CHAPTER 6
TESTING

6.1 Test Approach

Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. Software testing also provides an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include, but are not limited to, the process of executing a program or application with the intent of finding software bugs.

Black box testing:

In black box testing we test the system at random for some random functionalities and depending on the output that we get we come to the conclusion that whether the system we have built is right or wrong. Internal system design is not considered in this type of testing. Tests are based on requirements and functionality. The number of modules and number of java files required for each module is checked.

White box testing:

This testing is based on knowledge of the internal logic of an applications code. Also known as Glass box Testing. Internal software and code working should be known for this type of testing. Tests are based on coverage of code statements, branches, paths, conditions. All the modules are tested for their logic whether it functions properly or not. Code is checked by inserting different inputs to check its functionality.

Unit testing:

Testing of individual software components or modules. Each module was runner separately to check the output. Unit testing focuses first on the modules, independently of one another, to locate errors. This enables the tester to detect errors in coding and logical errors that is contained within that module alone. Those resulting from the interaction between modules are initially avoided. Here we test each module individually and integrate the overall system. Unit testing focuses verification efforts even in the smallest unit of software design in each module. This is also known as module testing. The modules of the system are tested separately.

Integration testing:

Integration testing is the testing process in software testing to verify that when two or more modules are interact and produced result satisfies with its original functional requirement or not. Integrated testing will start after completion of unit testing.

User Acceptance Testing:

User acceptance testing of the system is the key factor for the success of any system. A system under consideration is tested for user acceptance by constantly keeping in touch with the prospective system at the time of development and making change whenever required. This is done with regard to the input screen design and output screen design.

In order to achieve the more usability of the android application the User Acceptance Testing will be performed. Here we will test whether the proposed system is having well defined UI so that the citizens can interface the application more easily.

Functional Testing:

Functional testing is a technique in which all the functionalities of the program are tested to check whether all the functions that where proposed during the planning phase is full filled. This is also to check that if all the functions proposed are working properly. This is further done in two phases One before the integration to see if all the unit components work properly. Second to see if they still work properly after they have been integrated to check if some functional compatibility issues arise.

6.2 Test Cases

Test cases are built around specifications and requirements, i.e. what the application is supposed to do. Test cases are generally derived from external descriptions of the software, including specifications, requirements and design parameters. Although the tests used are primarily functional in nature, non-functional tests may also be used. The test designer selects both valid and invalid inputs and determines the correct output without any knowledge of the test object's internal structure.

Table 6.1: Test Cases

Steps	Test steps	Test data	Expected result	Actual Result	Status (Pass/Fail)
1	Start>class diagram>create new class structure	Class name:Area Variables:int a,b,c; Methods:void area();void display();	User should be able to generate class structure	User navigated to canvas page with successful creation.	Pass
2	Start>activity diagram>create new activity stucture	Variables:int a=20,b=10; Decision:if(a>b)	User should be able to generate activity structure.	User navigated to canvas page with successful creation.	Pass
3	Create relationship between two class entities.	class to class	Relationship between class to class is created.	Relationship between class to class is created.	Pass
4	Merging of class and activity code	Activity code into class code	Activity code should merged with the class code	Activity code and class code are generated at the different location	Fail

5	jarMaker>select class file>generate jar file	Main class file	Executable jar file should be generated ate the define location.	Jar file is generated at the user define location.	Pass
6	Save and load image	Diagram drawn by user	Image should be saved of class or activity diagram	Image is saved of class or activity diagram	Pass
7	Same class name over ridden	Same names of classes	It should create secondnew class as “New class 1”	It creates second new class as “new class”	Fail

6.3 Test Result

1. Select Diagram.

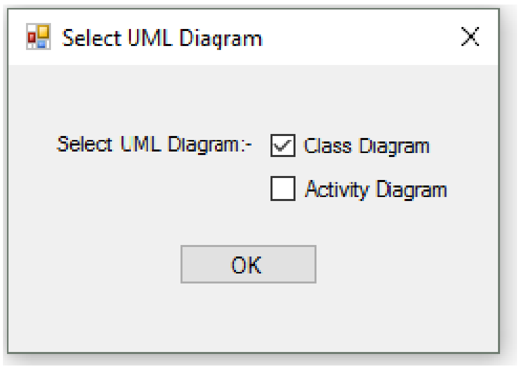


Figure 6.1: Select Diagram form

- Fig 6.1 select diagram form provides user an option to select class or activity diagram for generating java code.
- User can select any one of the two diagrams.

