.*
5	*Project Description Document*	*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*
6	*Project Description Document*	*The system SHALL not guarantee correct method overridden when it comes to Interface/abstract classes as it is user dependent*
7	*Project Description Document*	*The system SHALL not guarantee correct implementation of multiple subclasses of a single super class*

5 SCHEDULE, TASKS AND MILESTONES

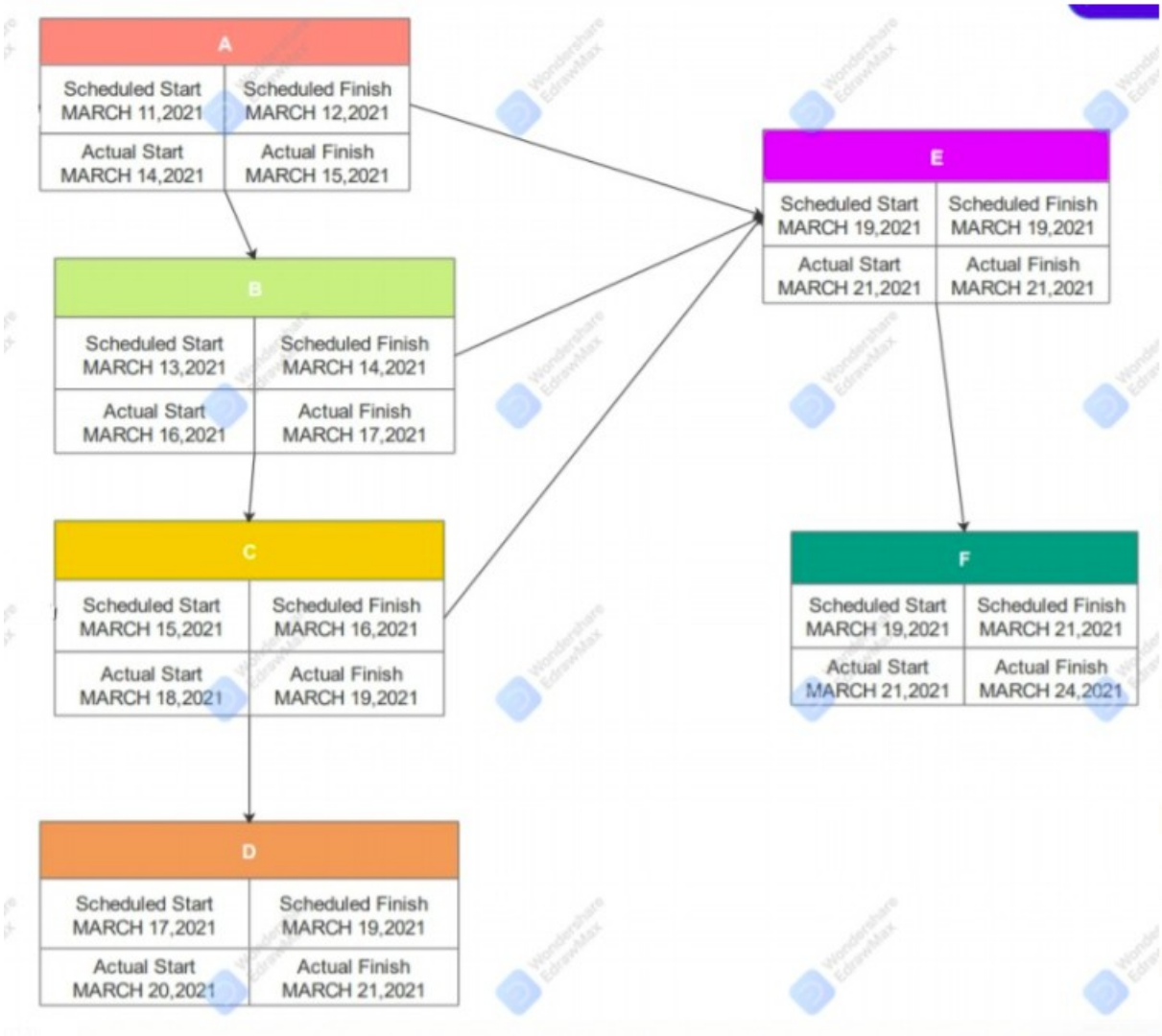
5.1 Work Breakdown Process based:



Product Based:

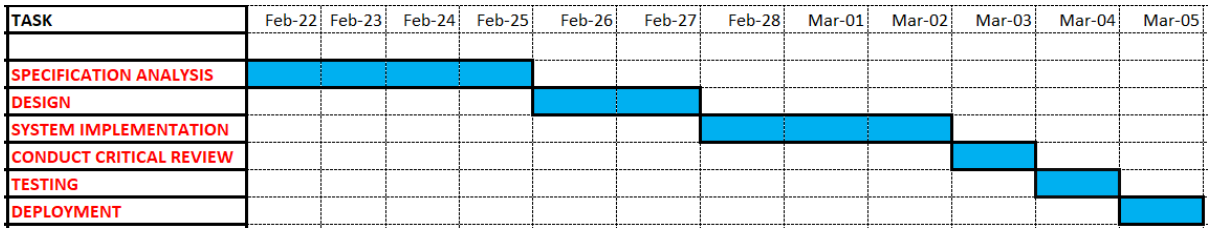


5.2 Activity Network Diagram:



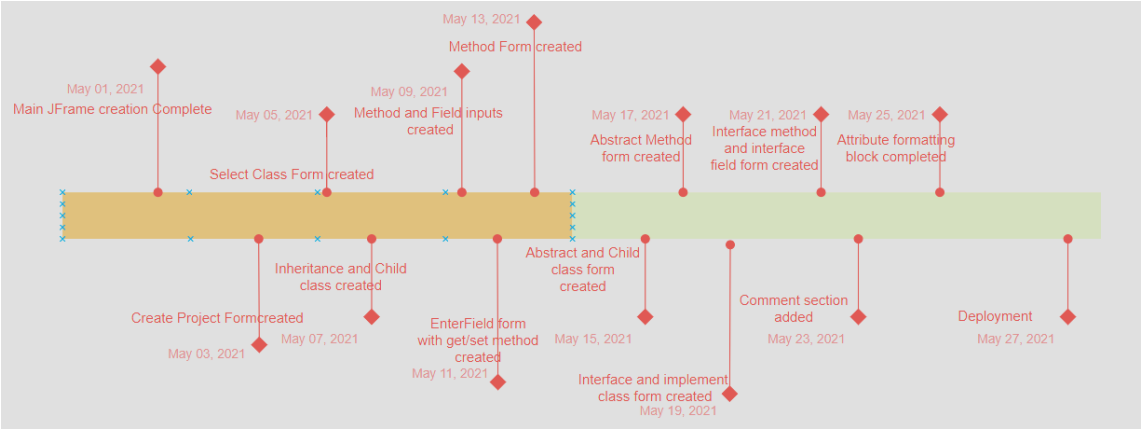
5.3 Gantt Chart

Process Based:



Product Based:

5.4 Milestone



6 PROJECT DEMONSTRATION .

a) GUI

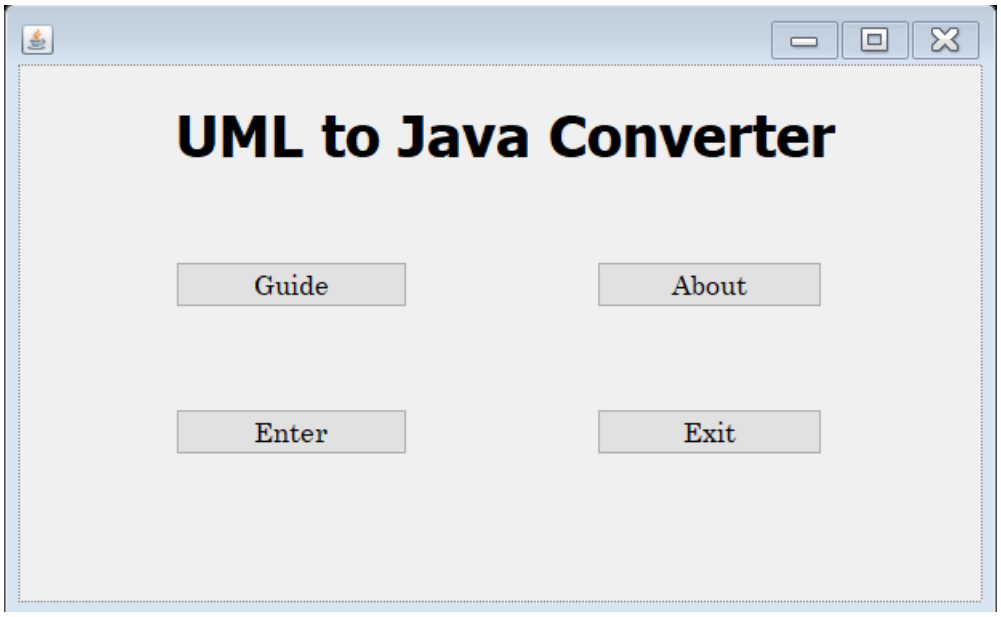


Figure 1:- Home Page

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.

