# SOFTWARE QUALITY ASSURANCE & TESTING

# GRADUATE PROJECT REPORT

**TEAM MEMBERS: Mayank Purohit and Piyush Mantri**

**TOPIC: SATM (Simple automatic teller machine) using MVC architecture.**

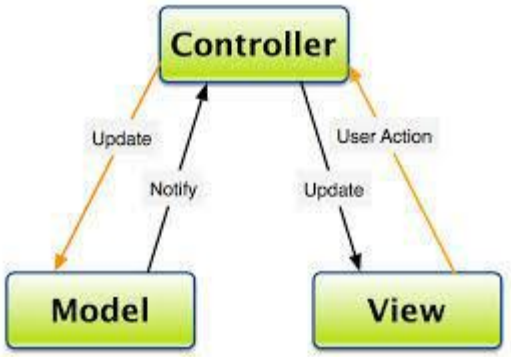
**OBJECTIVE:**
Designing and implementing a simple automatic teller machine using MVC (Model-View-Controller) Architecture and testing model, view and controller using Junit framework

**INTRODUCTION:**
Model-View-Controller Architecture:
The MVC pattern differentiates the modelling of a pattern, its presentation and the actions based on user input into three separate classes.

Model: A model class represents the information or data of an application and the rules to manipulate the data.

View: View represents the user interface of your application. Views handle the job by providing data to the web browser or other tool that is used to make requests from your application.

Controller: The controller provides the "flow" between models and views. Controllers are responsible for processing the incoming requests from the web browser, interrogating the models for data and passing that data on to the views for presentation.



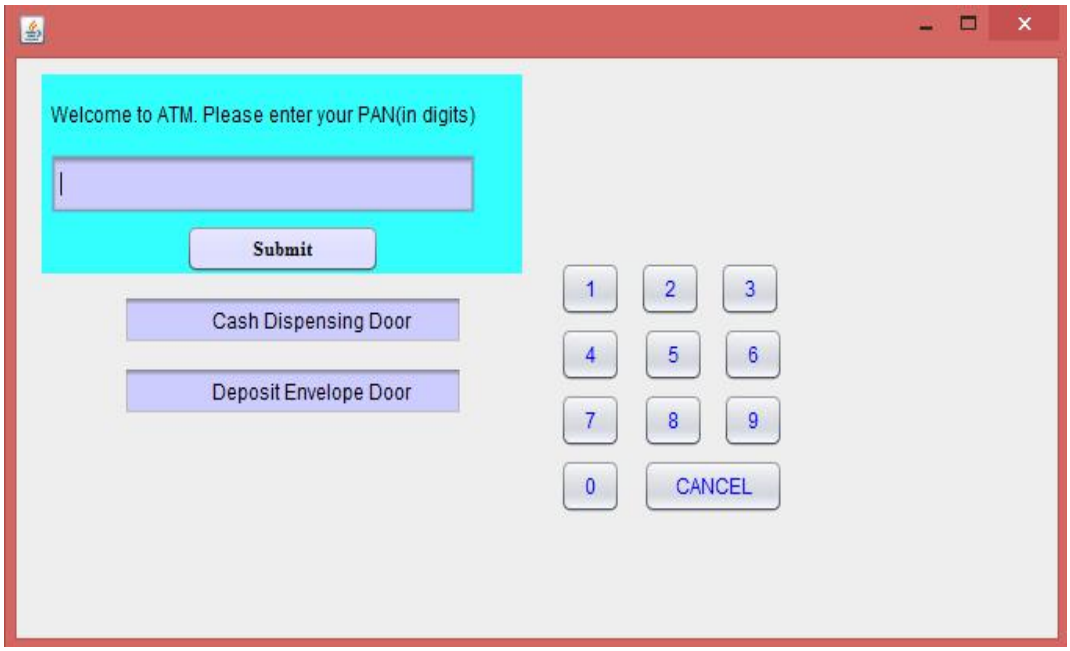
## SATM APPLICATION:

SATM is a simple automatic teller machine to communicate with the bank customers. The customer can perform any of three transaction types: deposit, withdrawals and balance inquires.

Customer can exit the system at any point during the transaction. MVC architecture will be used to develop the system. View part will be done by the GUI screens for interaction with the customer. Files will be used to handle the Model (data) part of the architecture. Controller will handle the transactions as well as the updation of model at times.