2. Main GUI.

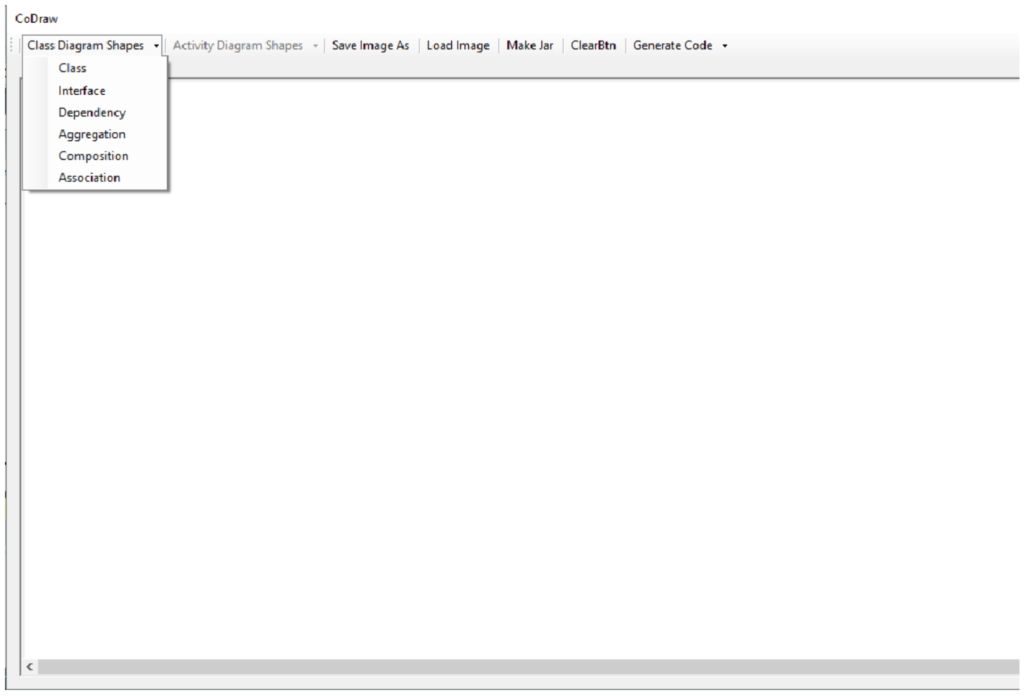


Figure 6.2: Main GUI

- Fig 6.2 is the main GUI of application .
- User can select various shapes or components based on diagram to be drawn. It also provides jar file generator and image file creator option.