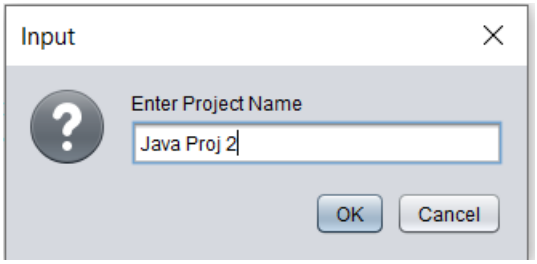


Figure 2:- Project name input box

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

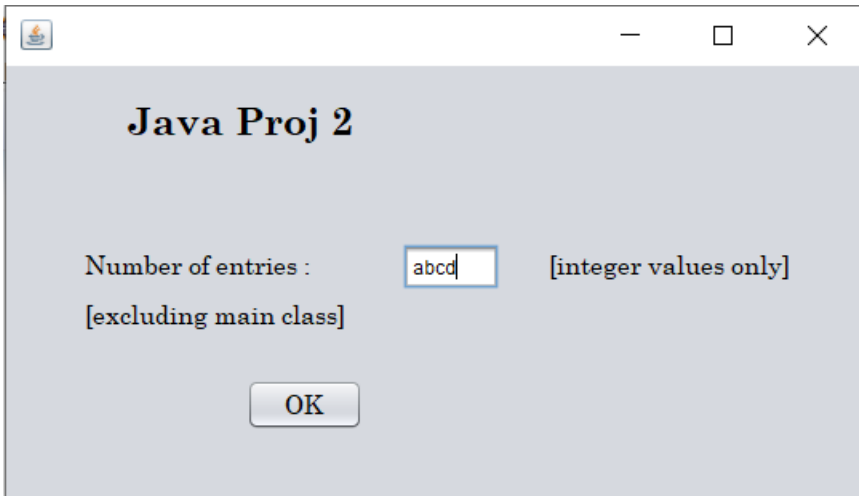


Figure 3:- Number of entries intake window with illegal entry

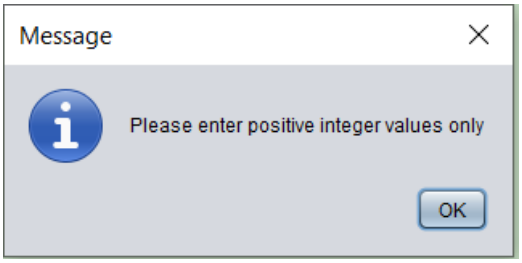


Figure 4:- Illegal entry pop up window

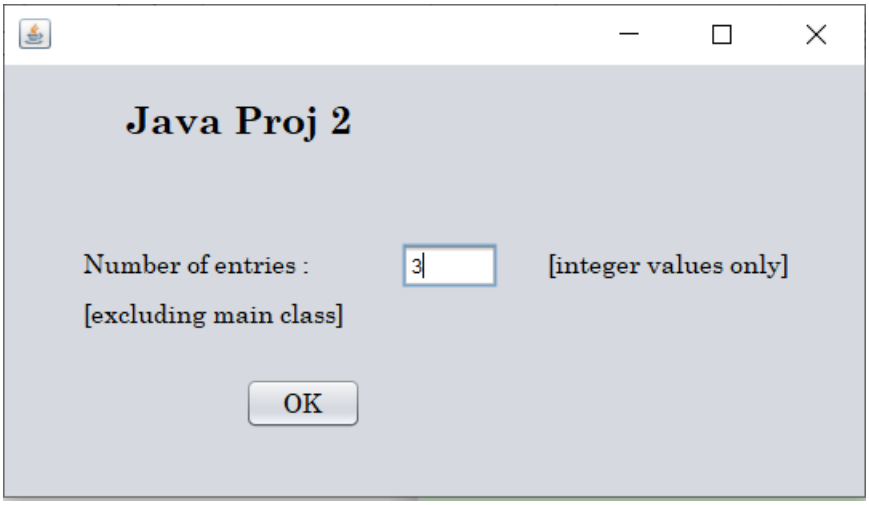


Figure 5:- Number of entries window

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.

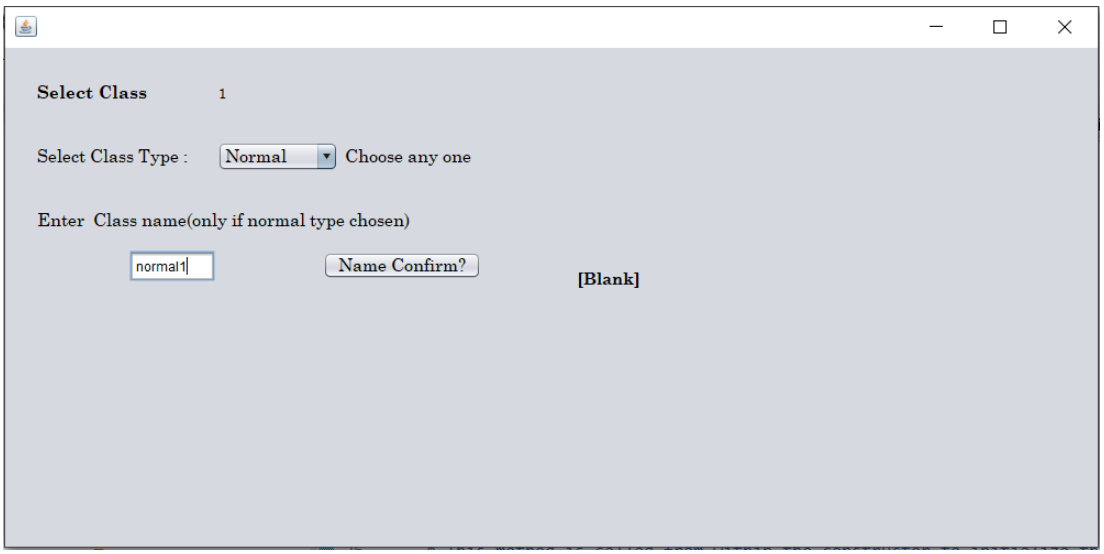


Figure 6:- Class entry window

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.

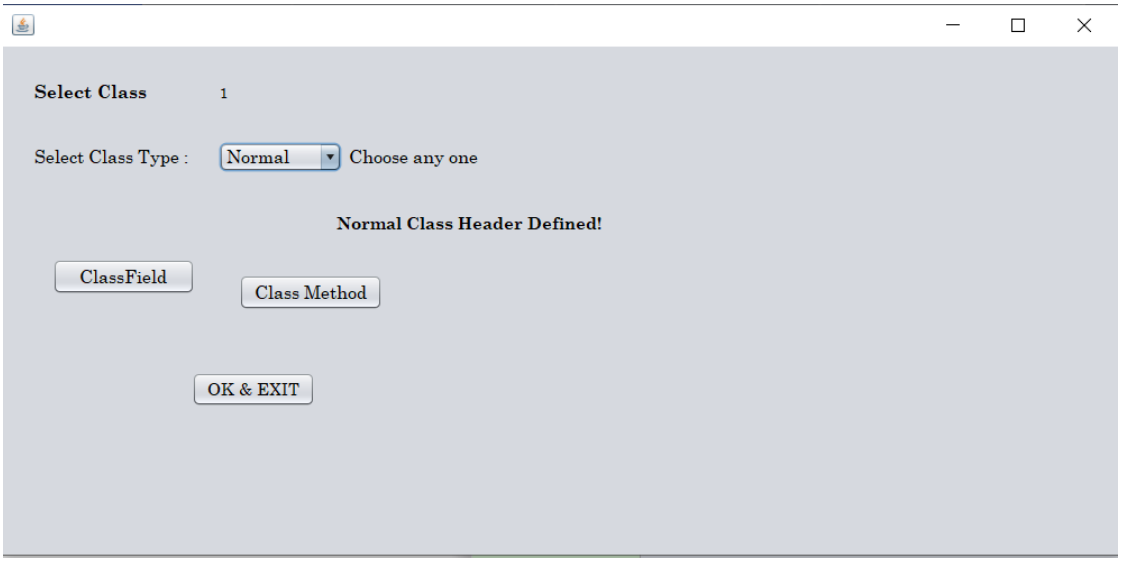


Figure 7:- Class entry window after name confirmation

After clicking on the name confirm button, the name confirm button is disables 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.

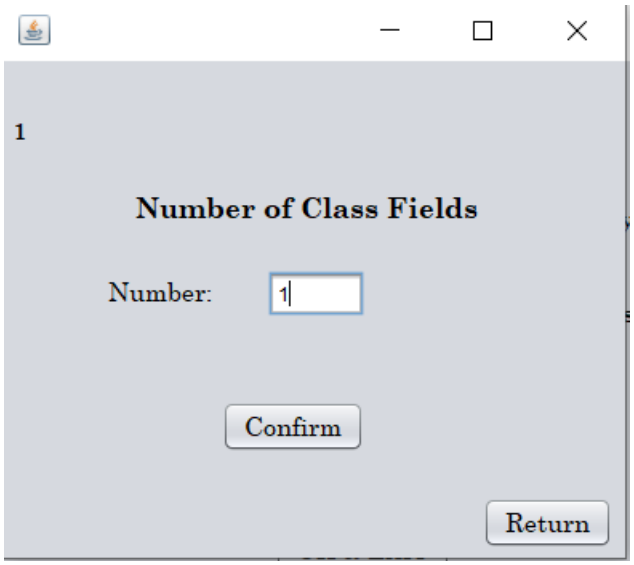


Figure 8:- Number of class fields entry window.

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.

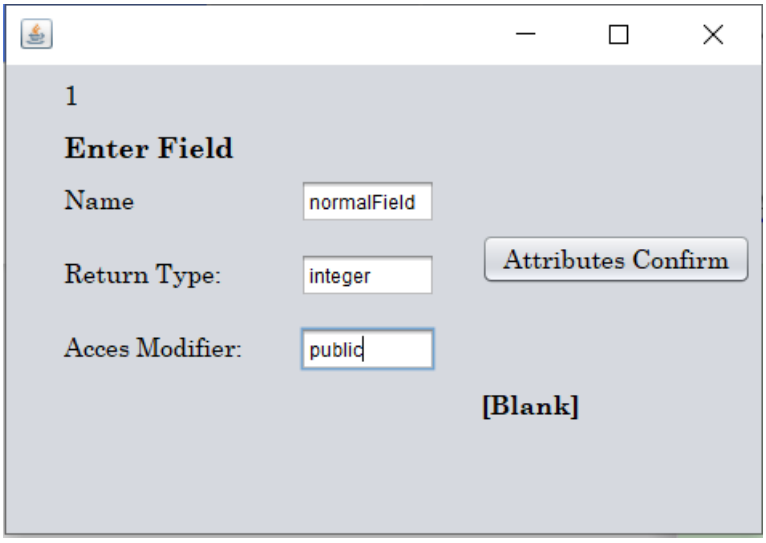


Figure 9:- Field entry window

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.

