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

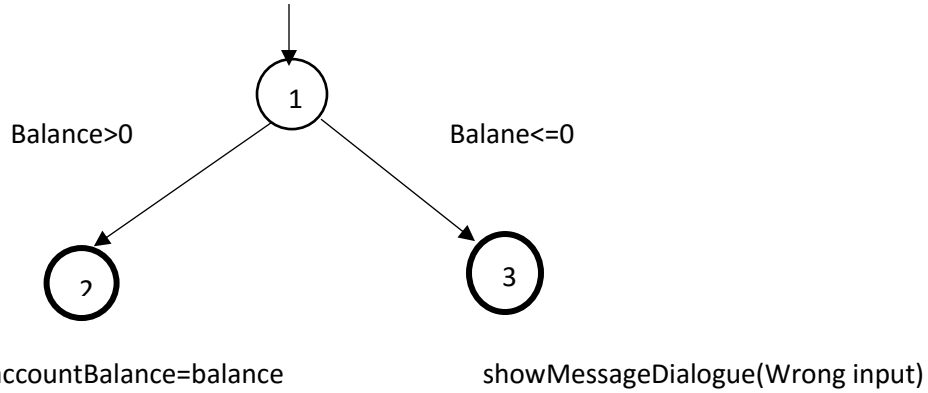
Method	accountantId>0	accountantId<=0
setAccountantId	this.accountantId = accountantId	Wrong Input

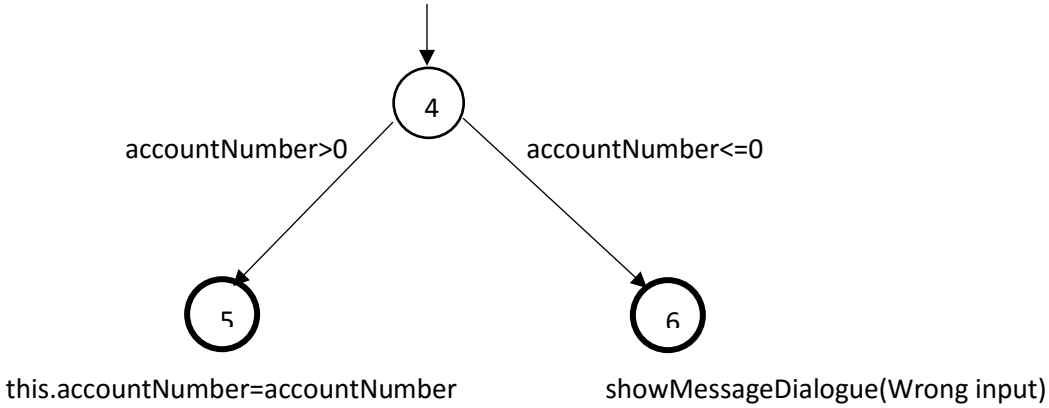
Method	previousBalance>0	previousBalance<=0
setPreviousBalance	this.previousBalance= previousBalance	Wrong Input

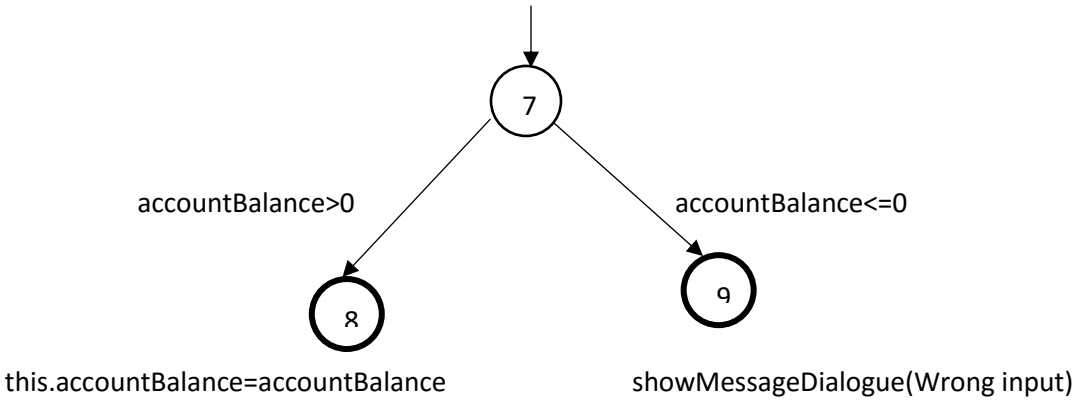
Method	transactionBalance>0	transactionBalance<=0
setTransactionBalance	this.transactionBalance = transactionBalance	Wrong Input

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

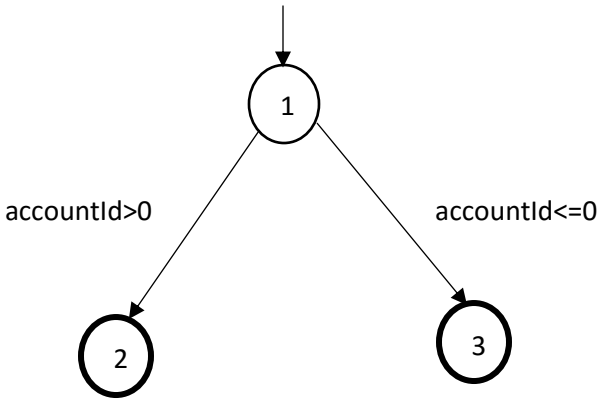
Graph Coverage for AccountHolder class







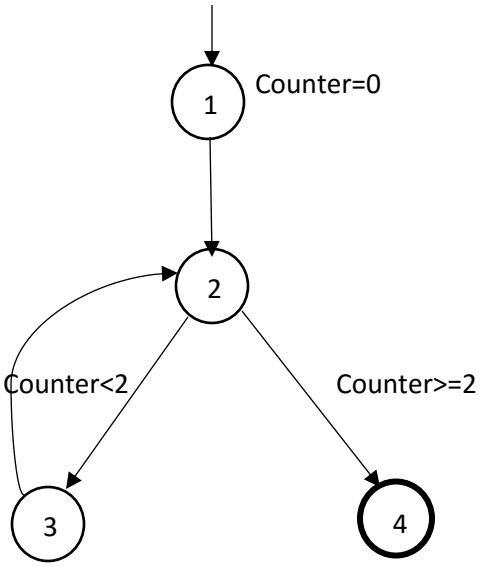
Graph Coverage for Accountant class



this. accountId= accountId

showMessageDialogue(Wrong input)

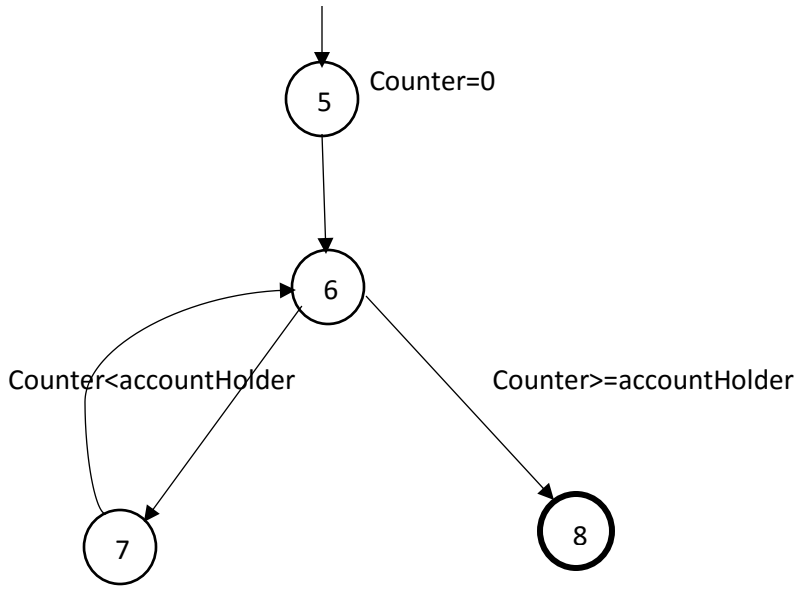
Graph Coverage for Accountant class



Accountrants[counter]:inputName()