3. Class diagram.

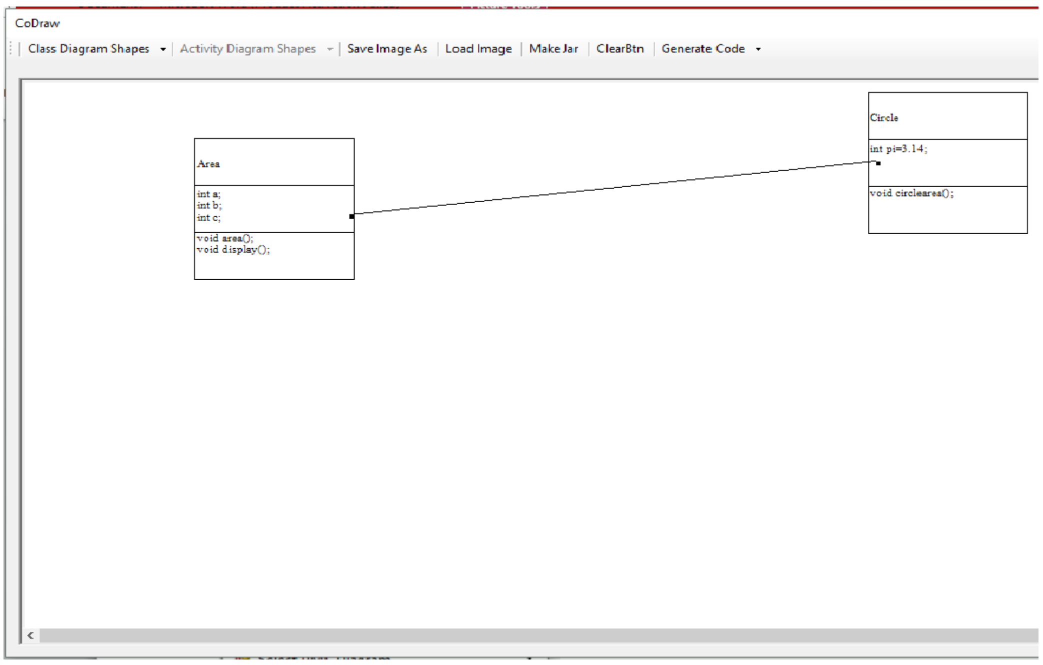
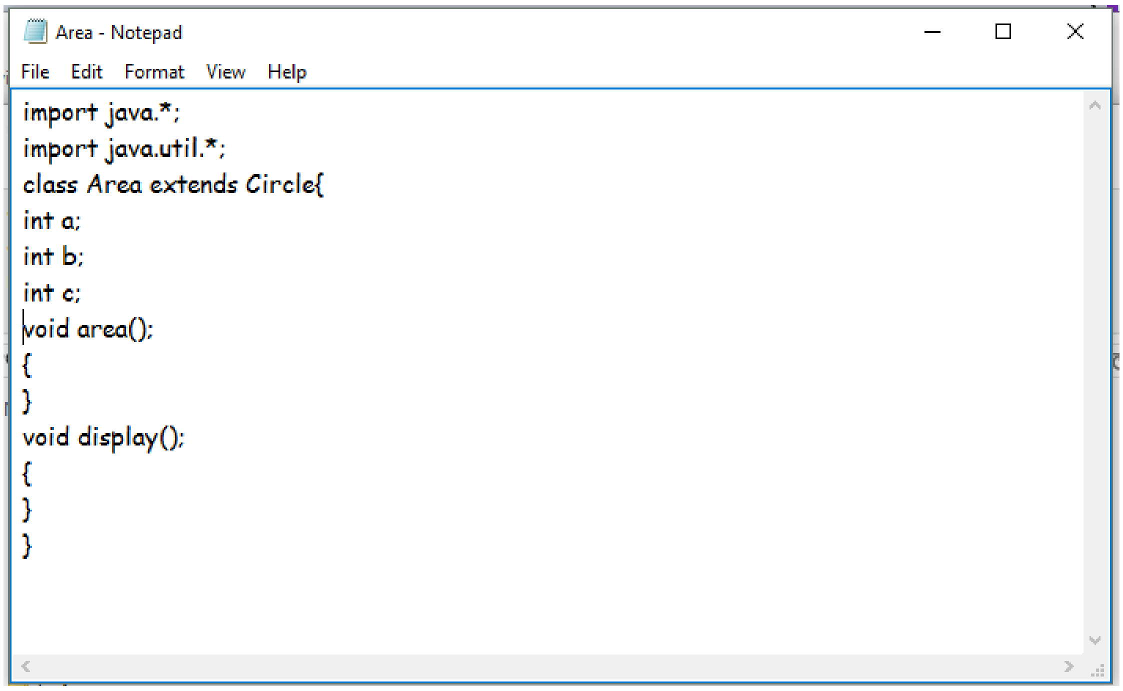


Figure 6.3: class Diagram

- Fig 6.3 shows class diagram drawn according to user’s requirement.
- User can show dependency or can inherit class from another class.
- User can enter different variables and methods as required for application.