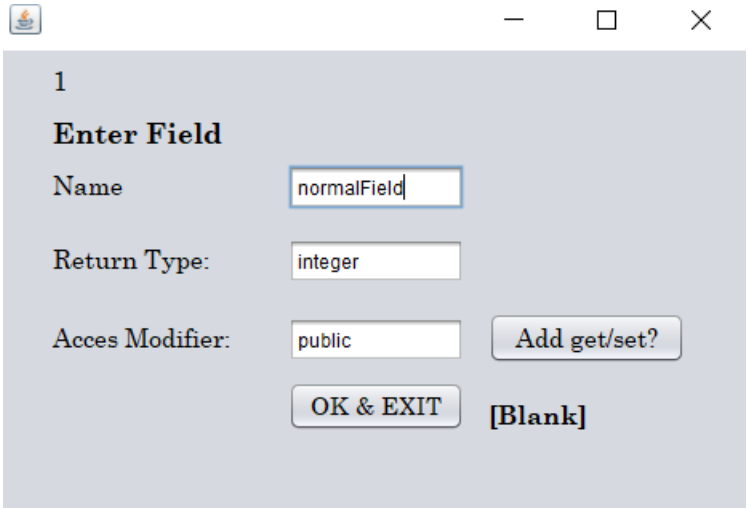


Figure 10:- Field entry window after attribute confirmation

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

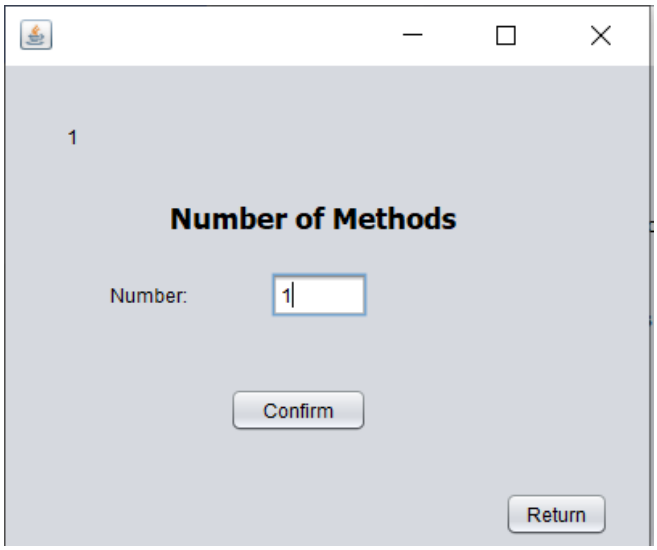


Figure 11:- Number of methods intake window

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.

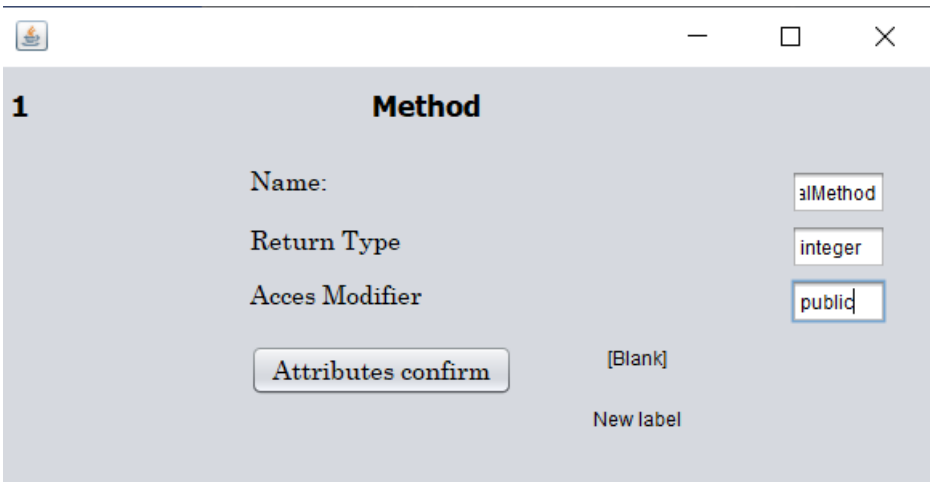


Figure 12:- Method entry box

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.

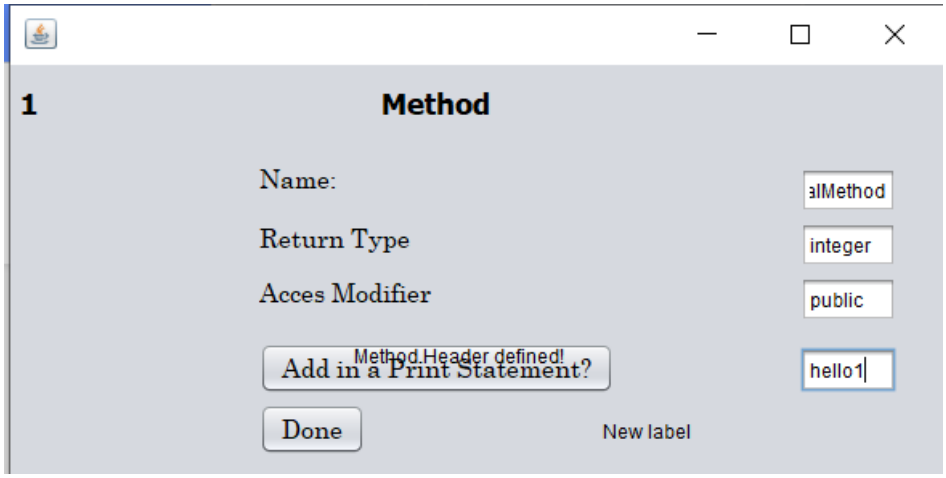


Figure13:- Method entry box after attribute confirmation

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.

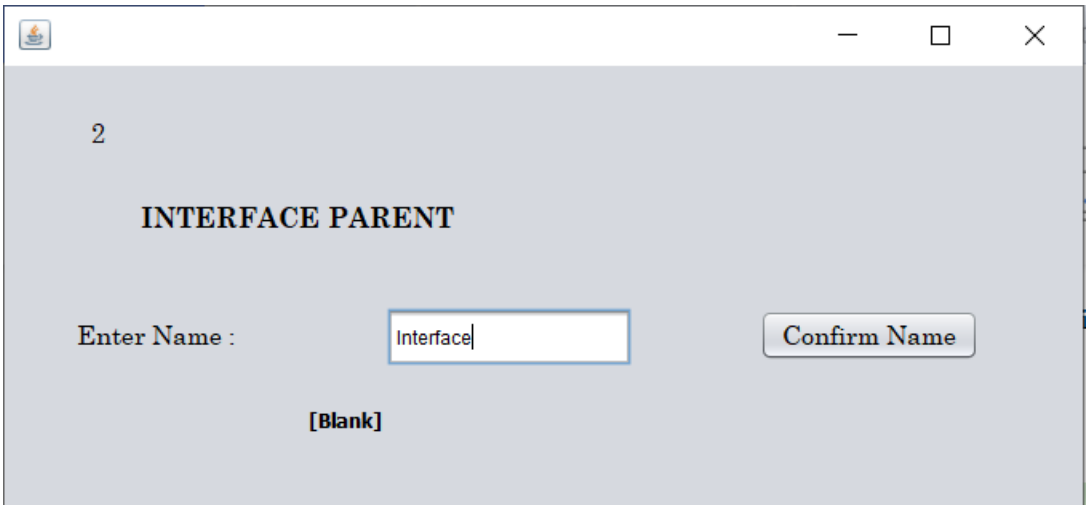


Figure 14:- Interface entry box

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.

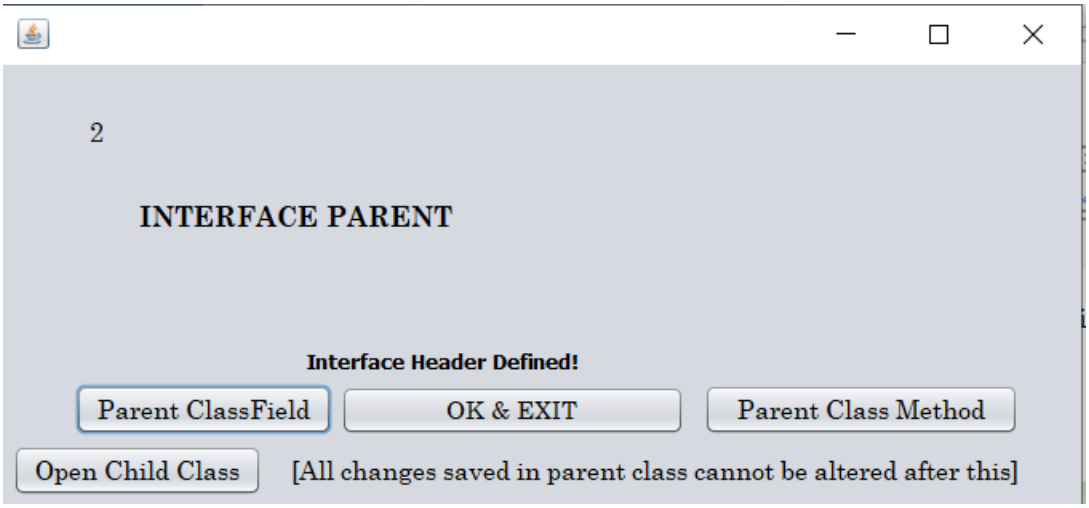


Figure 15:- Interface entry box after name confirmation

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.

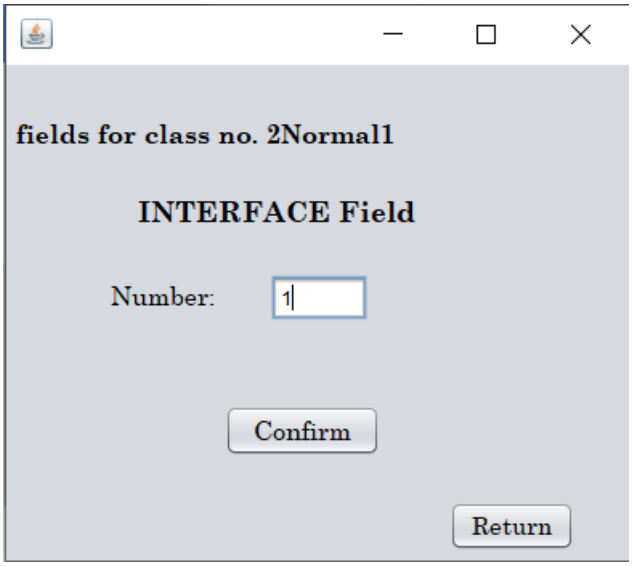


Figure 16:- Number of fields entry box for interface

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.

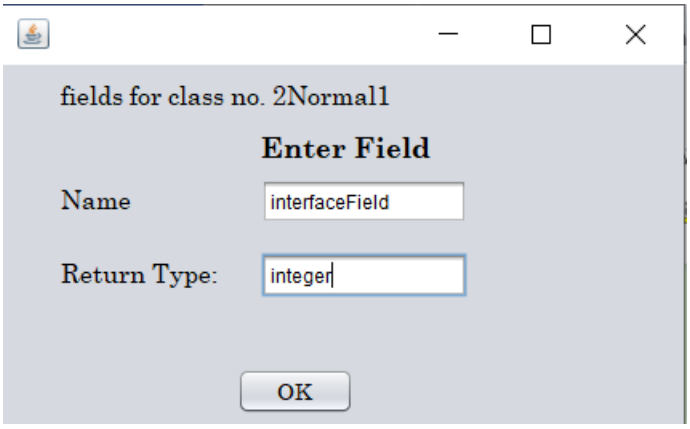


Figure 17:- Field entry box for interface

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.