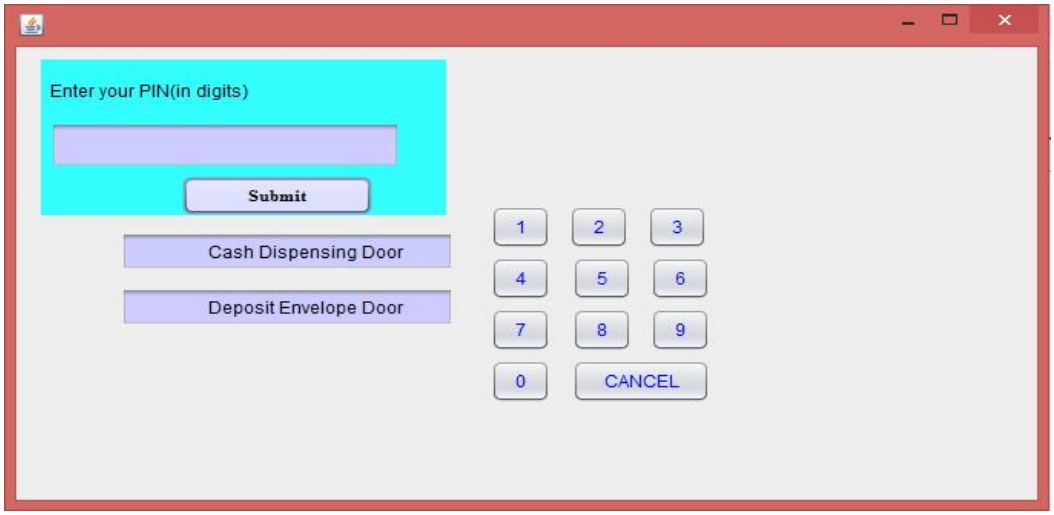
## IMPLEMENTATION OF SATM APPLICATION:

In below screen, customer is asked to enter his PAN.

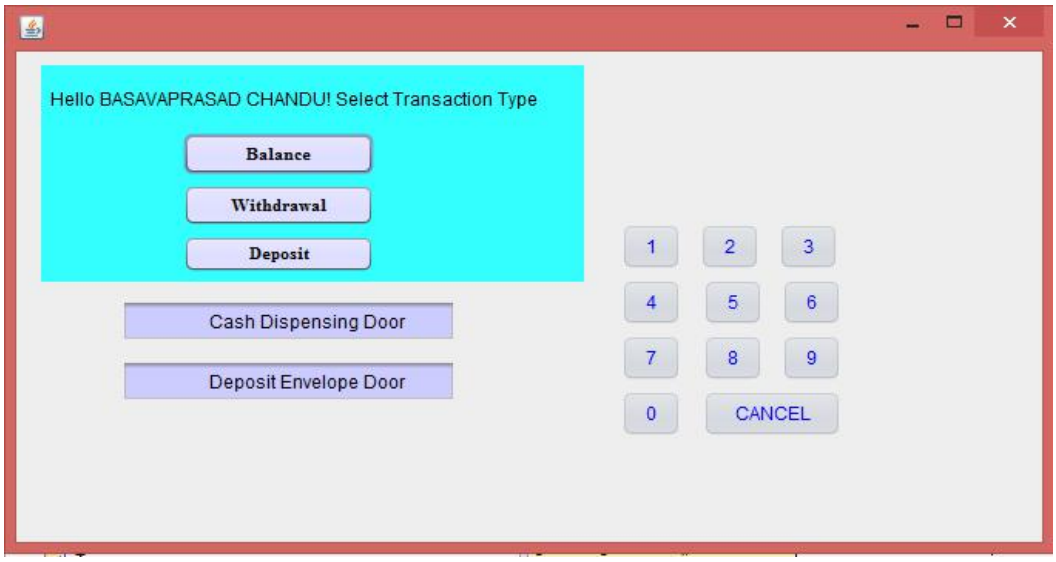



If the entered PAN is wrong, below screen will be displayed.