Accountrants[counter]:inputPhone()

Counter=counter+1

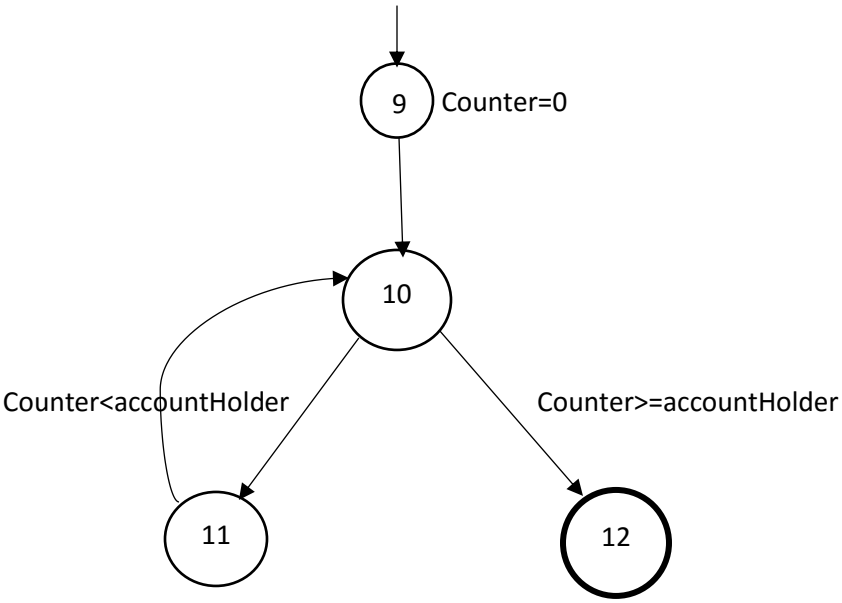


user[counter]:inputName()

user[counter]:inputPhone()

user[counter]:inputAccountBalance()

Counter=counter+1



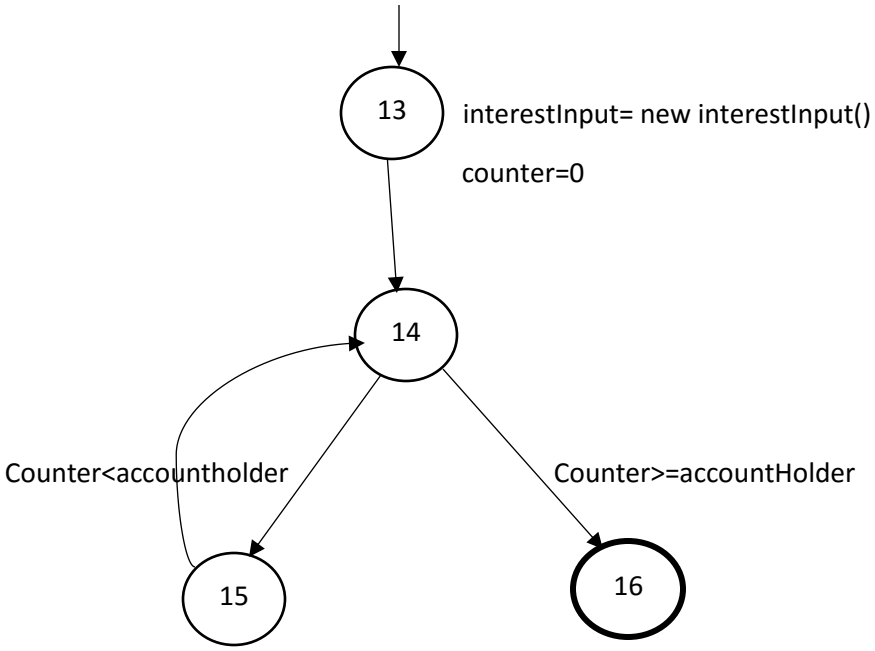
users[counter]:printAccountNumber()

users[counter]:printName()

users[counter]:printPhone()

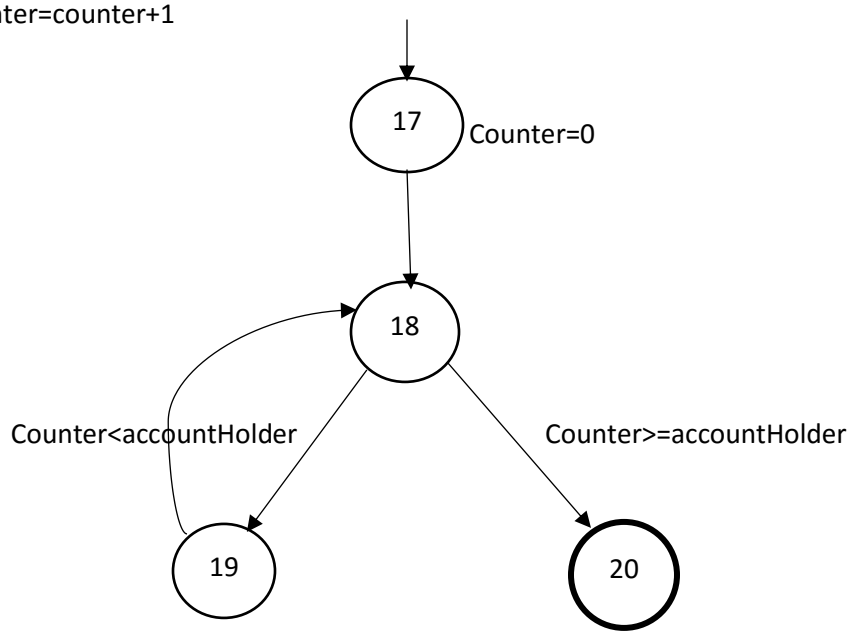
users[counter]:printAccountBalance()

Counter=counter+1



users[counter].receiveInterest(interestInput)

counter=counter+1

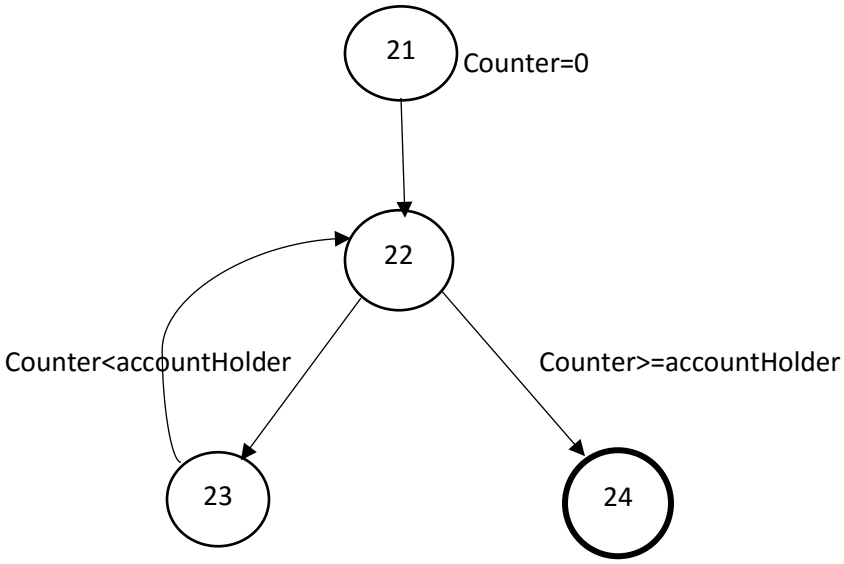


users[counter]:printAccountNumber()

users[counter]:printAccountBalance()