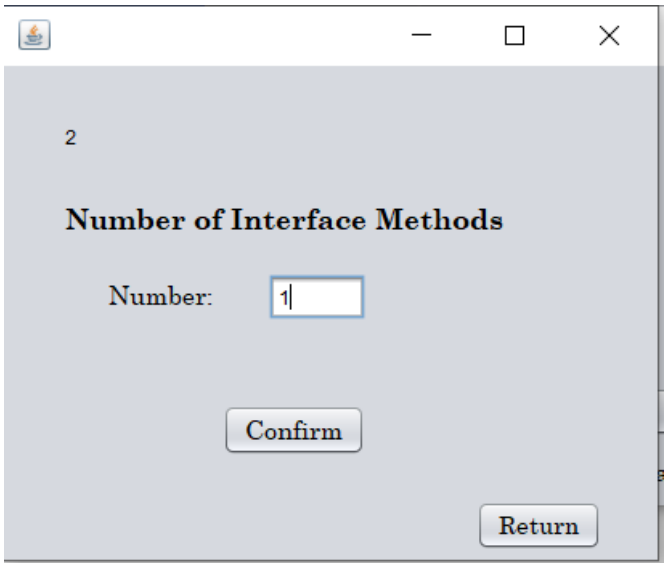


Figure 18:- Number of interface methods entry box

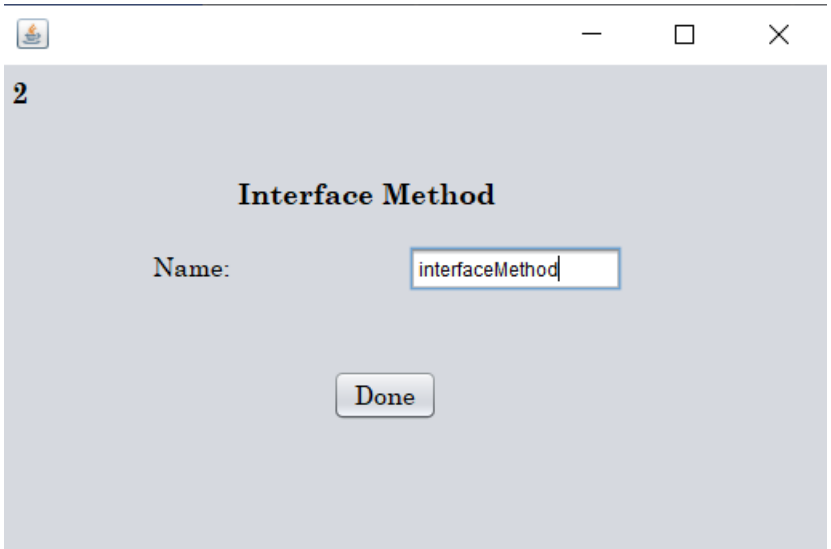


Figure 19:- Interface method entry box.

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.

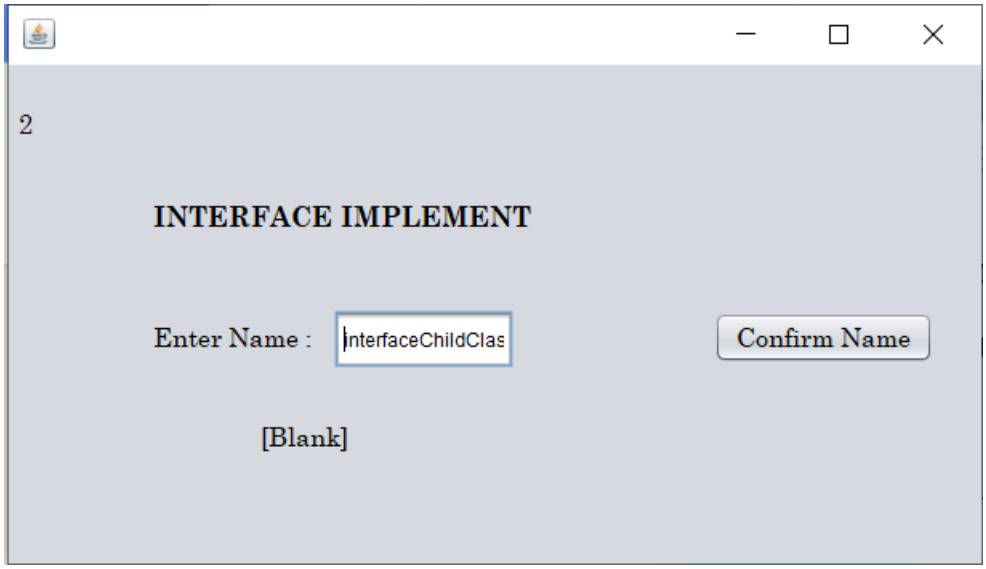


Figure 20:- Implemented interface entry box

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

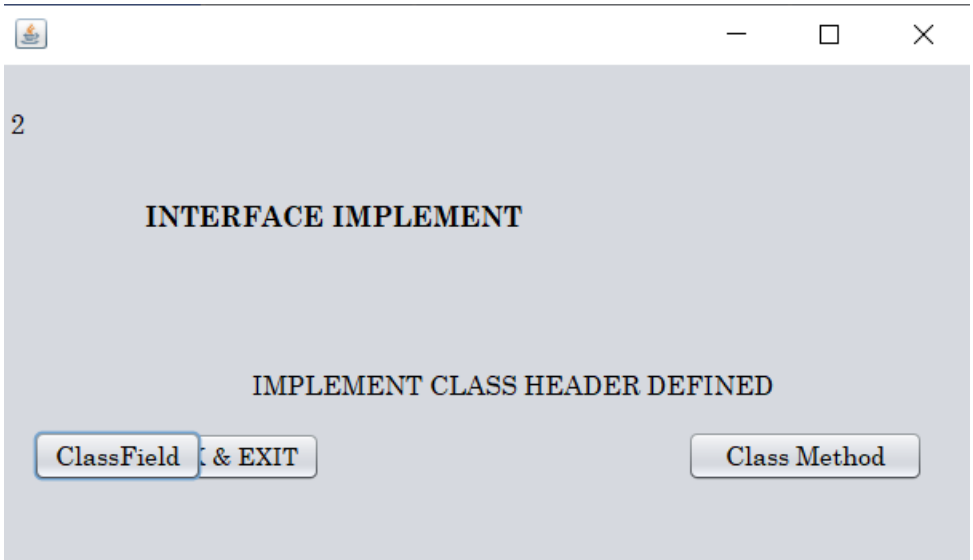


Figure 21:- Implemented interface entry box after confirmation

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.

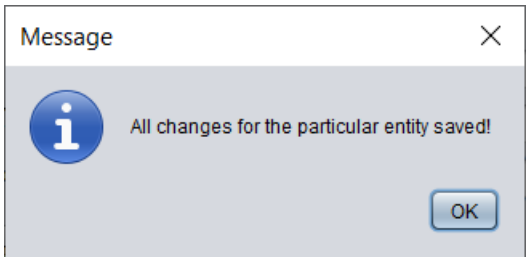


Figure 22:- All changes saved dialog box

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.

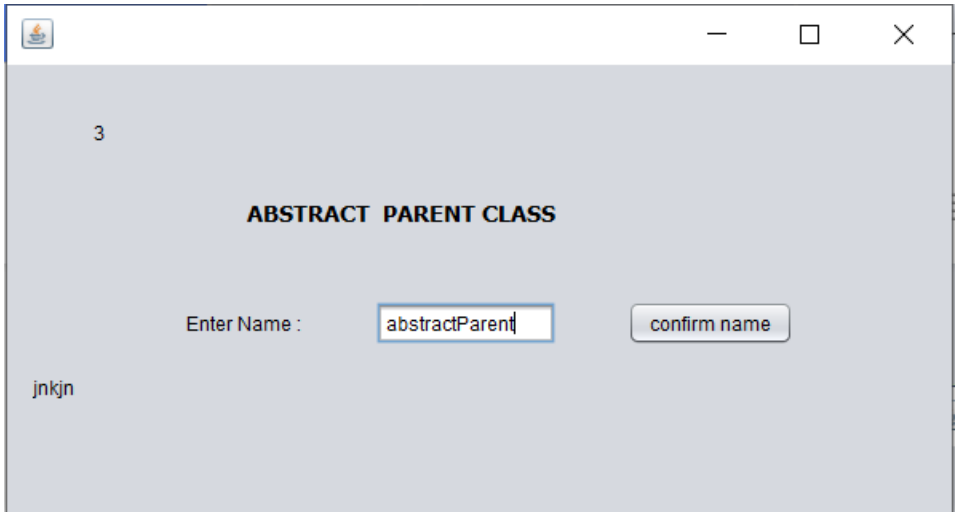


Figure 23:- Abstract parent entry box

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.