4. Code generated from class diagram(inherited).

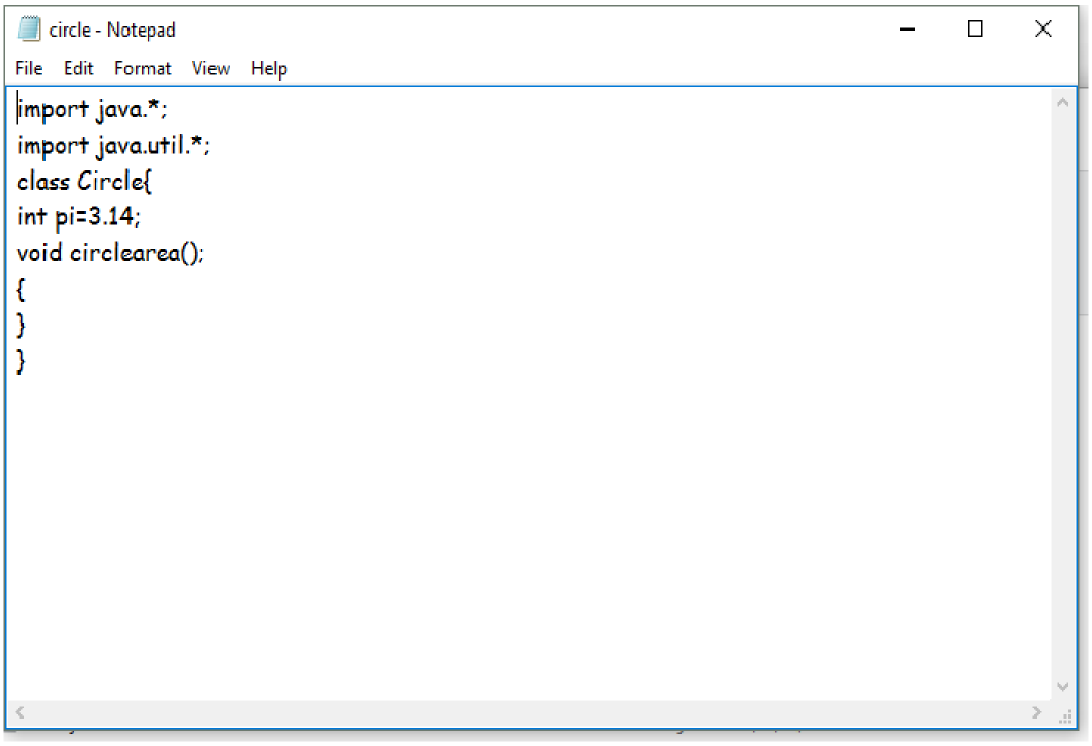


```
import java.*;
import java.util.*;
class Area extends Circle{
int a;
int b;
int c;
void area();
{
}
void display();
{
}
}
```

Figure 6.4: code generated from class diagram(inheritance)

- Fig 6.4 shows the generated code from class diagram.
- Class diagram gives structural view of the expected java code. In the above figure class Area extends class Circle and defines various variables and methods.

5. Code generated from class diagram.

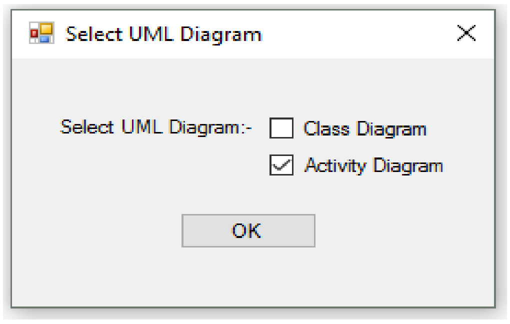


```
import java.*;
import java.util.*;
class Circle{
int pi=3.14;
void circlearea();
{
}
}
```

Figure 6.5: code generated from class diagram

- Fig 6.5 shows the generated code from class diagram.
- Class diagram gives structural view of the expected java code. In the above figure class Circle is main diagram and contain method and variables.