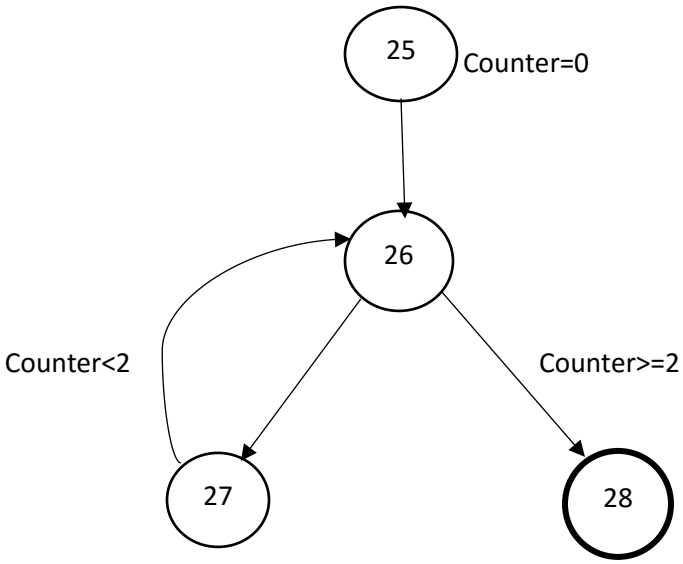
users[counter]:printInterestBalance()