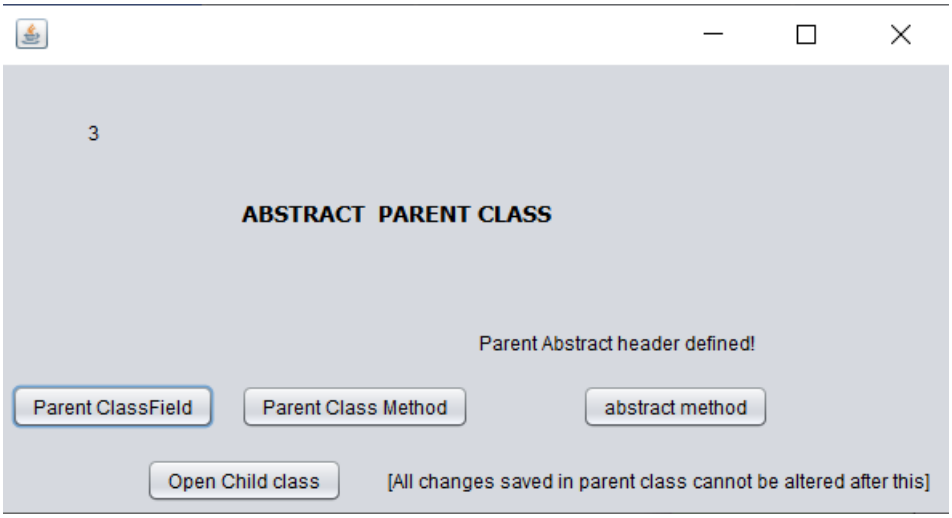


Figure 24:- Abstract parent entry box after name confirmation

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

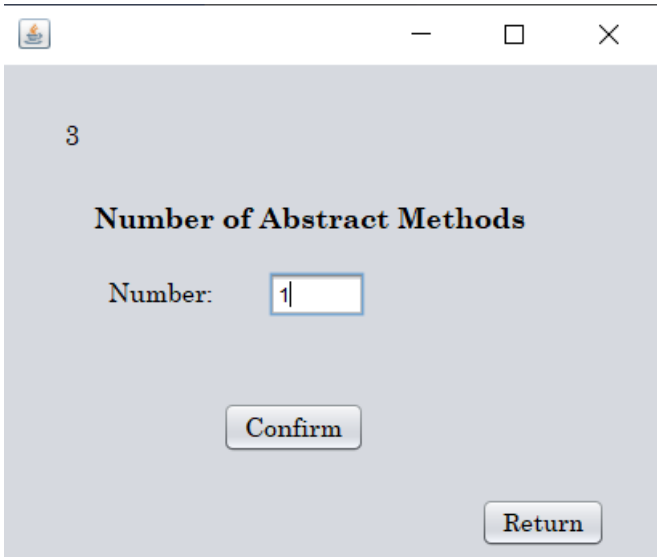


Figure 25:- Number of abstract methods entry box

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.

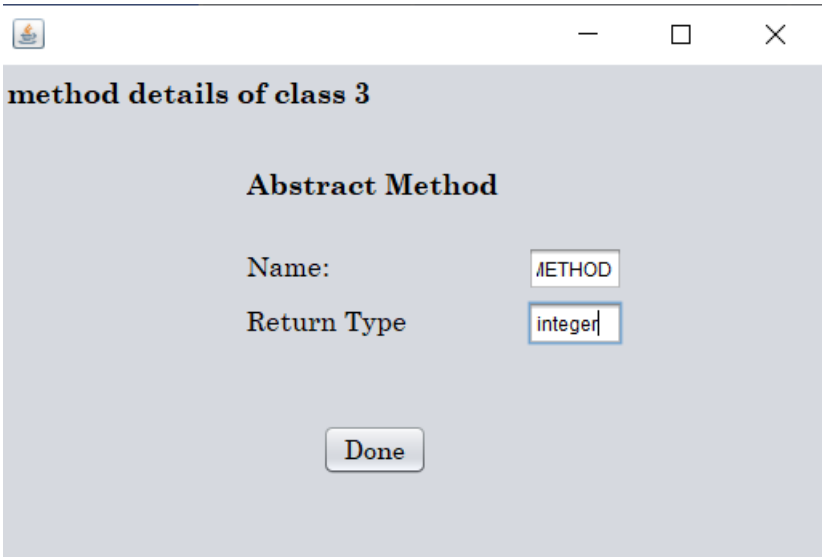


Figure 26:- Abstract method entry box

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.

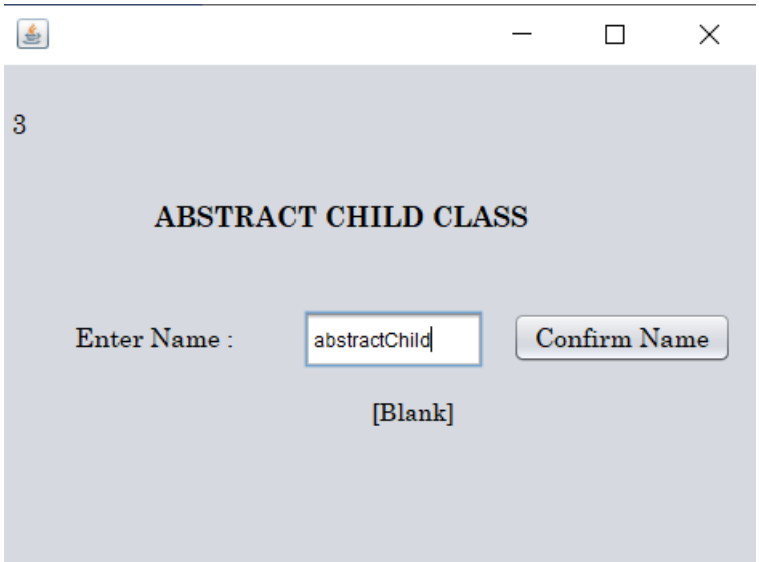


Figure 27:- Abstract child entry box

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.