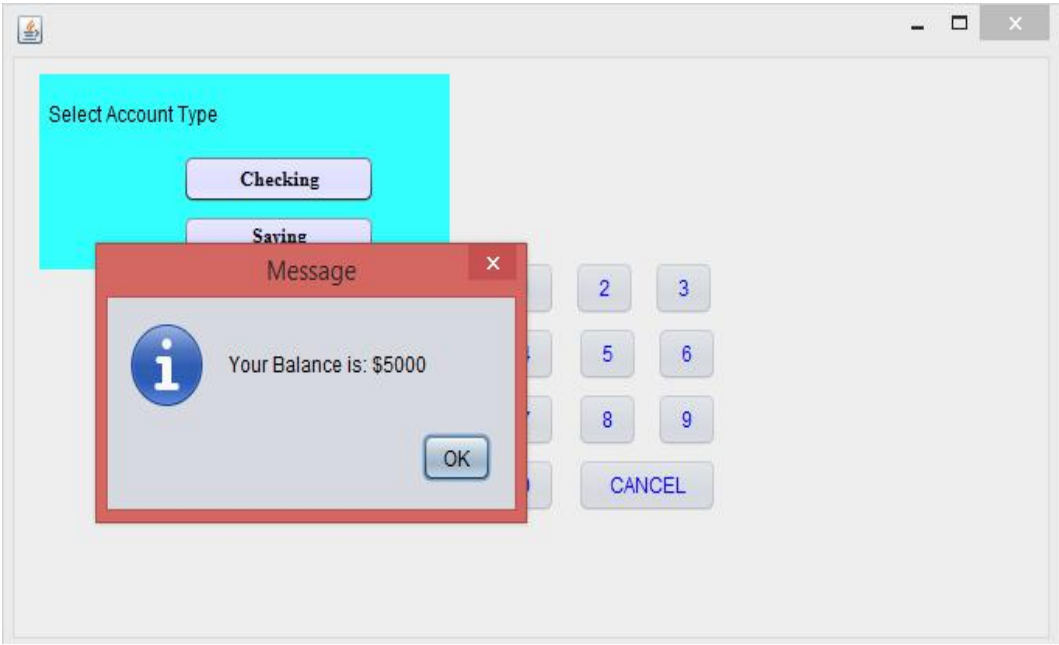
If the entered PAN is correct, it will take him/her to the next page.



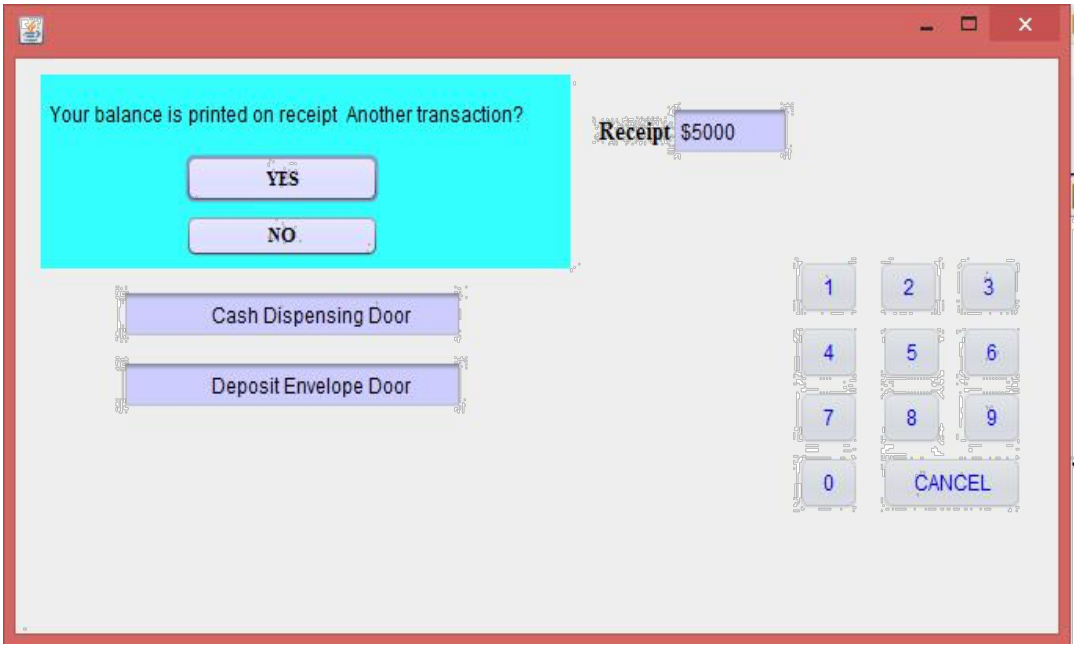

If entered PIN is correct he will be taken to the next screen.



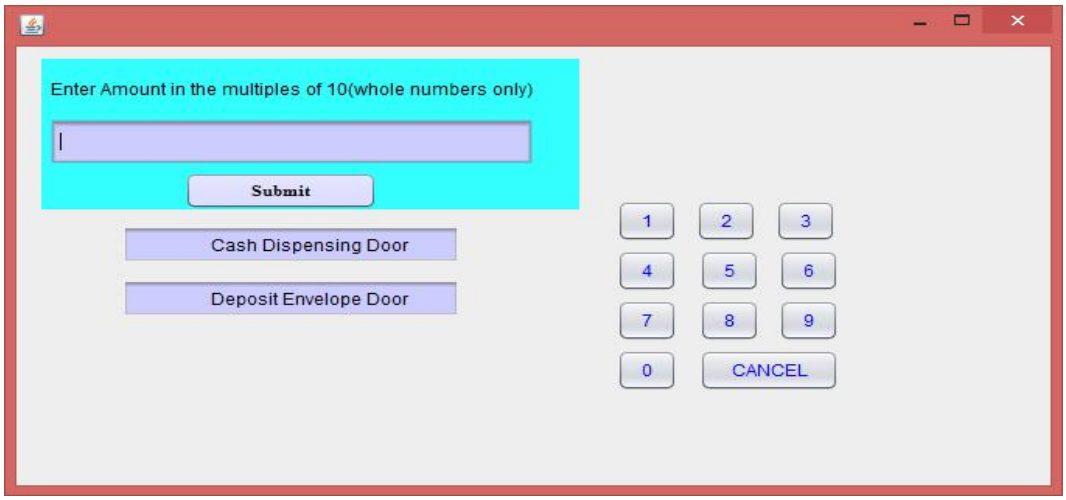
Whatever the account user selects, he will be shown balance of that account which can be seen in below screen.



