

**Project Report**

**Project:** Bank Management System

**Course:** Software Quality Assurance & Testing

**Course Code:** CSE427

**Semester:** Spring2019

**Submitted to-**

Shaikh Shawon Arefin Shimon

**Faculty Initial-** SAS3

**Submitted by-**

Group – 19

Members- Farhan Israk Yen – 1520725042

**GitHub Link:** https://github.com/nsuspring2019cse427/Group19

**Project Description**

By using this project, admin will be able to create new bank account, add money or deposit money, check balance of the accountants, check number of account holders, check percentage of interest and update it, search account for info, check for loan availability, give loan, withdrawing balance if account-holder wants and successful transaction of money. It will make a bank easier to perform these operations to save info rather than saving it manually.

It is always difficult to keep record of account-holders information manually as there might be huge amount of mistake by a person. So in order to keep track of such information, we can use a system where all the information related to accounts of the account-holders of a bank will be available. These information will include the number of account holders, their deposited money, transaction amount, amount of interests applied, withdrawal money amount etc. Thus the user will be able to keep record of such important things easily by using this application. So the time will be saved, it will be easier to find necessary info regarding accounts and keep these info safe.

User will help the person to open a bank account by asking him/her for his details like, name, phone no etc. Then user will input the name and number .Then it will ask to input the total account holder. After that it will ask to input the number of account holders name, phone no. and balance of their account. Also it will ask to add interest percentage. Then the user will be able to search account number in order to withdraw or to deposit money in his/her account. Also it will provide the information of number of transactions done in a particular account. We will use **Mockito**, **Hamcrest** and **JUnit** framework in order to test whole project. **Mockito** is a mocking framework that tastes really good. It lets you write beautiful tests with a clean & simple API. **Hamcrest** is a framework for writing matcher objects allowing ‘match’ rules to be defined declaratively. On other hand, **JUnit** is a simple framework to write repeatable tests. There are a number of situations where matchers are invaluble, such as UI validation, or data filtering, but it is in the area of writing flexible tests that matchers are most commonly used. This tutorial shows you how to use **Hamcrest** for unit testing.

**Input Space Partition**

For AccountHolder class-

|  |  |  |
| --- | --- | --- |
| Method | accountNumber>0 | accountNumber<=0 |
| setAccountNumber() | this.accountNumber = accountNumber | Wrong Input |

|  |  |  |
| --- | --- | --- |
| Method | accountBalance>0 | accountBalance <=0 |
| setAccountBalance() | this.accountBalance = accountBalance | Wrong Input |

**For Accountant class-**

|  |  |  |
| --- | --- | --- |
| Method | accountantId>0 | accountantId<=0 |
| setAccountantId | this.accountantId = accountantId | Wrong Input |

**For ForeignAccountHolder class-**

|  |  |  |
| --- | --- | --- |
| Method | interestPercentage>0 | interestPercentage<=0 |
| recieveInterest | continue | Wrong Input |

**For LocalAccountHolder class-**

|  |  |  |
| --- | --- | --- |
| Method | loanAmount>0 | loanAmount<=0 |
| recieveLoan | this.loanBalance = loanAmount | Wrong Input |

**For Person class-**

|  |  |  |  |
| --- | --- | --- | --- |
| Method | phone.length()==11 | phone.length()<11 | phone.length()>11 |
| setPhone | this.loanBalance = loanAmount | Wrong Input | Wrong Input |

|  |  |  |
| --- | --- | --- |
| Method | !NewName.isEmpty() | NewName.isEmpty() |
| inputName | name= new String(NewName.toCharArray()) | Wrong Input |

**For Transaction class-**

|  |  |  |
| --- | --- | --- |
| Method | transactionId>0 | transactionId<=0 |
| setTransactionId | this.transactionId = transactionId | Wrong Input |

|  |  |  |
| --- | --- | --- |
| Method | accountNumber>0 | accountNumber <=0 |
| setAccountNumber | this.accountNumber = accountNumber | Wrong Input |

|  |  |  |
| --- | --- | --- |
| Method | accountantId>0 | accountantId<=0 |
| setAccountantId | this.accountantId = accountantId | Wrong Input |

|  |  |  |
| --- | --- | --- |
| Method | previousBalance>0 | previousBalance<=0 |
| setPreviousBalance | this.previousBalance= previousBalance | Wrong Input |

|  |  |  |
| --- | --- | --- |
| Method | transactionBalance>0 | transactionBalance<=0 |
| setTransactionBalance | this.transactionBalance = transactionBalance | Wrong Input |

|  |  |  |
| --- | --- | --- |
| Method | accCheck>0 | accCheck<=0 |
| inputPreviousBalance | this.previousBalance= previousBalance | Wrong Input |

Graph Coverage for AccountHolder class

Balance>0 Balane<=0

this.accountBalance=balance showMessageDialogue(Wrong input)

accountNumber>0 accountNumber<=0

this.accountNumber=accountNumber showMessageDialogue(Wrong input)

accountBalance>0 accountBalance<=0

this.accountBalance=accountBalance showMessageDialogue(Wrong input)