6. Select diagram(activity diagram).

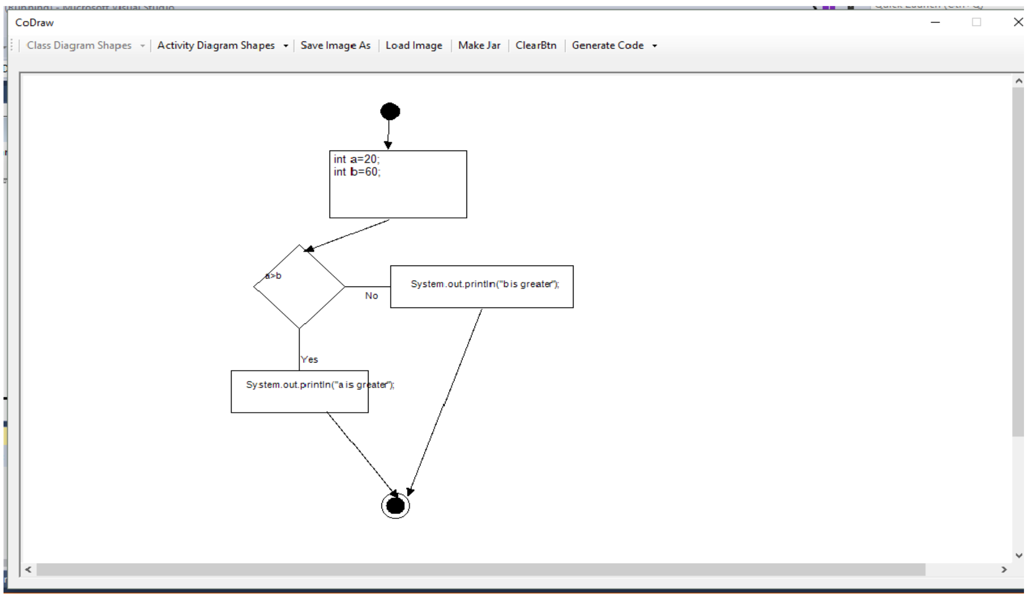


Select UML Diagram

Select UML Diagram:- [ ] Class Diagram [✓ Activity Diagram]

Figure 6.6: select diagram(activity diagram)

7. Activity diagram.



```
int a=20;
int b=60;
```

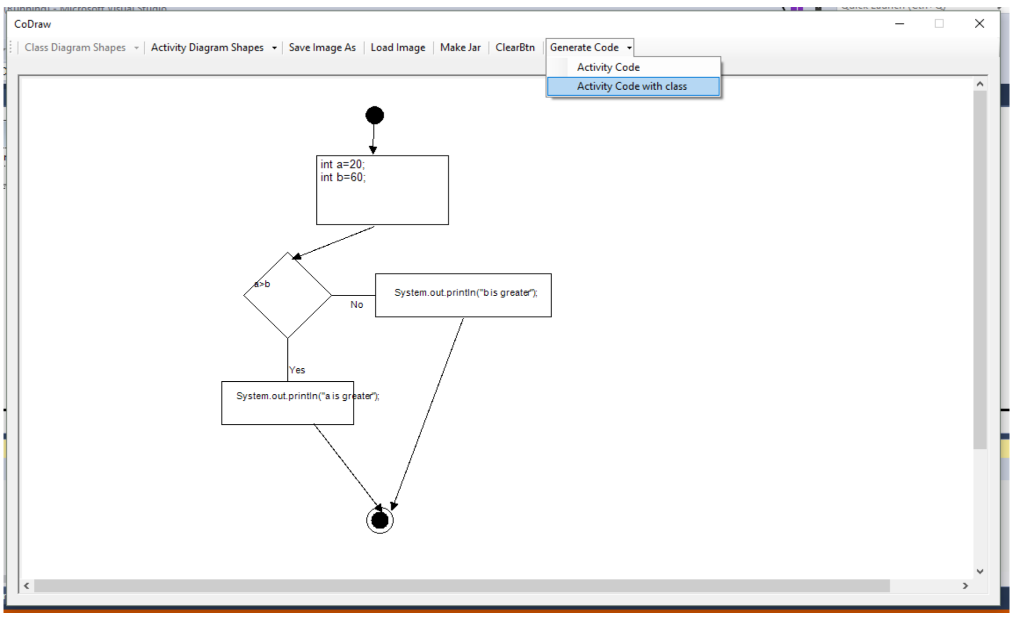
```
System.out.println("b is greater");
```

```
System.out.println("a is greater");
```

Figure 6.7: Activity diagram

- Fig 6.7 shows the activity diagram drawn on user interface.
- Activity diagram gives behavioural view of the expected java code. In the above figure “if else” condition is used to generate result.