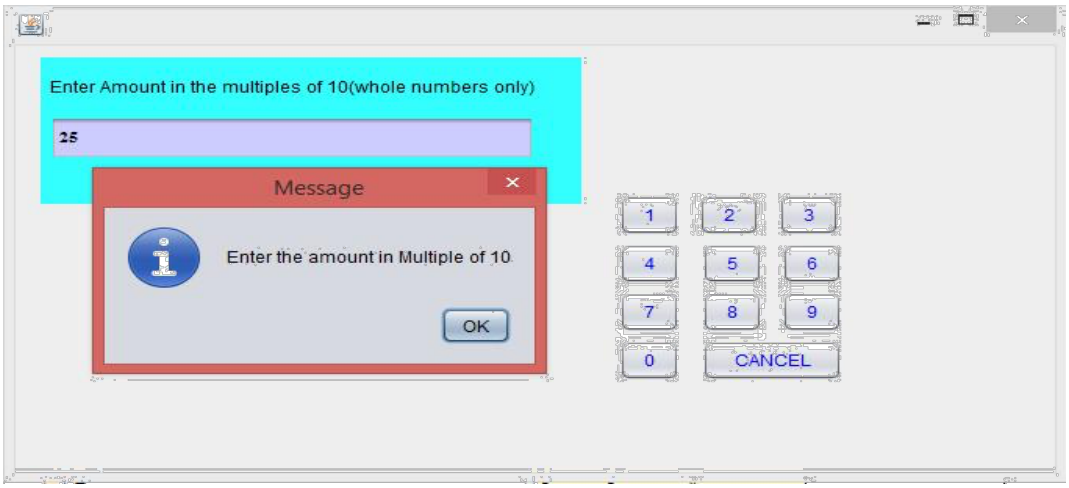

After, this customer will be asked that if he want to do another transaction.



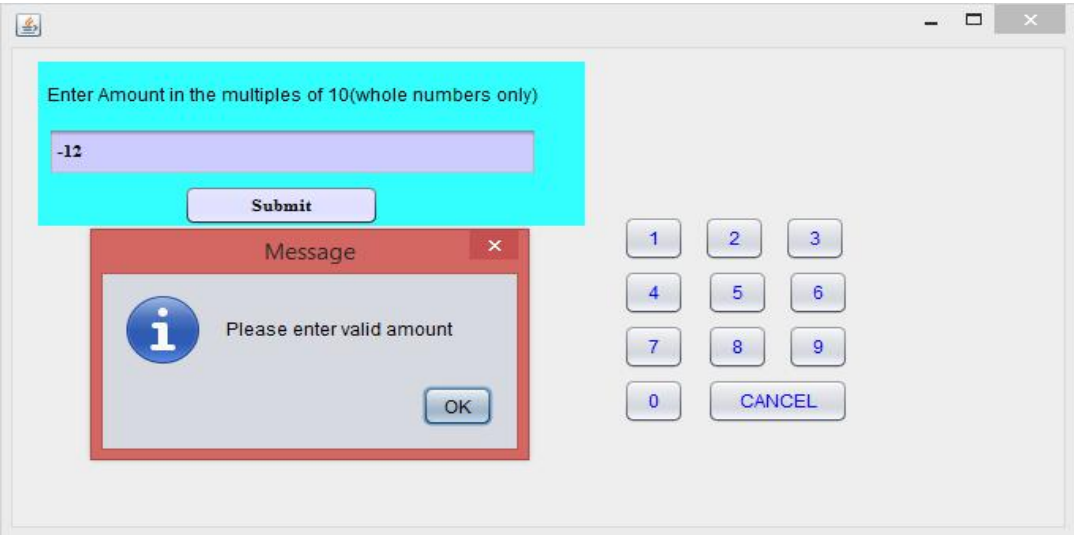
If he selects yes, customer will be shown screen where he will asked the transaction he/she wants to do, like balance, deposit and withdrawal. If customer selects withdrawal, following screen will be displayed.