Counter=counter+1



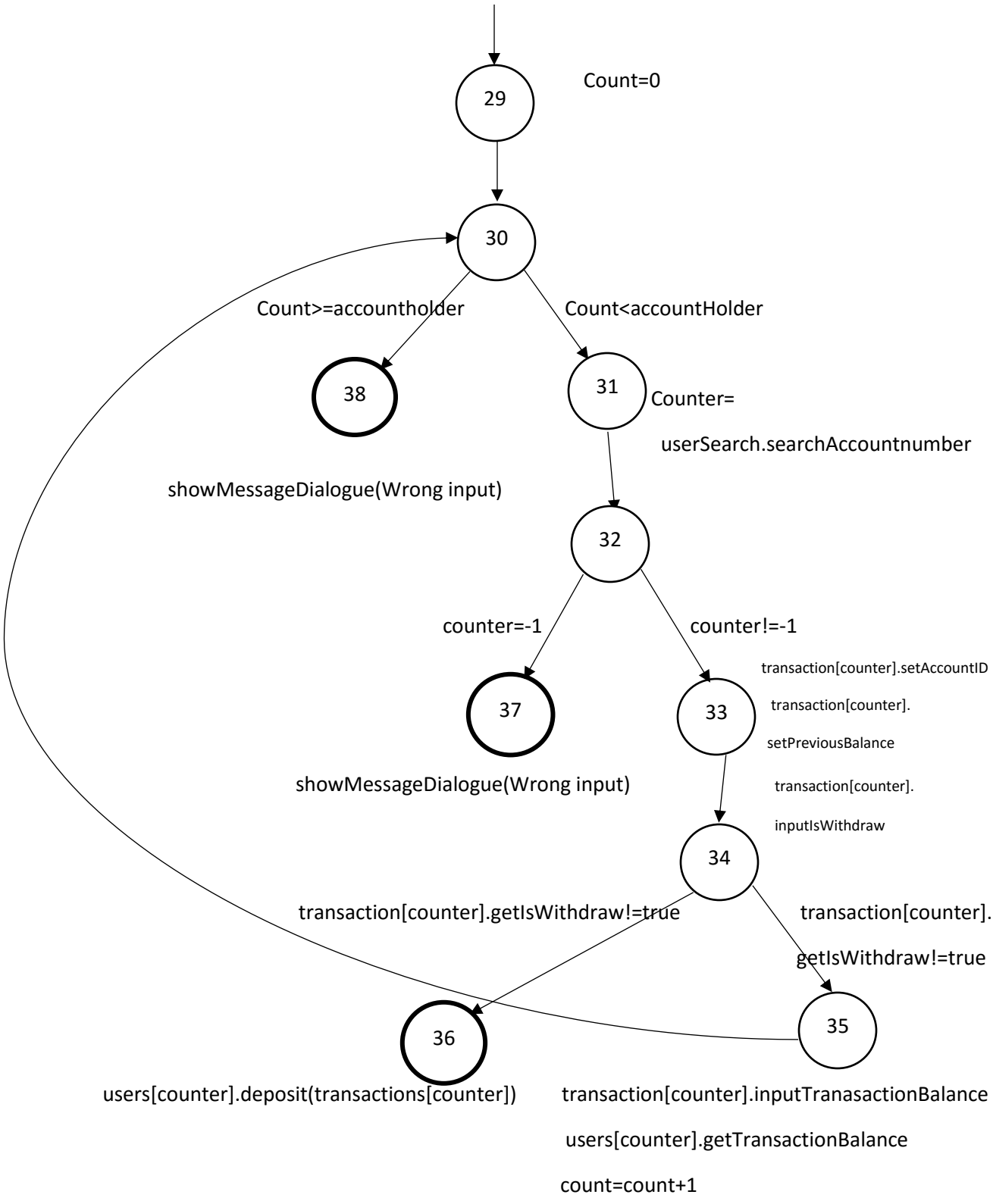
userSearch.setAccountNumber

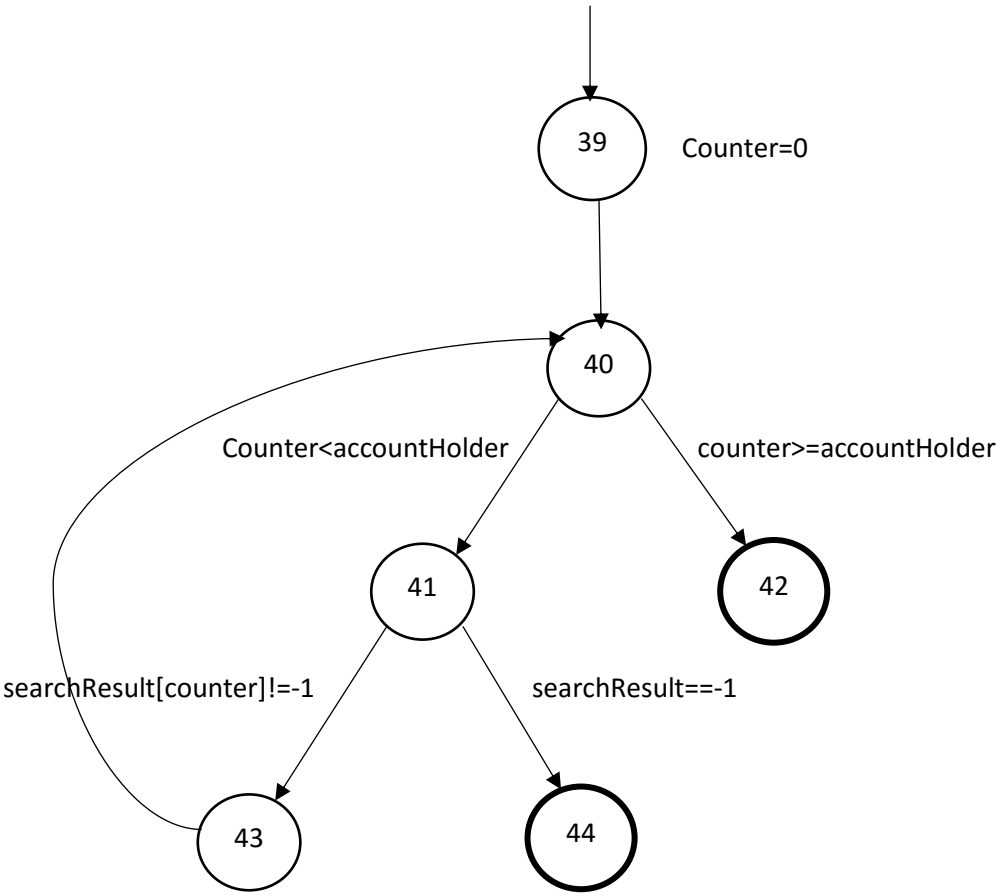
Counter=counter+1



userSearch.setAccountantId

Counter=counter+1


29
Count=0
30
Count>=accountholder
Count<accountHolder
38
31
Counter=
userSearch.searchAccountnumber
showMessageDialogue(Wrong input)
32
counter=-1
counter!=-1
37
33
transaction[counter].setAccountID
transaction[counter].
setPreviousBalance
showMessageDialogue(Wrong input)
transaction[counter].
inputIsWithdraw
34
transaction[counter].getIsWithdraw!=true
transaction[counter].
getIsWithdraw!=true
36
35
users[counter].deposit(transactions[counter])
transaction[counter].inputTranasactionBalance
users[counter].getTransactionBalance
count=count+1



users[counter].printAccountNumber()

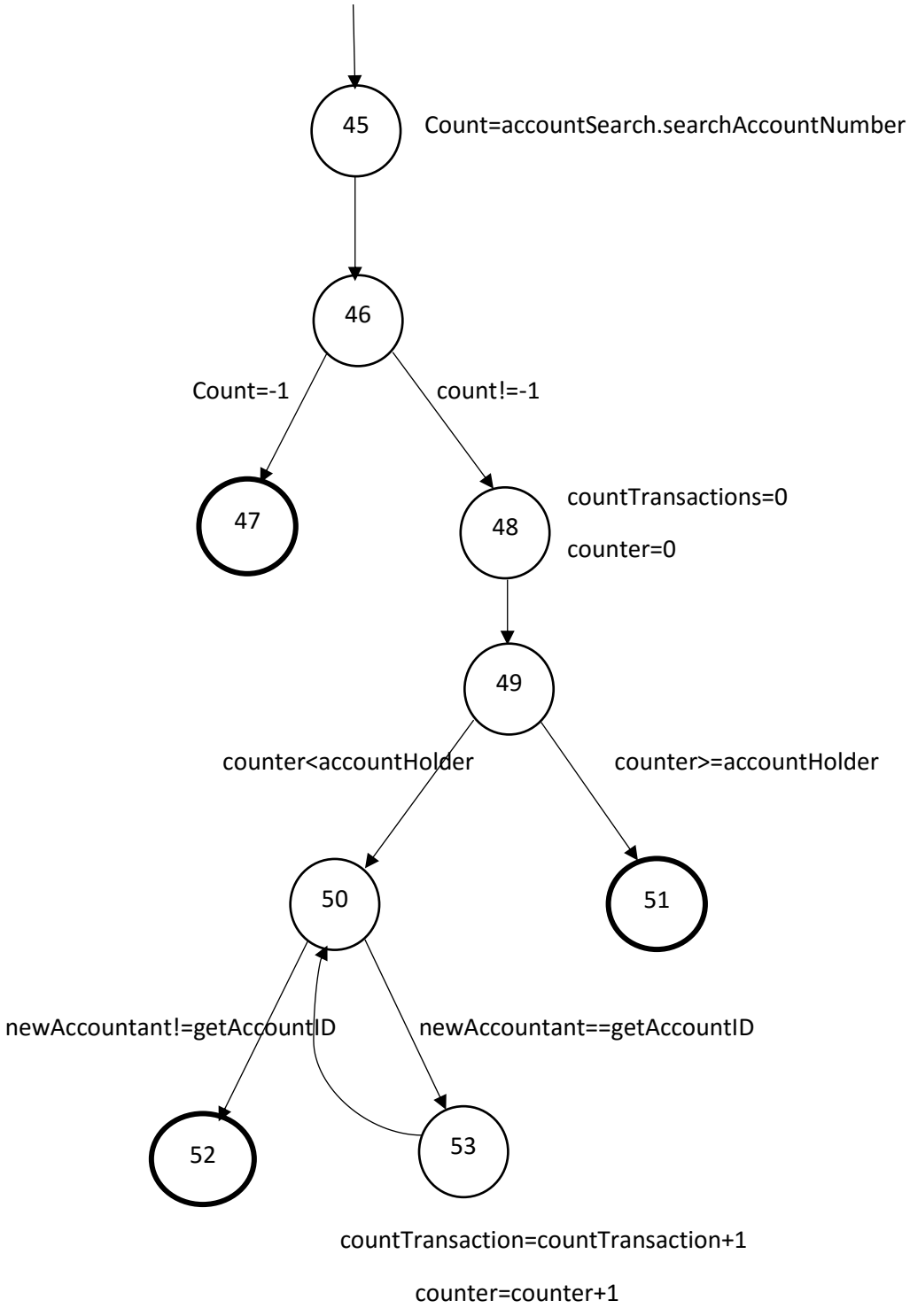
transactions[counter].printTransactionID