8. Generate activity diagram.



```
int a=20;
int b=60;
```

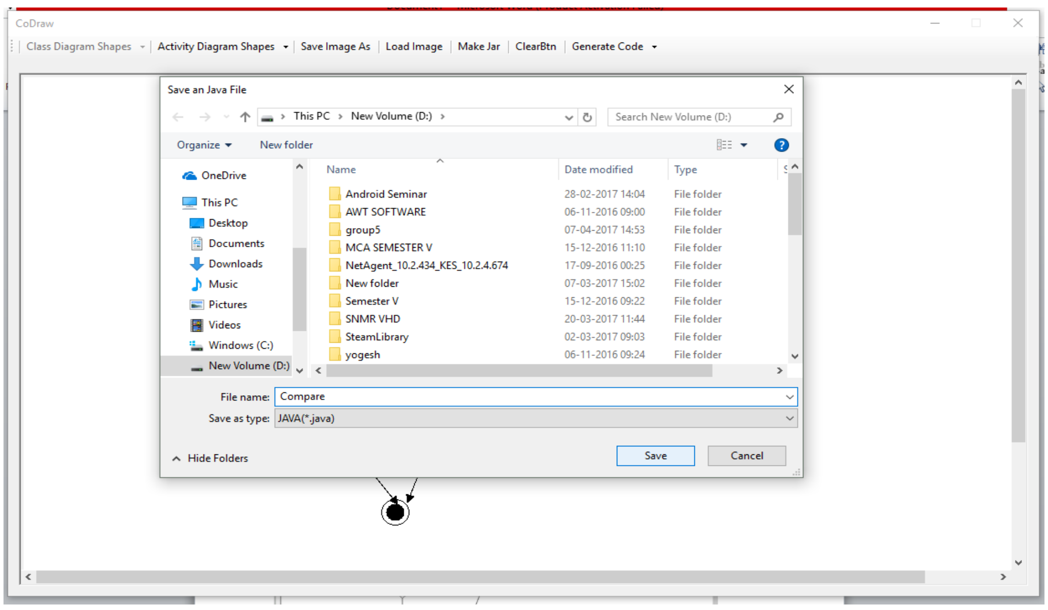
```
System.out.println("b is greater");
```

```
System.out.println("a is greater");
```

Figure 6.8: generate activity diagram.

- User can save code with the main class or can generate only behavioral code.