If the entered amount is not in multiple of 10, following screen will show up.

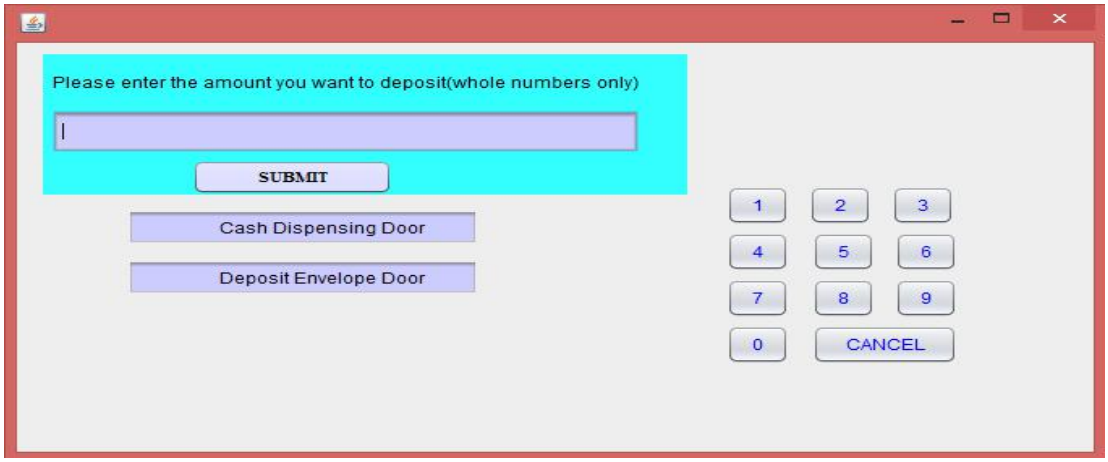


If the entered amount is invalid, following screen will come up.