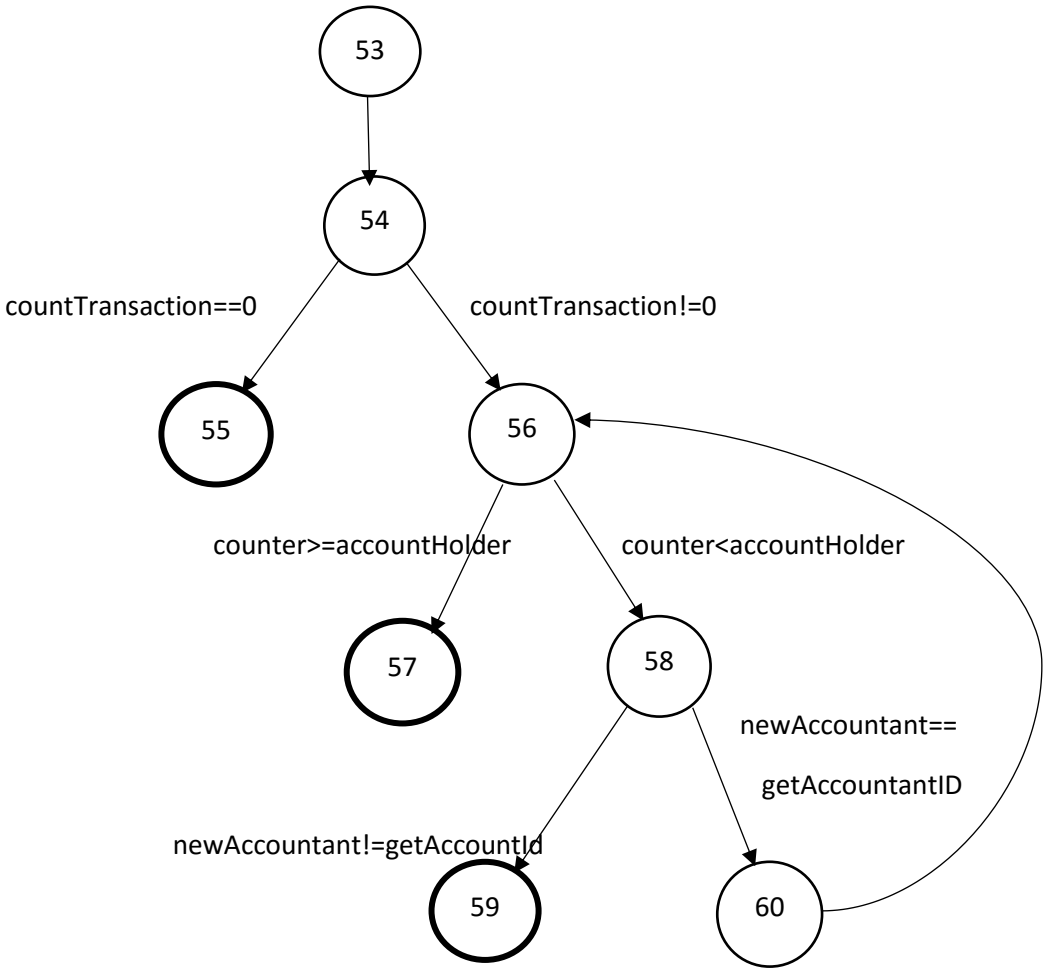
transactions[counter].printAccountID

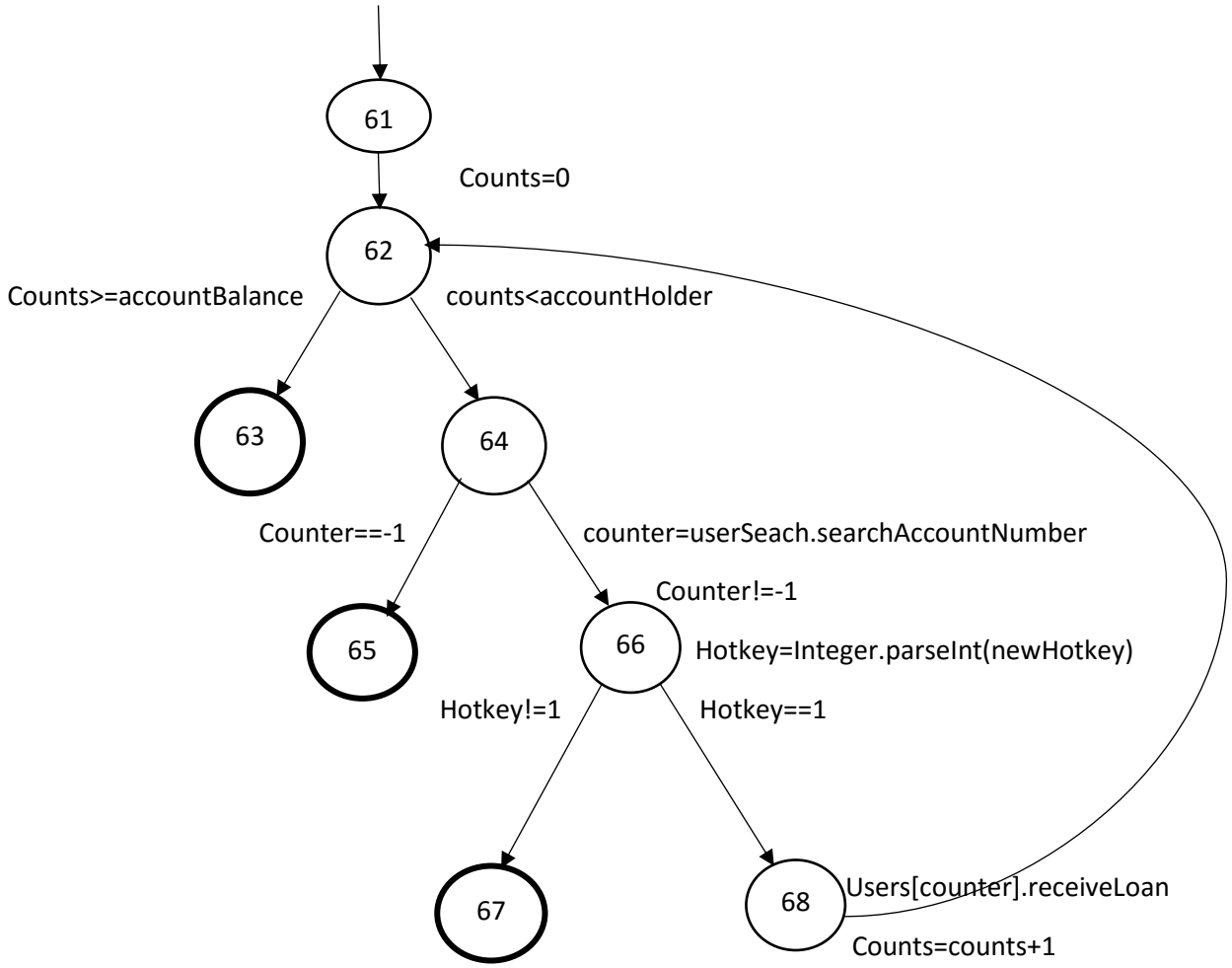
transactions[counter].printPreviousBalance