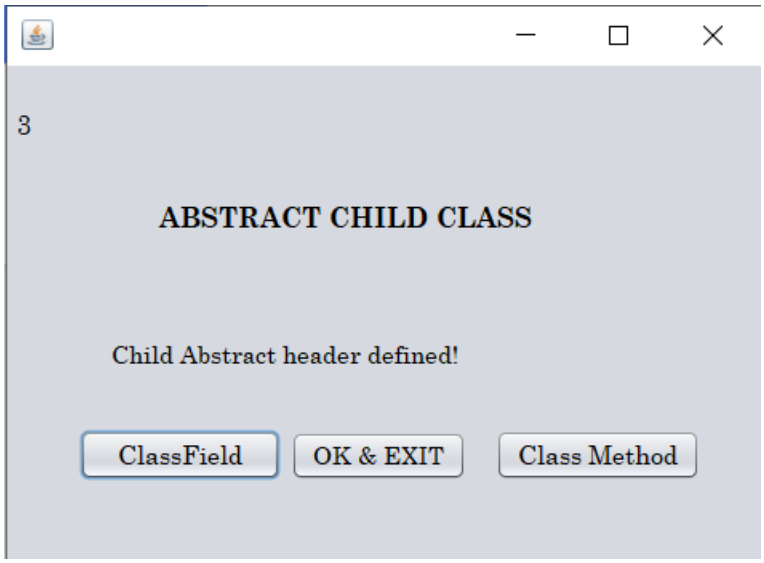


Figure 28:- Abstract class entry box after name confirmation

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() {
 }
 }
abstact class  abstractParent  {
public integer abstractField ;
public integer abstractMethod() {
 }
public abstract  integer METHOD() ;
 }
class  abstractChild extends abstractParent {
public integer abstractChildField ;
public integer abstractChildMethod() {
 }
 }
```

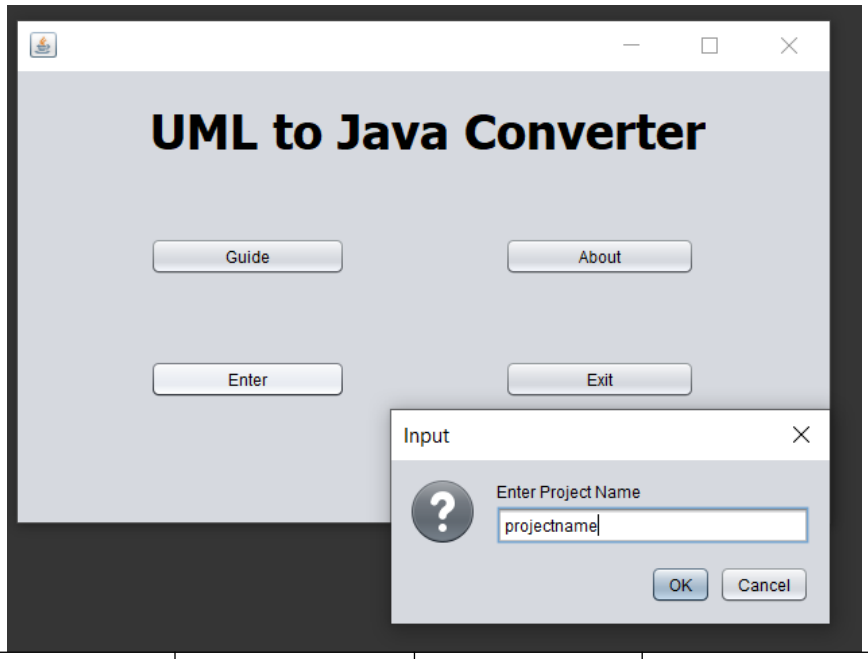
OUTPUT:

Figure 29:- Sample code output

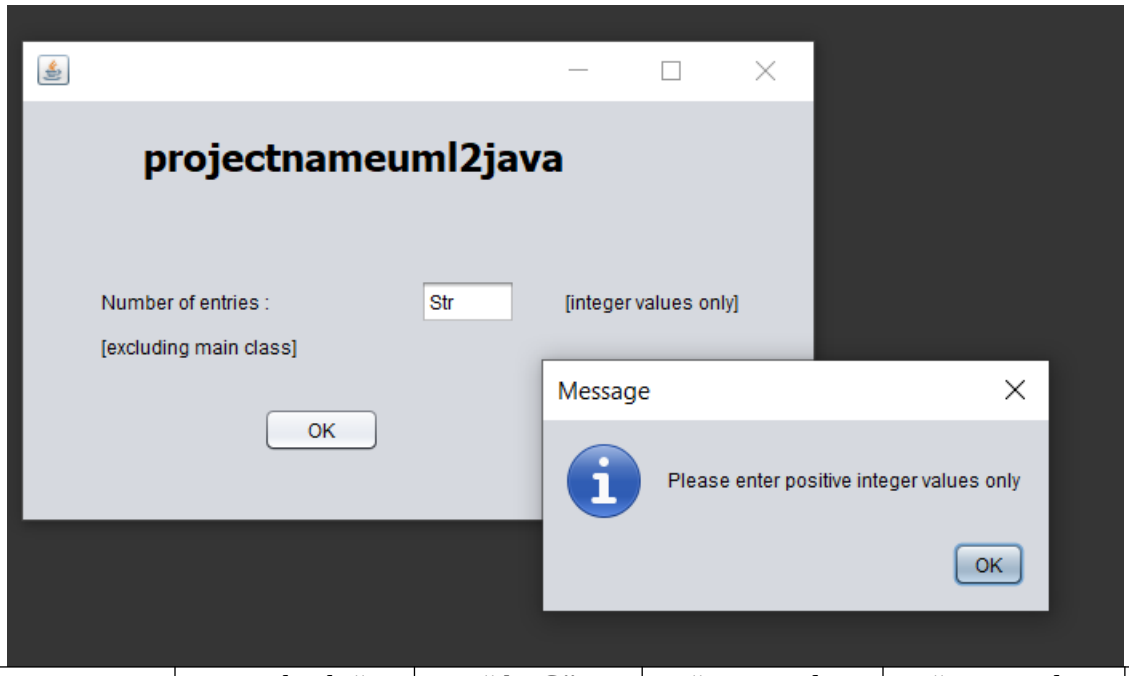
Finally figure 29 shows us the output code the user receives in the form of a text file once they have completely entered the whole UML into the program.

b) Testing

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

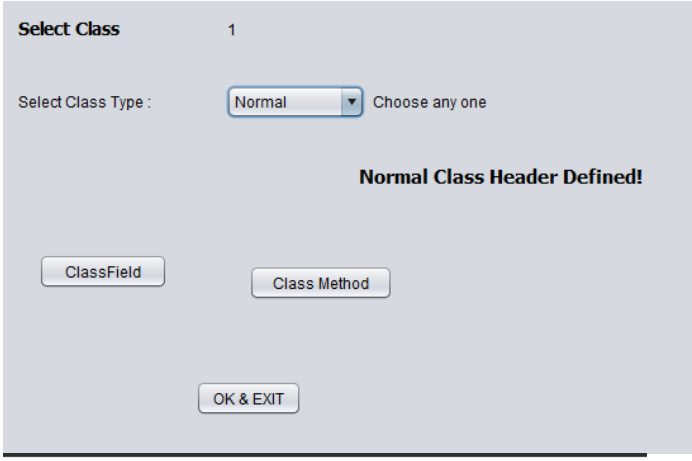


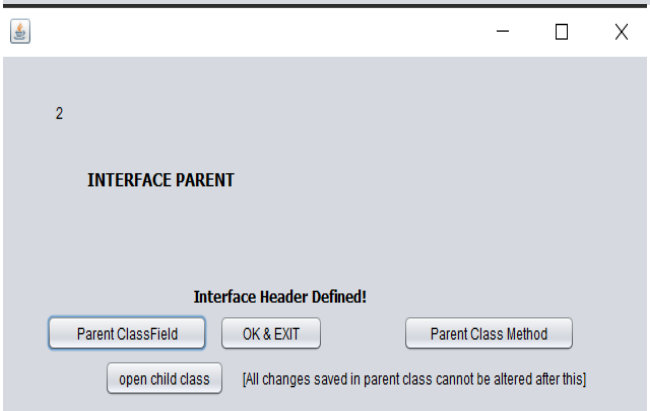
1.	To check whether "**Enter Project Name**" JOptionPane pops up after selecting Enter Jbutton.	Select Enter JButton from MainJFrame	"**Enter Project Name**" JOptionPane should pop up	"**Enter Project Name**" JOptionPane should popped up	Pass



2.	To check " **No.of entries**" text field is	"**ABC**"	"Enter only positive integers"	"Enter only positive integers"	Pass

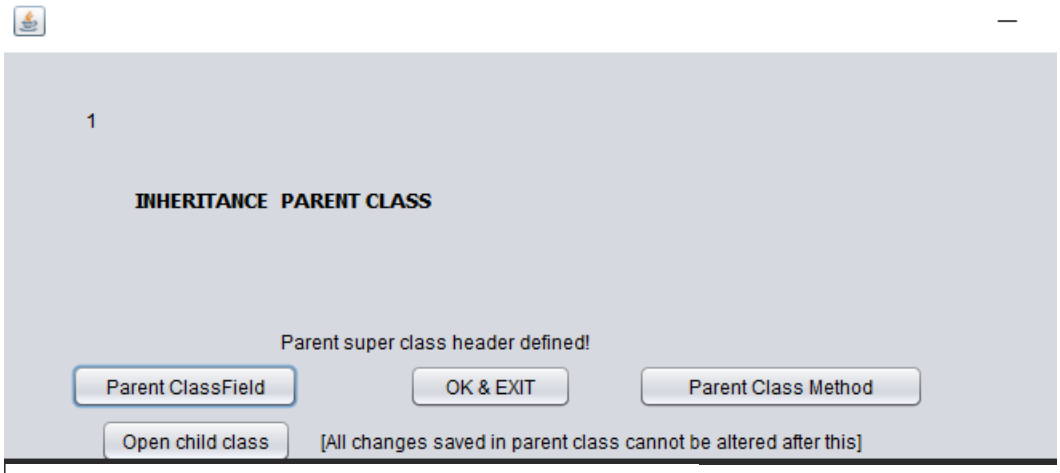
	accepting **any String** or not		message should pop up	message should popped up	
3.	To check “ **No.of entries**” text field is accepting **0** or **negative** or any other numeric positive datatype	**0** or any negative integer(= **-3**) or any double value(=**4.00**)	“Enter only positive integers”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

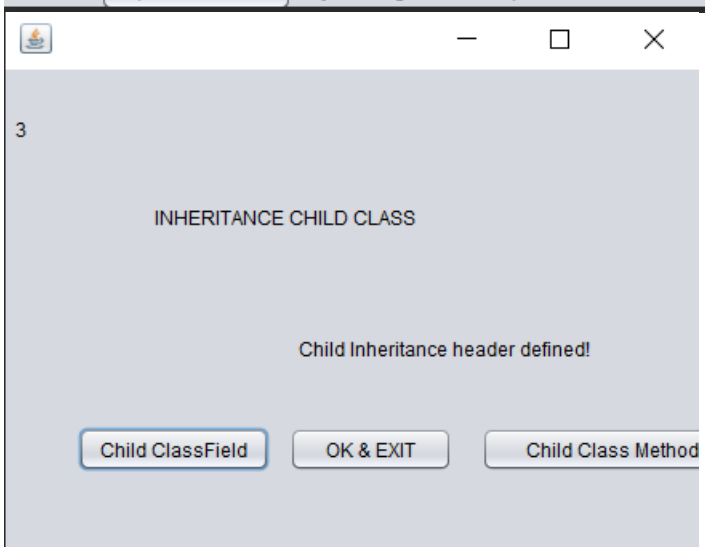




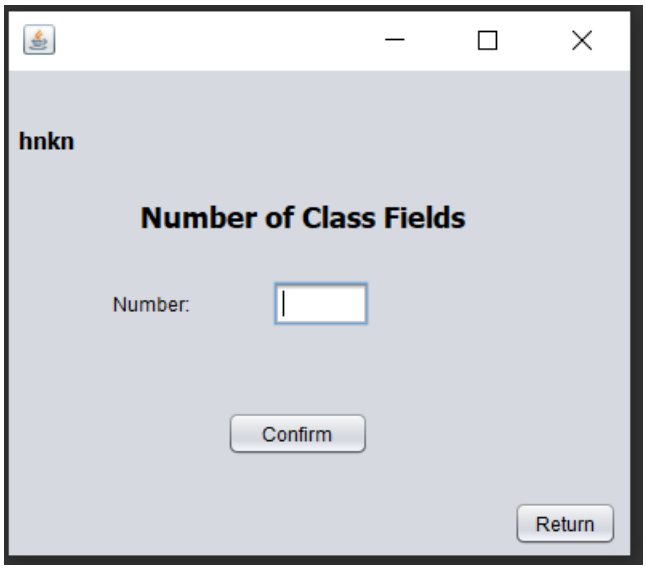
4.	To check whether **Normal Class header** gets confirmed after confirm button selection to populate class code further with fields and methods	Select “**Normal**” and Enter “**class name**”**(eg: Studentava)**	“Normal Class Header Defined” should show up in label And Class name should get set to **Studentava**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