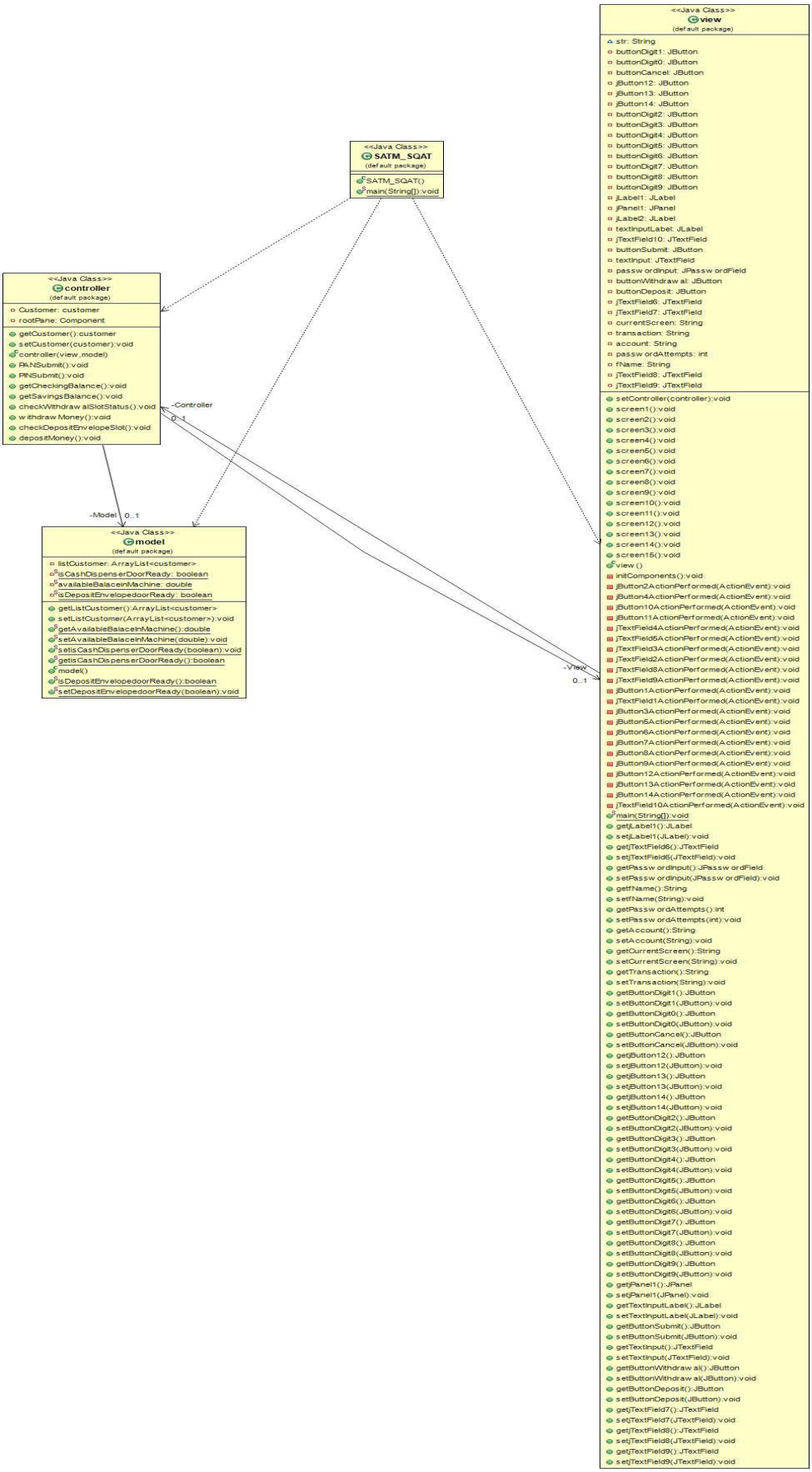

If he enters the correct amount, following screen will come up.

On Pressing done button, customer will be shown screen where he will be asked the transaction he/she wants to do, like balance, deposit and withdrawal. If customer selects balance, following screen will be displayed.




On entering correct amount, customer will be shown screen where he will asked if he want to do another transaction, if he says no. following screen will come.

# CLASS DIAGRAM OF MVC ARCHITECTURE:



The controller class acts as an interface between the view class and the model class. The model class performs the computations and then gives the result. The controller takes the result from the model class and passes it to the view class to display it on the GUI.

## TESTING OF SATM APPLICATION

Designing test cases and implementing the test cases is the primary function of this phase. Here, we test the SATM application through the test cases. Junit test drivers are used to implement them. Since, we have implemented the SATM application using MVC architecture, we can test entire application in units or components. We start the testing at unit level, then later for integration testing. This is later surpassed by object oriented system testing.

1. **Test cases for controller:**

Testing of controller is done using the model based testing strategy. We have different user scenarios and use cases. As discussed, the use case scenarios would be similar to that of model class and view class. However, the implementing setter and getter methods would change. The method that calculates the change will also get changed.

| **Test Case** | **Function** | **Input** | **Expected Output** |
|---|---|---|---|
| 1 | PANSubmit() | PAN = 0123456789 | Screen2 |
| 2 | PINSubmit() | PIN = 1990 | Screen5 |
| 3 | getCheckingBalance() | Account = Checking | Screen14 |
| 4 | getSavingBalance | Account = Savings | Screen14 |
| 5 | checkWithdrawalSlotStatus() | getCurrentScreen() | Screen7 |
| 6 | withdrwaMoneyChecking() | Account= Checking, Amount = 350 | Scree11 |
| 7 | withdrwaMoneyChecking() | Account= Checking, Amount = 35 | Screen7 |
| 8 | withdrwaMoneyChecking() | Account= Checking, Amount = 100000 | Screen9 |
| 9 | withdrwaMoneySaving() | Account= Saving, Amount = 350 | Screen11 |
| 10 | withdrwaMoneySaving() | Account= Saving, Amount = 35 | Screen7 |
| 11 | withdrwaMoneySaving() | Account= Saving, Amount = 100000 | Screen9 |
| 12 | checkDepositEnvelopeSlot() | getCurrentScreen() | Screen13 |

Java - SATMSQAT

File Edit Source Refactor Navigate Search Project Test Field Assist Menu X2 Run Window Hel

Package Explorer | JUnit

Finished after 4.652 seconds

Runs: 14/14 Errors: 0 Failures: 0

DriverController [Runner: JUnit 4] (3.966 s)

- testDepositMoneychecking (0.019 s)
- testDepositMoneysaving (0.080 s)
- testcheckDepositEnvelopeSlot (0.052 s)
- test2withdrawMoneychecking (1.706 s)
- testgetCheckingBalance (0.523 s)
- testcheckWithdrawalSlotStatus (0.051 s)
- test3withdrawMoneysaving (0.040 s)
- test2withdrawMoneysaving (0.540 s)
- testPANSubmit (0.038 s)
- test1withdrawMoneysaving (0.035 s)
- test1withdrawMoneychecking (0.030 s)
- testPINSubmit (0.041 s)