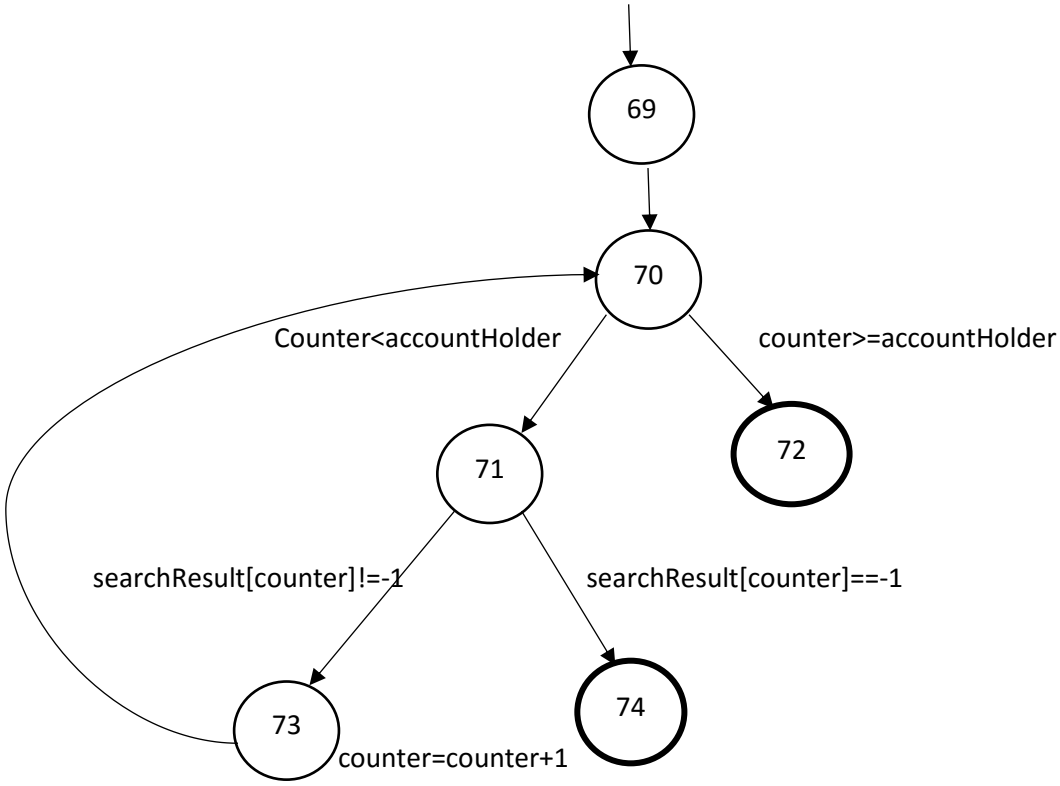
users[counter].printAccountBalance

counter=counter+1

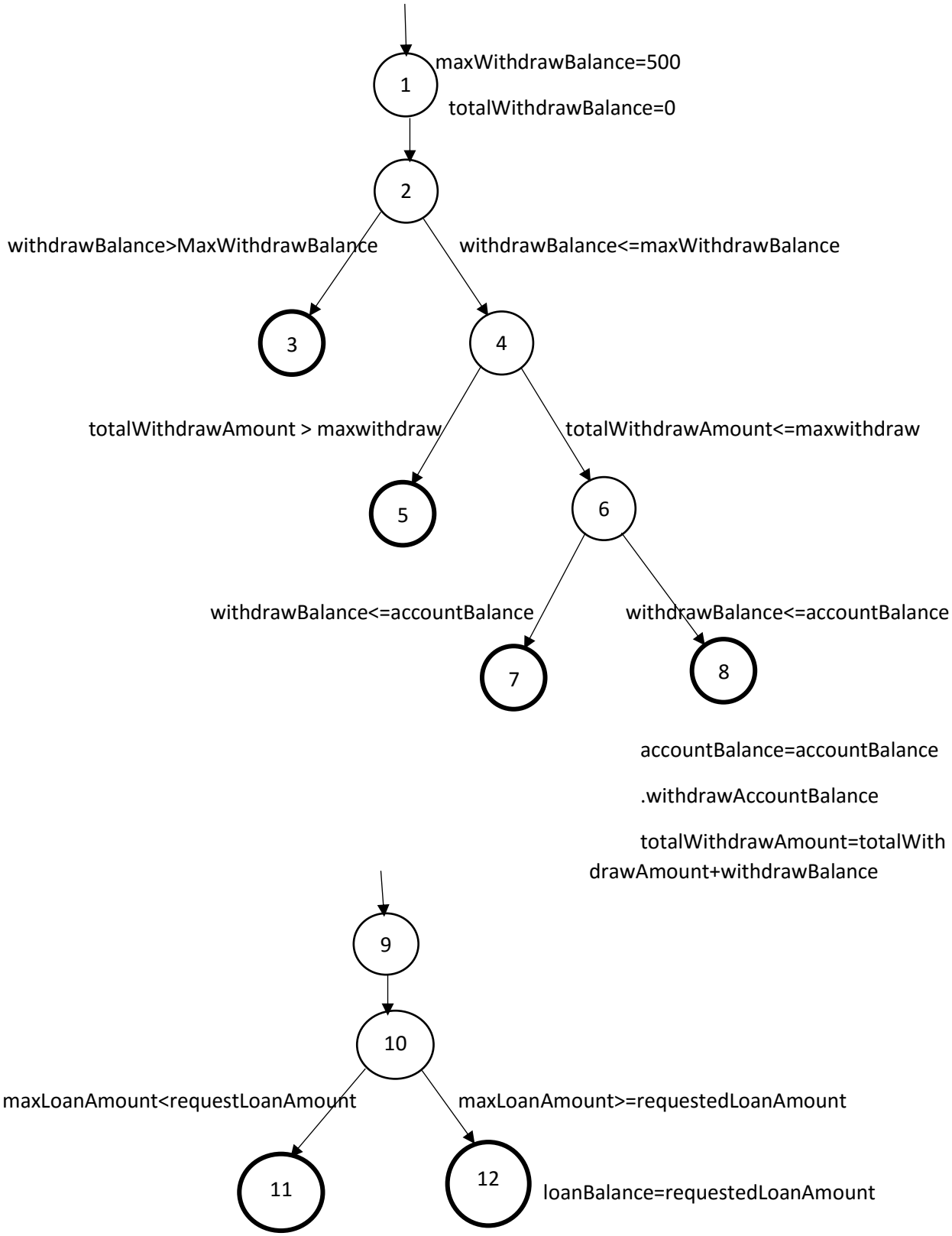

45
Count=accountSearch.searchAccountNumber
46
Count=-1
count!=-1
47
48
countTransactions=0
counter=0
49
counter<accountHolder
counter>=accountHolder
50
51
newAccountant!=getAccountID
newAccountant==getAccountID
52
53
countTransaction=countTransaction+1
counter=counter+1

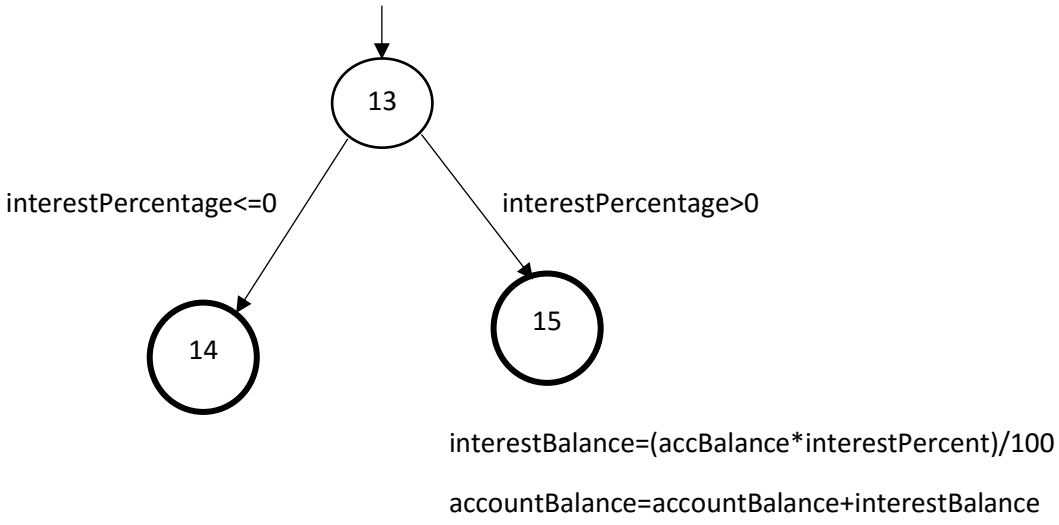

53
54
countTransaction==0
countTransaction!=0
55
56
counter>=accountHolder
counter<accountHolder
57
58
newAccountant==
getAccountantID
newAccountant!=getAccountId
59
60
users[counter].printAccountNumber
transaction[counter].printTransID
transaction[counter].printIsWithdraw
users[counter].printAccountBalance
counter=counter+1


61
Counts=0
62
Counts>=accountBalance
counts<accountHolder
63
64
Counter==-1
counter=userSeach.searchAccountNumber
Counter!=-1
65
66
Hotkey=Integer.parseInt(newHotkey)
Hotkey!=1
Hotkey==1
67
68
Users[counter].receiveLoan
Counts=counts+1
Users[counter].receiveLoan(balance)