	whether **Interface Class header** gets confirmed after confirm button selection to populate class code further with fields and methods	"**Interface**"and enter "**class name**"**(eg: Studentava)**	Header Defined" should show up in label And Class name should get set to **Studentava**	Header Defined" shows up in label And Class name gets set to **Studentava**	
6.	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
7.	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

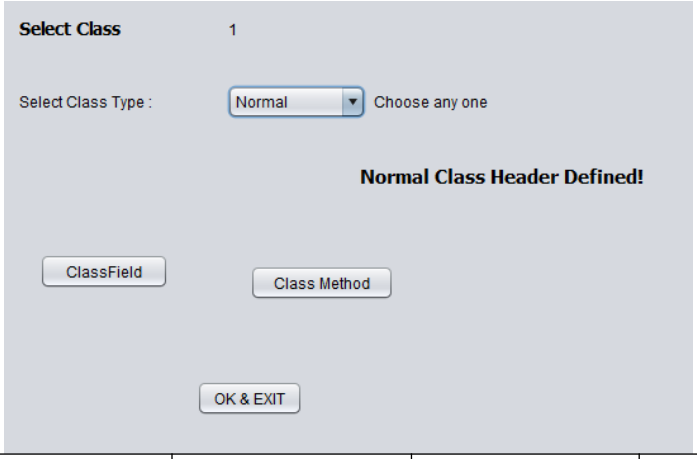




8.	To check whether **Child Class header** gets confirmed after confirm button selection to populate class code further with fields and methods	Select "**Child class**" after selecting "**parent class**" and enter "**class name**"	"**Child class Header Defined**" should show up in label	"Child Class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



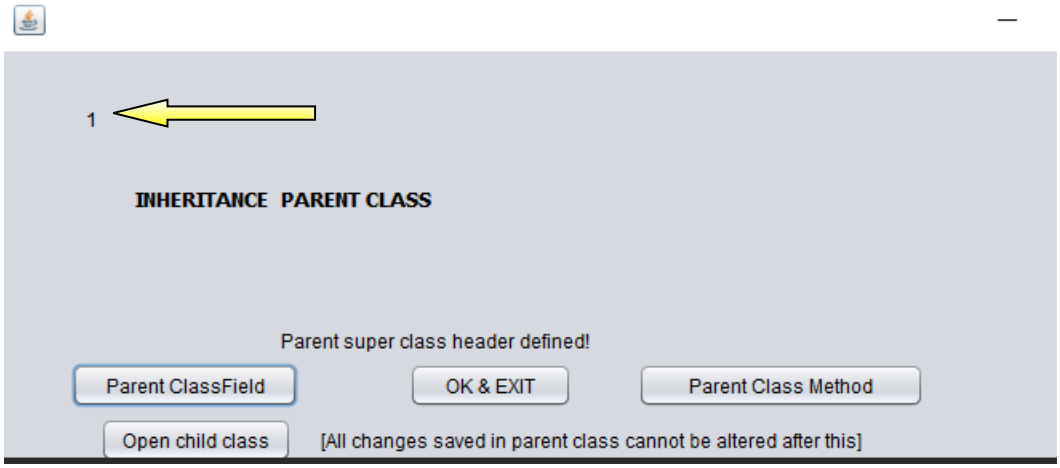
9.	To check whether "**Class field form**" gets disposed after selection of **return** button	Select **return button**	Field Form should dispose without affecting code	Form disposed successfully without affecting output code	Pass
10.	To check whether "**Class Method form**" gets disposed after selection of **return** button	Select **return button**	Method Form should dispose without affecting code	Form disposed successfully without affecting output code	Pass



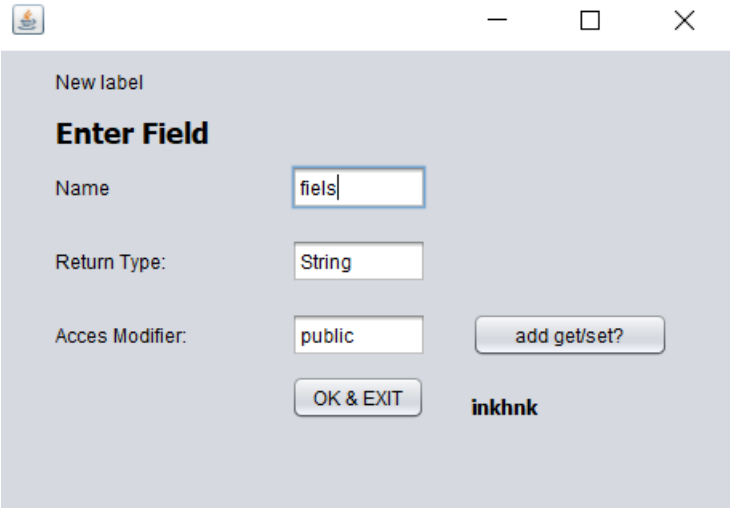
11.	To check whether Form disposes on selecting "**OK&EXIT**"	"**OK&EXIT**" Button clicked	The particular form should dispose with all input values being saved	Form disposed successfully with all changes saved	Pass

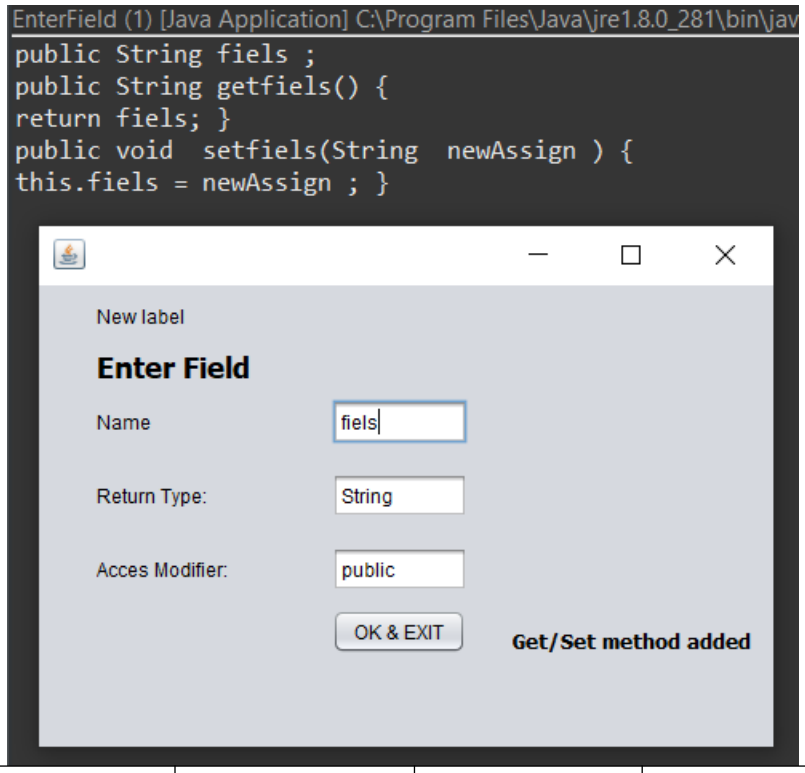
	button				

12.	**Number of class fields** entered as zero	Entered “0”	“number of class fields cannot be zero” message should pop up	Form disposes without affecting code similar to return button	Fail
13.	**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



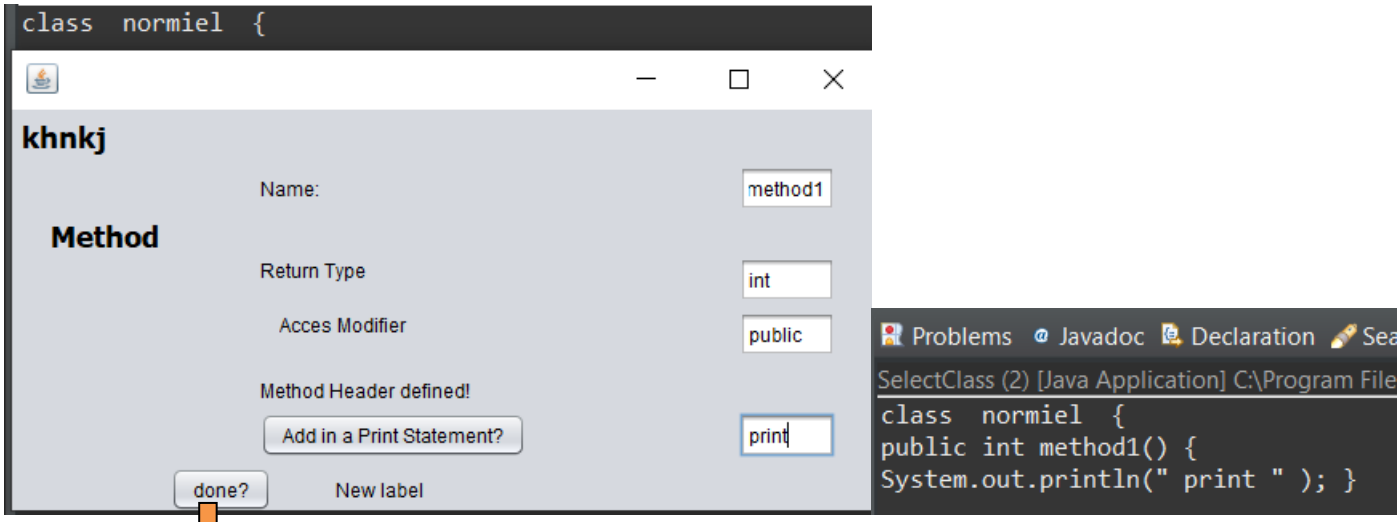
14.	To check whether All subsequent **form labels** should get numbered automatically when there are multiple classes to avoid confusion	Number of entries set to **4**	**Respective** Class field and class methods get numbered according to the the Class number	Respective Class field and class methods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



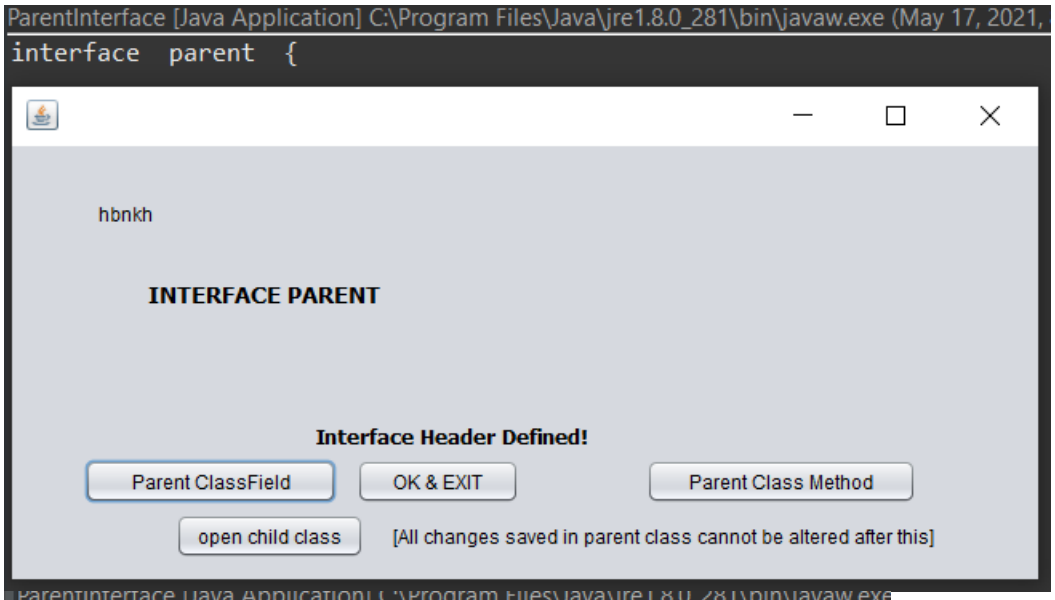