Failure Trace

controller.java | DriverContro... | vi

```
1  import static org.junit.Assert.*;
6
7
8  public class DriverController {
9      model model1 = new model();
10     view view1 =  new view();
11     customer cust = new customer("
12     customer cust1 = new customer(
13     controller control = new contr
14
15     @Test
16     public void testPANSubmit()
17         {
18             view1.getTextInput().s
19             control.PANSubmit();
20             String str = view1.get
21             assertEquals("screen2"
22         }
23     @Test
24     public void testPINSubmit()
25     {
26         control.setCustomer(cust1)
27         view1.getPasswordInput().s
28         control.PINSubmit();
29         String str1 = view1.getCur
30         assertEquals("screen5", st
31     }
32     @Test
33     public void testgetCheckingBal
34     {
35         control.setCustomer(cust1)
36         control.getCheckingBalance
37         String str2 = view1.getCur
38         assertEquals("screen14", s
39     }
40     @Test
41     public void testgetSavingBalan
```

2. **Test cases for model**:

We can design the test cases for "Model" class by testing the methods within the class along using the functionality of the model in the driver. The class "Model" has methods which get invoked when respective buttons are clicked. There are number of test cases to test the model class of the SATM application. Following are the test cases of Model class.

| **Test Case** | **Function** | **Input** | **Expected Output** |
|---|---|---|---|
| 1 | Model1() | getCustomerList().get(0) | PIYUSH MANTRI , 0123456789, 1990, 0, 0, 5000, 5000 |
| 2 | Model2() | getCustomerList().get(1) | MAYANK PUROHIT , 4802785779, 1991, 0, 0, 5000, 5000 |
| 3 | Model3() | getCustomerList().get(2) | JOHN CENA, 1234567890, 1989, 0, 0, 5000, 5000 |
| 4 | Model4() | getCustomerList().get(3) | DWAYNE JOHNSON, 9876543210, 1992, 0, 0, 5000, 5000 |
| 5 | getisCashDispenserDoorReadygetter() | getisCashDispenserDoorReady() | True |
| | | | |
| 6 | getisCashDispenserDoorReadysetter() | getisCashDispenserDoorReady() | True |
| | | | |
| 7 | getAvailableBalanceInMachine() | getAvailableBalanceInMachine() | 100000 |
| | | | |
| 8 | setAvailableBalanceInMachine() | getAvailableBalanceInMachine() | 100000 |
| | | | |
| 9 | setDepositEnvelopdoorReady() | isDepositEnvelopedoorReady() | True |
| | | | |

File Edit Source Refactor Navigate Search Project Test Field Assist Menu

Package Explorer | JUnit

Finished after 0.952 seconds

Runs: 10/10 Errors: 0 Failures: 0

- DriverModel [Runner: JUnit 4] (0.377 s)
  - testgetisCashDispenserDoorReadysetter (0.032 s)
  - testmodel1 (0.070 s)
  - testmodel2 (0.042 s)
  - testmodel3 (0.037 s)
  - testmodel4 (0.060 s)
  - testsetAvailableBalaceInMachine (0.036 s)
  - testgetAvailableBalaceInMachine (0.039 s)
  - testsetDepositEnvelopedoorReady (0.030 s)
  - testCustomer (0.030 s)
  - testgetisCashDispenserDoorReadygetter (0.001 s)

Failure Trace

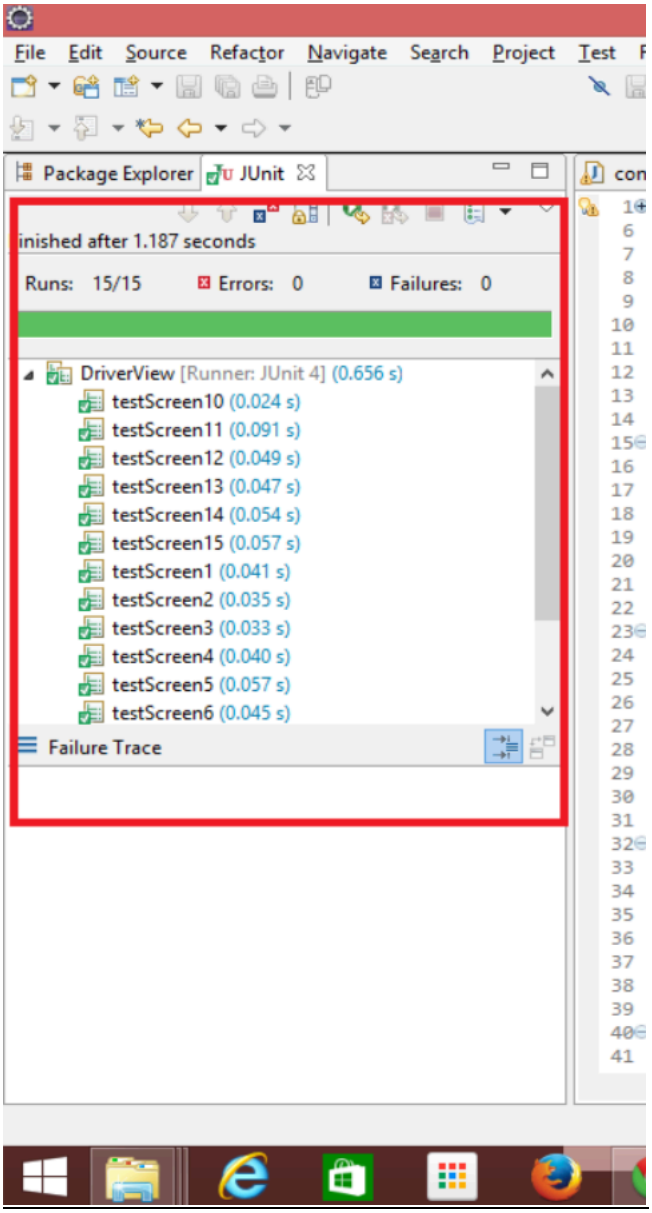
controller.java

```
1  import static
6
7
8  public class D
9      model mode
10     view view1
11     customer c
12     customer c
13     controller
14
15     @Test
16     public voi
17         {
18             vi
19             co
20             St
21             as
22         }
23     @Test
24     public voi
25     {
26         contro
27         view1.
28         contro
29         String
30         assert
31     }
32     @Test
33     public voi
34     {
35         contro
36         contro
37         String
38         assert
39     }
40     @Test
41     public voi
```