Graph Coverage for Accountant class

accountId>0 accountId<=0

this. accountId= accountId showMessageDialogue(Wrong input)

Graph Coverage for Accountant class

Counter=0

Counter<2 Counter>=2

Accountrants[counter]:inputName()

Accountrants[counter]:inputPhone()

Counter=counter+1

Counter=0

Counter<accountHolder Counter>=accountHolder

user[counter]:inputName()

user[counter]:inputPhone()

user[counter]:inputAccountBalance()

Counter=counter+1

Counter=0

Counter<accountHolder Counter>=accountHolder

users[counter]:printAccountNumber()

users[counter]:printName()

users[counter]:printPhone()

users[counter]:printAccountBalance()

Counter=counter+1

interestInput= new interestInput()

counter=0

Counter<accountholder Counter>=accountHolder

users[counter].receiveInterest(interestInput)

counter=counter+1

Counter=0

Counter<accountHolder Counter>=accountHolder

users[counter]:printAccountNumber()

users[counter]:printAccountBalance()

users[counter]:printInterestBalance()

Counter=counter+1

Counter=0

Counter<accountHolder Counter>=accountHolder

userSearch.setAccountNumber

Counter=counter+1

Counter=0

Counter<2 Counter>=2

userSearch.setAccountantId

Counter=counter+1

Count=0

Count>=accountholder Count<accountHolder

Counter=

userSearch.searchAccountnumber

showMessageDialogue(Wrong input)

counter=-1 counter!=-1

transaction[counter].setAccountID

transaction[counter].

setPreviousBalance

showMessageDialogue(Wrong input) transaction[counter].

inputIsWithdraw

transaction[counter].getIsWithdraw!=true transaction[counter].

getIsWithdraw!=true

users[counter].deposit(transactions[counter]) transaction[counter].inputTranasactionBalance

users[counter].getTransactionBalance

count=count+1

Counter=0

Counter<accountHolder counter>=accountHolder

searchResult[counter]!=-1 searchResult==-1

users[counter].printAccountNumber()

transactions[counter].printTransactionID

transactions[counter].printAccountID

transactions[counter].printPreviousBalance

users[counter].printAccountBalance

counter=counter+1

Count=accountSearch.searchAccountNumber

Count=-1 count!=-1

countTransactions=0

counter=0

counter<accountHolder counter>=accountHolder

newAccountant!=getAccountID newAccountant==getAccountID

countTransaction=countTransaction+1

counter=counter+1

countTransaction==0 countTransaction!=0

counter>=accountHolder counter<accountHolder

newAccountant==

getAccountantID

newAccountant!=getAccountId

users[counter].printAccountNumber

transaction[counter].printTransID

transaction[counter].printIsWithdraw

users[counter].printAccountBalance

counter=counter+1

Counts=0

Counts>=accountBalance counts<accountHolder

Counter==-1 counter=userSeach.searchAccountNumber

Counter!=-1

Hotkey=Integer.parseInt(newHotkey)

Hotkey!=1 Hotkey==1

Users[counter].receiveLoan

Counts=counts+1

Users[counter].receiveLoan(balance)

Counter<accountHolder counter>=accountHolder

searchResult[counter]!=-1 searchResult[counter]==-1

counter=counter+1

Graph Coverage for ForeignAccountHolder class

maxWithdrawBalance=500

totalWithdrawBalance=0

withdrawBalance>MaxWithdrawBalance withdrawBalance<=maxWithdrawBalance

totalWithdrawAmount > maxwithdraw totalWithdrawAmount<=maxwithdraw

withdrawBalance<=accountBalance withdrawBalance<=accountBalance

accountBalance=accountBalance

.withdrawAccountBalance

totalWithdrawAmount=totalWith drawAmount+withdrawBalance

maxLoanAmount<requestLoanAmount maxLoanAmount>=requestedLoanAmount

loanBalance=requestedLoanAmount

interestPercentage<=0 interestPercentage>0

interestBalance=(accBalance\*interestPercent)/100

accountBalance=accountBalance+interestBalance

Graph Coverage for LocalAccountHolder class

withdrawBalance>accountBalance withdrawBalance<=accountBalance

withdrawBalance<0 withdrawBalance>=0

accountBalance=accountBalance -

withdrawBalance

loanAmount<=0 loanAmount>0

this.loanBalance=loanAmount

Graph Coverage for Person class

newPhone.length!=11 newPhone.length==11

this.phone=newPhone

Graph Coverage for Transaction class

transactionID<=0 transactionID>0

this.transactionId=transactionId

accountNumber<=0 accountNumber>0

this.accountNumber=accountNumber

accountantId<=0 accountantId>0

this.accountId=accountId

previousBalance<=0 previousBalance>0

this.previousBalance=previousBalance

newIsWithdraw.equals(“true”)||equals(“false”)

this.isWithdraw=newIsWithdraw

accCheck=newPreviousBalance

accCheck<=0 accCheck>0

this.previousBalance=accCheck

accCheck=newAccountNumber

accCheck<=0 accCheck>0

this.accountNumber=accCheck

accCheck=newAccountID

accCheck<=0 accCheck>0

this.accountId=accCheck