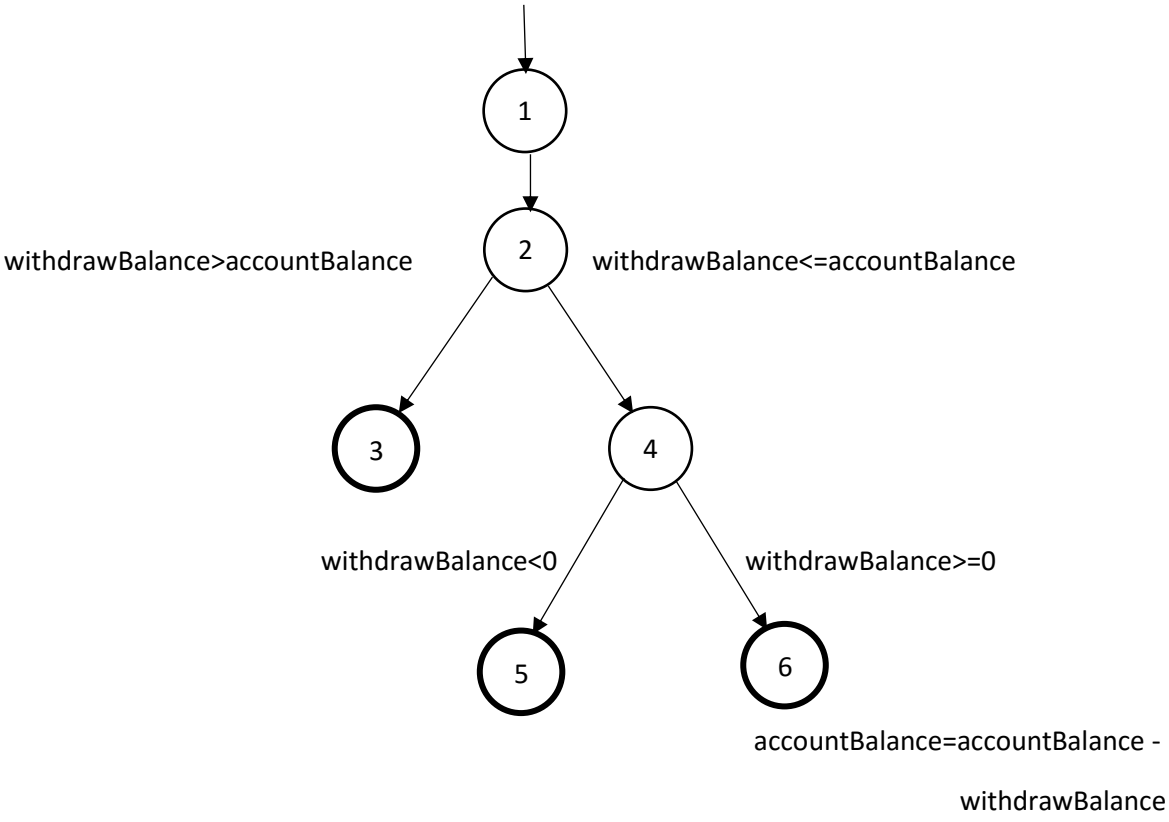

69
70
Counter<accountHolder
counter>=accountHolder
71
72
searchResult[counter]!=-1
searchResult[counter]==-1
73
74
counter=counter+1

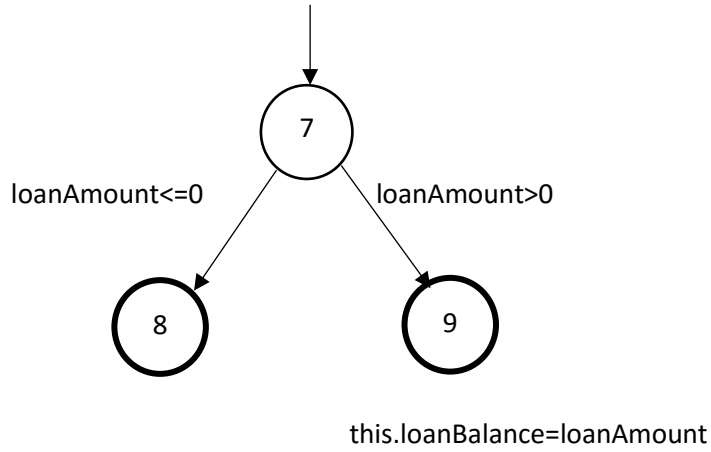
Graph Coverage for ForeignAccountHolder class