3. **Test cases for view:**

Testing of VIEW is also done using model based testing strategy. However, the implementing setter and getter methods would change. The method that calculates the change will also get changed.

| Test Case | Function | Input | Expected Output |
|---|---|---|---|
| 1 | Screen1() | getCurrentScreen() | currentScreen=1 |
| 2 | Screen2() | getCurrentScreen() | currentScreen=2 |
| 3 | Screen3() | getCurrentScreen() | currentScreen=3 |
| 4 | Screen4() | getCurrentScreen() | currentScreen=4 |
| 5 | Screen5() | getCurrentScreen() | currentScreen=5 |
| 6 | Screen6() | getCurrentScreen() | currentScreen=6 |
| 7 | Screen7() | getCurrentScreen() | currentScreen=7 |
| 8 | Screen8() | getCurrentScreen() | currentScreen=8 |
| 9 | Screen9() | getCurrentScreen() | currentScreen=9 |
| 10 | Screen10() | getCurrentScreen() | currentScreen=10 |
| 11 | Screen11() | getCurrentScreen() | currentScreen=11 |
| 12 | Screen12() | getCurrentScreen() | currentScreen=12 |
| 13 | Screen13() | getCurrentScreen() | currentScreen=13 |
| 14 | Screen14() | getCurrentScreen() | currentScreen=14 |
| 15 | Screen15() | getCurrentScreen() | currentScreen=15 |


File Edit Source Refactor Navigate Search Project Test
Package Explorer
JUnit
inished after 1.187 seconds
Runs: 15/15
Errors: 0
Failures: 0
DriverView [Runner: JUnit 4] (0.656 s)
testScreen10 (0.024 s)
testScreen11 (0.091 s)
testScreen12 (0.049 s)
testScreen13 (0.047 s)
testScreen14 (0.054 s)
testScreen15 (0.057 s)
testScreen1 (0.041 s)
testScreen2 (0.035 s)
testScreen3 (0.033 s)
testScreen4 (0.040 s)
testScreen5 (0.057 s)
testScreen6 (0.045 s)
Failure Trace

## CODE COVERAGE:

Code coverage is a measure used to describe the degree to which the source code of a program is tested by a particular test suite. EclEmma is used in eclipse to calculate the coverage. The test cases of my SATM application covers the 81% of code in which it covers around 90% of the controller and 93% of the model. This thing can be seen in the screenshot attached below.

| Element | Coverage | Covered Instructio... | Missed Instructions | Total Instructions |
|---|---|---|---|---|
| SATMSQAT | 80.6 % | 4,065 | 978 | 5,043 |
| src | 80.6 % | 4,065 | 978 | 5,043 |
| (default package) | 80.6 % | 4,065 | 978 | 5,043 |
| view.java | 75.6 % | 2,760 | 892 | 3,652 |
| controller.java | 89.5 % | 351 | 41 | 392 |
| SATM_SQAT.java | 0.0 % | 0 | 26 | 26 |
| customer.java | 83.6 % | 61 | 12 | 73 |
| model.java | 92.5 % | 86 | 7 | 93 |
| DriverController.java | 100.0 % | 337 | 0 | 337 |
| DriverModel.java | 100.0 % | 259 | 0 | 259 |
| DriverView.java | 100.0 % | 211 | 0 | 211 |

Ctrl Contrib (Bottom)