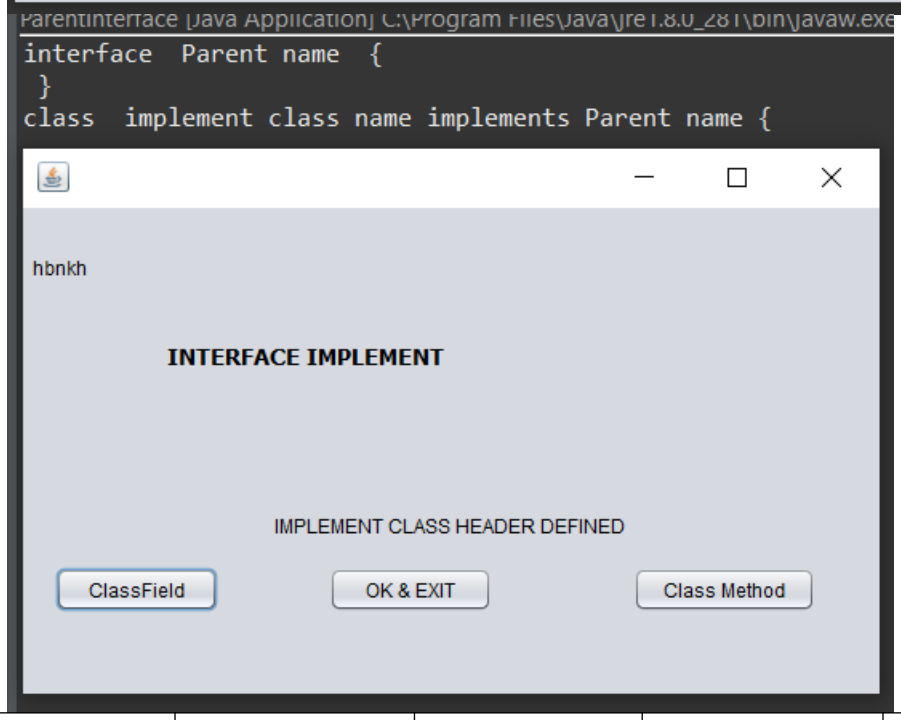
15.	To check whether **Custom Get/set method** should appears only after confirming field attributes	Select **Attributes confirm button**	**Add Get/set** button should appear	**Add Get/set** button appeared	Pass
16.	To check whether Access field members can be accessed using **add**	Select **Add Get/set**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

	get/set button			field members	



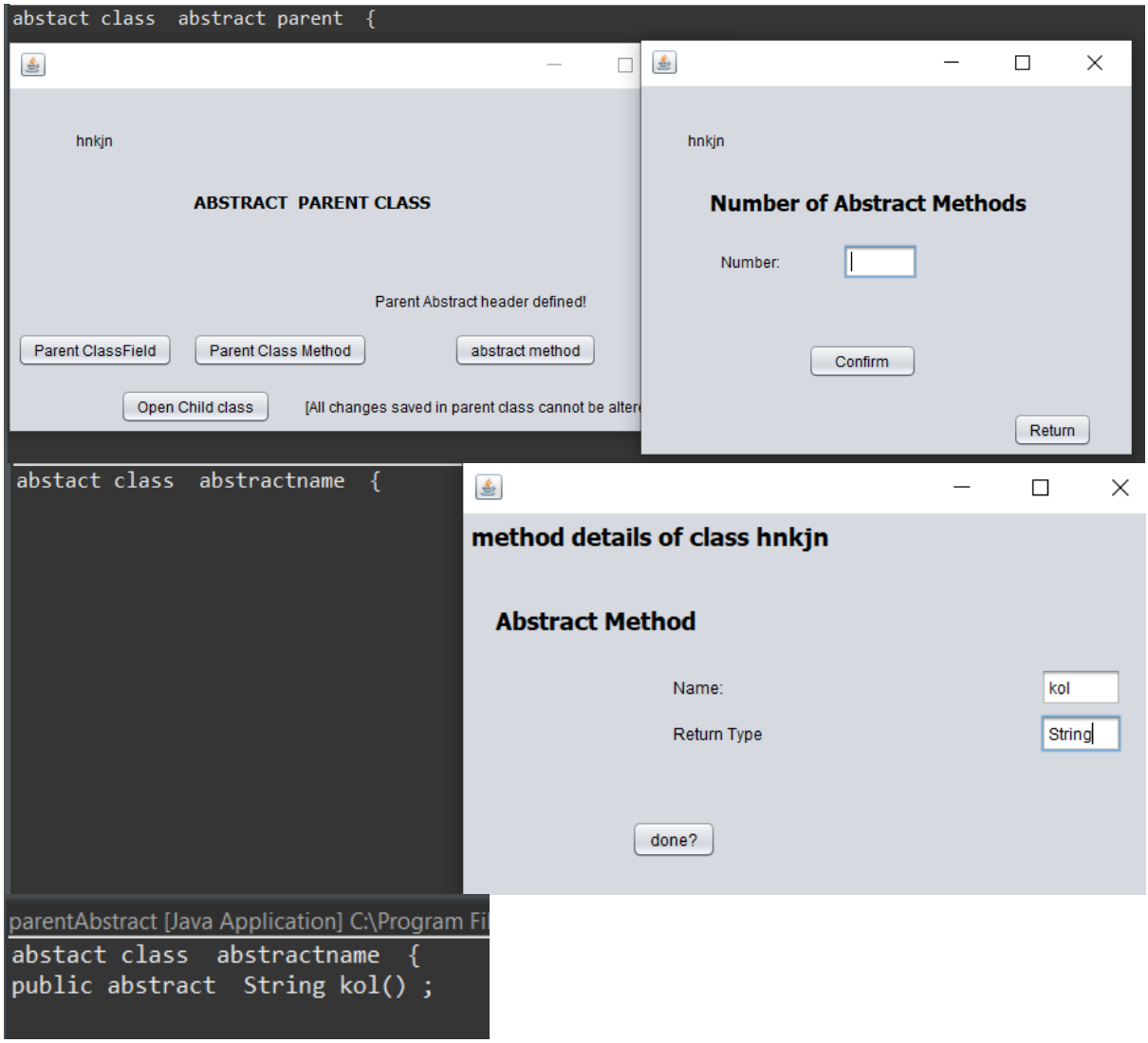
17.	To check whether changes in method form gets saved after selecting "**done?**" button	Select "**done?**" button	Output code file should get updated with method details	Method of the particular class gets updated with the input info	Pass
18.	To check whether Multiple custom add print statements can 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





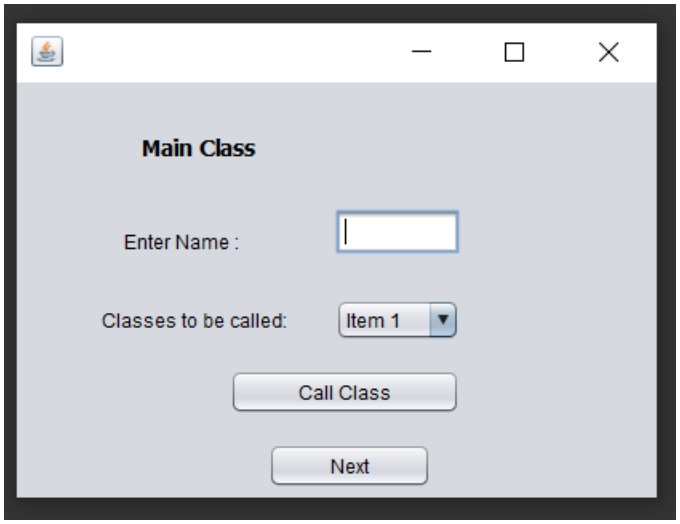
19.	To check whether **Implement class** can be **added** or not on opting for Interface Class type	Choose **Interface class type** from Select class type form	Should **open** parent form and give choice for implement class	Opened parent form and gave choice for implement class	Pass
20.	To check whether **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				
21.	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

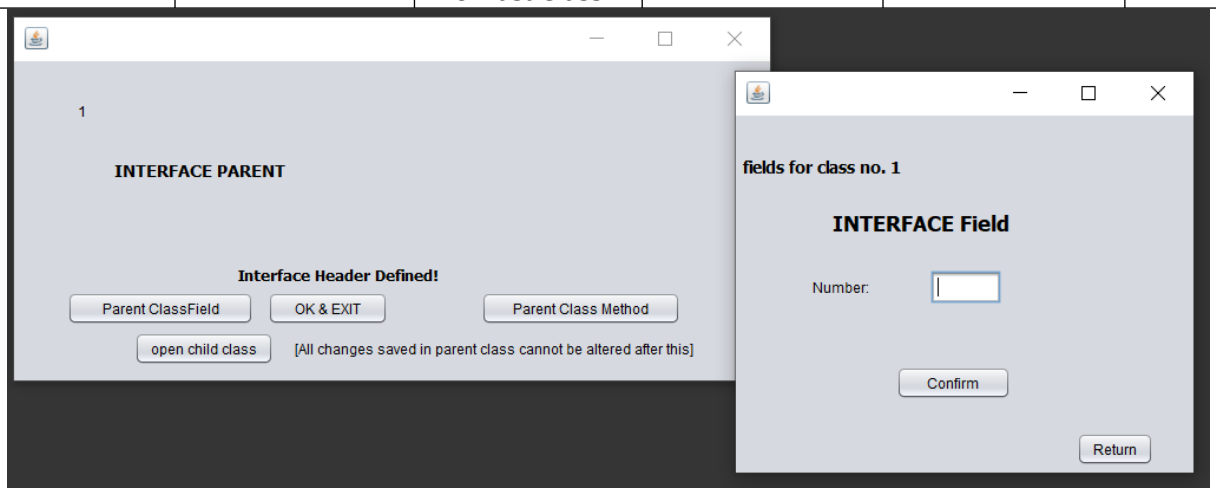


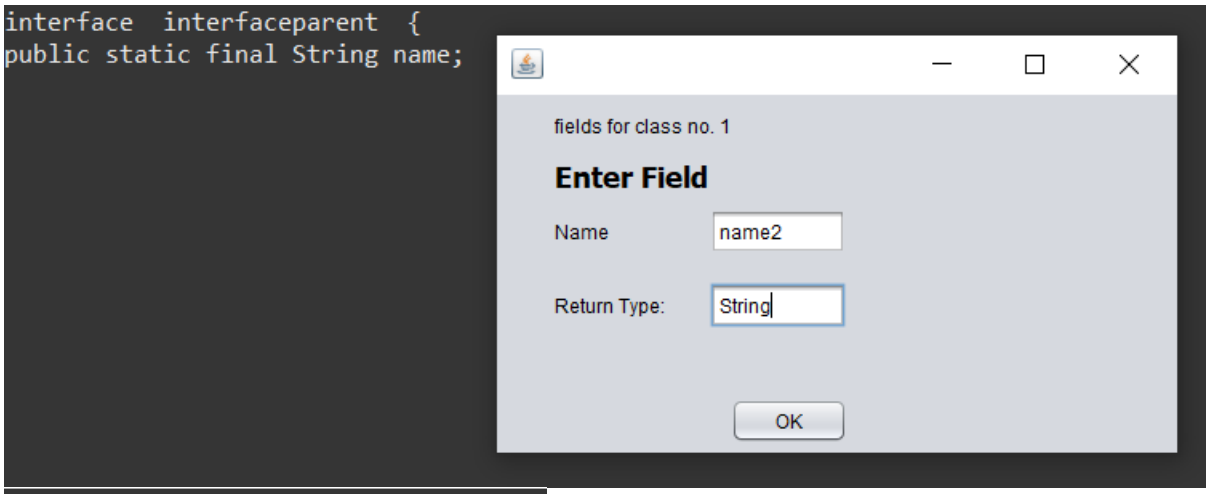
22.	To check whether a choice is offered between abstract methods and normal methods in abstract	**Abstract class** type chosen from Select class type form	**"Abstract method"** button should open different form other than normal method	"Abstract method" button opened different method form other than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

	parent class			different	



23.	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






```
[Console output redirected to file:C:
interface  interfaceparent  {
public static final String name;
public static final String name2;
```

24.	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

Table 2.19

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 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. References:

1. WindowsBuilder documentation
2. Swing documentation
3. Visual paradigm