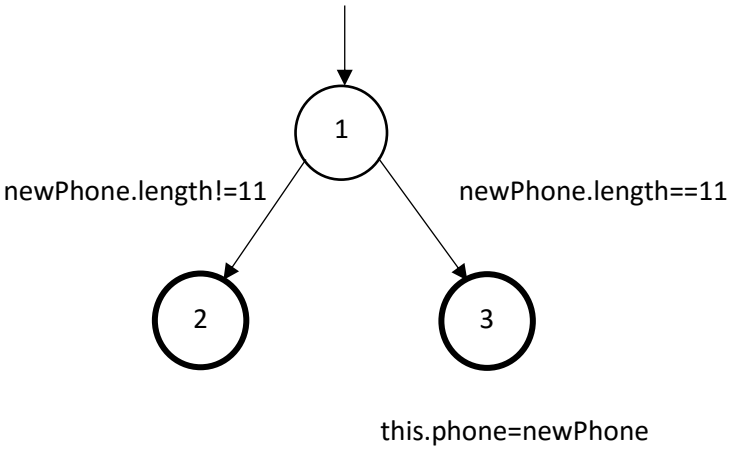


Graph Coverage for LocalAccountHolder class

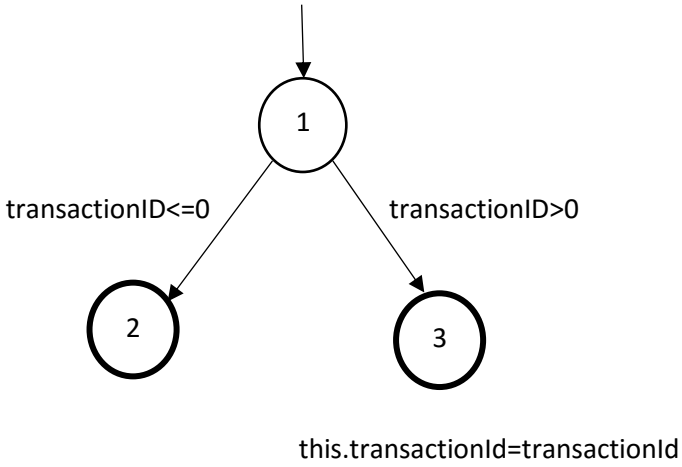


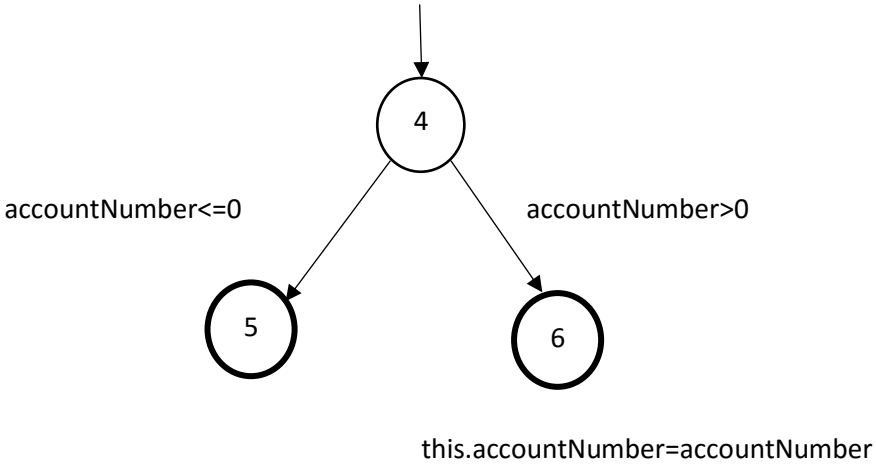


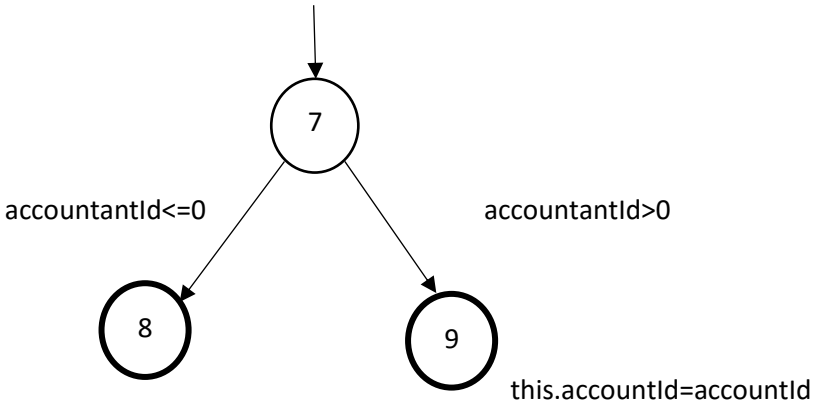
Graph Coverage for Person class

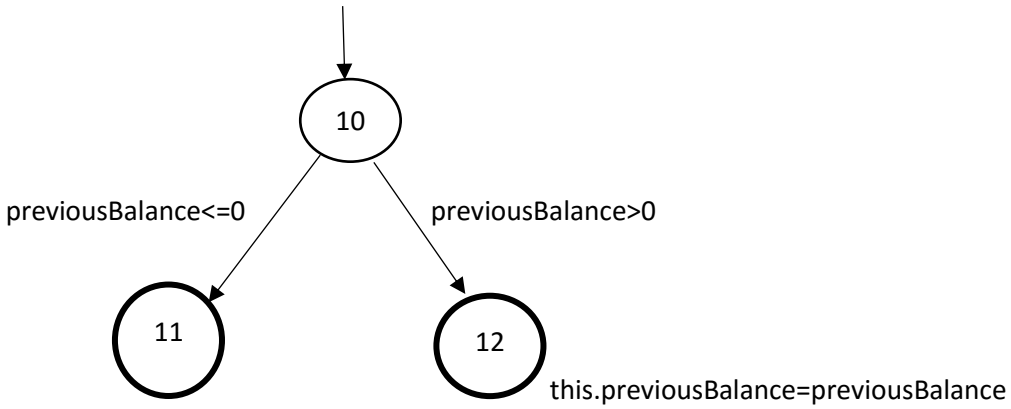


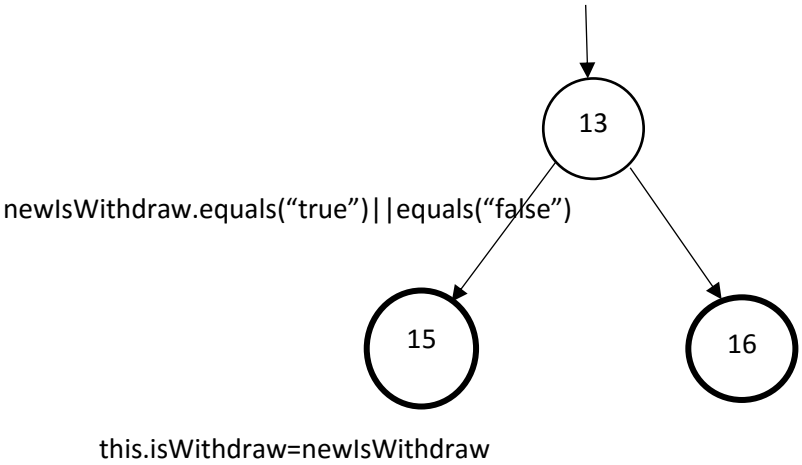
Graph Coverage for Transaction class

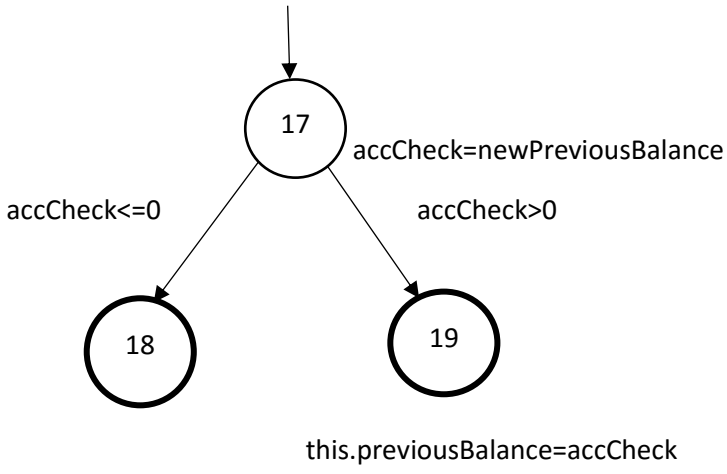


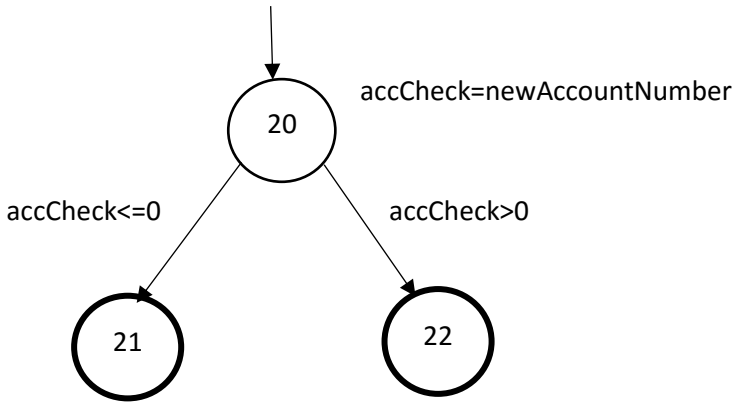




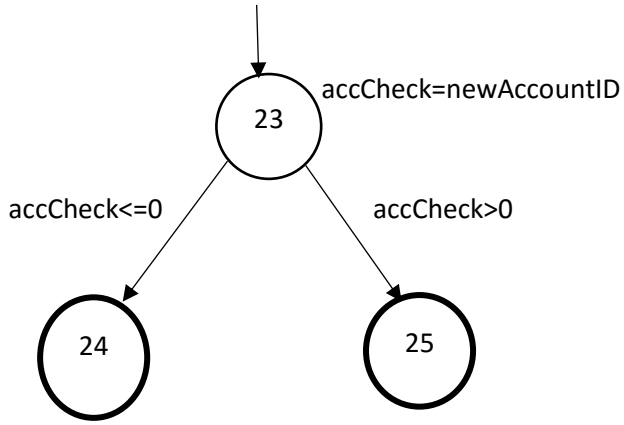

10
previousBalance<=0
previousBalance>0
11
12
this.previousBalance=previousBalance


13
newIsWithdraw.equals(“true”)||equals(“false”)
15
16
this.isWithdraw=newIsWithdraw


17
accCheck=newPreviousBalance
accCheck<=0
accCheck>0
18
19
this.previousBalance=accCheck



this.accountNumber=accCheck



this.accountId=accCheck