9. Save code generated from activity diagram.

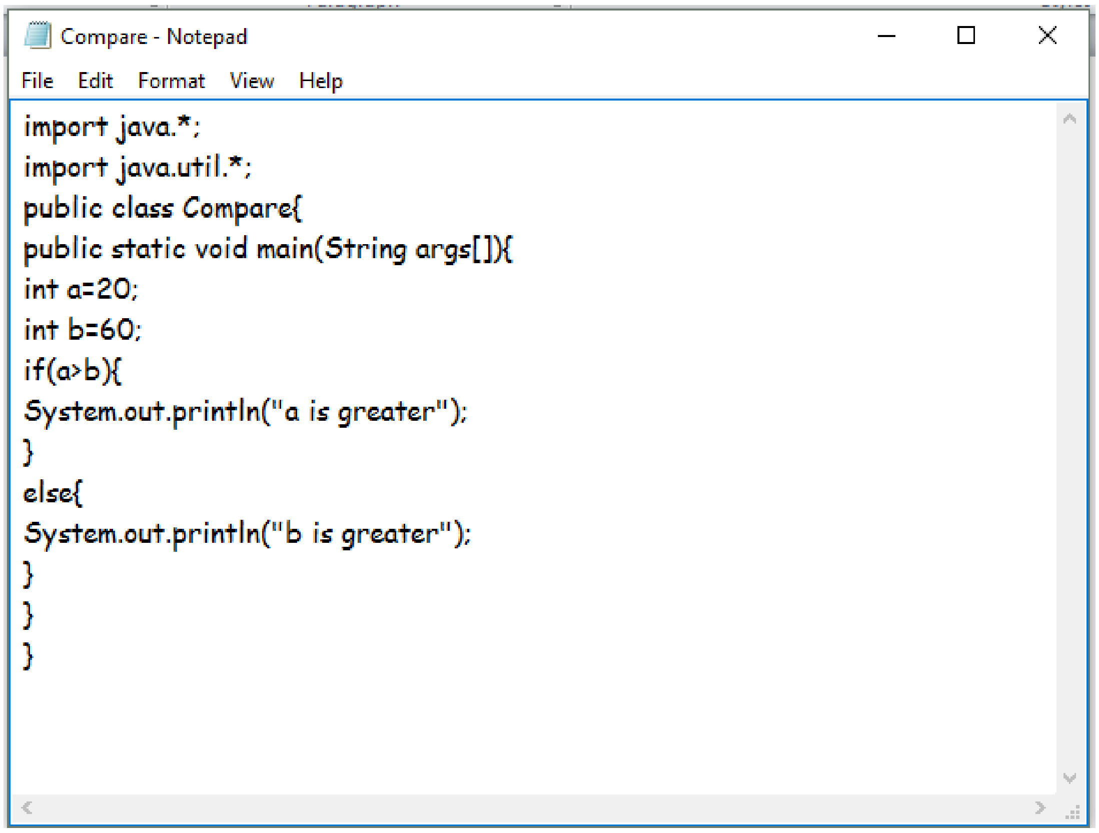


Name	Date modified	Type
Android Seminar	28-02-2017 14:04	File folder
AWT SOFTWARE	06-11-2016 09:00	File folder
group5	07-04-2017 14:53	File folder
MCA SEMESTER V	15-12-2016 11:10	File folder
NetAgent_10.2.434_KES_10.2.4.674	17-09-2016 00:25	File folder
New folder	07-03-2017 15:02	File folder
Semester V	15-12-2016 09:22	File folder
SNMR VHD	20-03-2017 11:44	File folder
SteamLibrary	02-03-2017 09:03	File folder
yogesh	06-11-2016 09:24	File folder

File name: Compare
Save as type: JAVA(*.java)

Figure 6.9: save code generated from activity diagram.

10. Generated code from activity diagram.

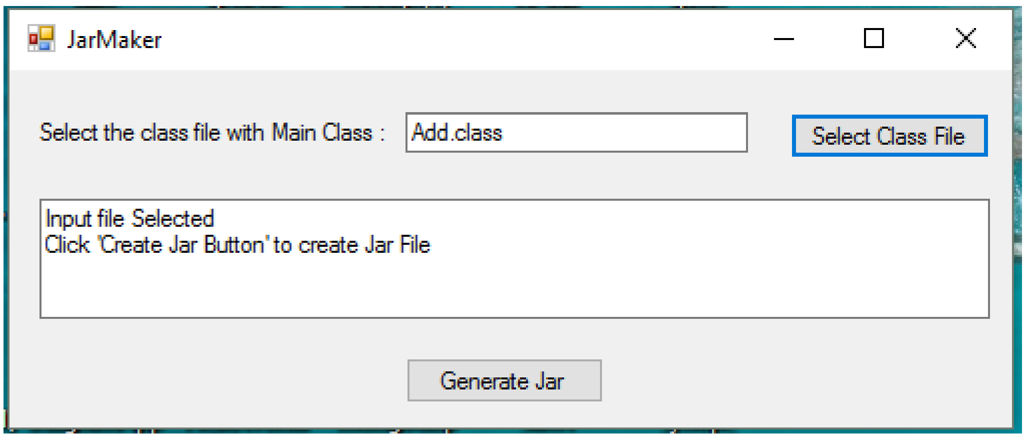


```
import java.*;
import java.util.*;
public class Compare{
public static void main(String args[]){
int a=20;
int b=60;
if(a>b){
System.out.println("a is greater");
}
else{
System.out.println("b is greater");
}
}
}
```

Figure 6.10: generated code from activity diagram.

- Code generated from activity diagram can be save at the user defined location.
- User can either save code with the main class or can save only behavioral code.
- In Fig. 6.10 behavioral code is stored with the class Compare.

11. Jar file maker.



Select the class file with Main Class : Add.class

Input file Selected
Click 'Create Jar Button' to create Jar File

Figure 6.11: jar file maker.

- Jar maker generates jar file of the class file defined by user.
- Generated jar file can be saved on the user defined location.
- In fig 6.11 Jarmaker generates jar file of the Add.class