Bank Transaction

**Grade settings**: Maximum grade: 100  
**Disable external file upload, paste and drop external content**: Yes  
**Run**: Yes **Evaluate**: Yes  
**Automatic grade**: Yes

[***Click here to download the code template***](https://cognizant.tekstac.com/pluginfile.php/54382/mod_vpl/intro/BankTransaction.zip?time=1614944685846)

***Bank System***is an automated bank transaction management system. Using the application, you can maintain information of Transactions where the transaction type is checked and information can be accessed for given transaction id or based on the account number the transactions performed can be retrieved.

Prithvi has developed an application for the above purpose. The details of the various functions supported by the system are provided in this case study.

You are required to write Junit test case and check the correctness of the application developed.

**Functional Requirements:**

The application has the below classes and methods implemented.

You are provided with a model class Transaction

**Component Specification: Transaction** **(Model Class)**

|  |  |  |
| --- | --- | --- |
| **Type(Class)** | **Attributes** | **Methods** |
| Transaction | int transactionId  Date transactionDate  String accountNumber  String transactionType  double amount | Necessary getters and setters are provided.  A constructor is also provided. |

Here the **transactionType**can take a value either “Credit” or “Debit”.

**Component Specification:**InvalidTransactionException**(This class inherits the Exception Class)**

|  |  |
| --- | --- |
| **Type(Class)** | **Methods** |
| InvalidTransactionException | Provided with a single argument constructor – InvalidTransactionException(String message) |

You are also provided with a utility class Bank with business methods.

**Component Specification:**Bank **(Utility Class)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Component Name** | **Type(Class)** | **Attributes** | **Methods** |
| Perform various transactions in a Bank | Bank |  | Has various methods to manipulate the transactions. |

The below are the requirements  implemented in the Utility class for which JUnit test cases are to be written and tested.

**Component Specification:**Bank **(Utility Class)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Component Name** | **Type (Class)** | **Methods** | **Responsibilities** | **Exception** |
| Validate the transaction type | Bank | public boolean validateTransactionType(String transactionType) | Validate the transactionType..  If valid return true , else this method should throw a user defined exception | Throw a user defined exception “InvalidTransactionException”  if the transactionType Is neither “Debit” nor “Credit” |
| View the Transaction based on transactionId | Bank | public Transaction viewTransaction(List<Transaction> transactionList,int transactionId) | This method should return the Transaction object with the transactionId passed as parameter from list of  transaction which is also passed as parameter.  If the transactionList is empty or if there is no transaction in the given transaction id it should throw a user defined exception. | Throw a user defined exception “InvalidTransactionException” if the  transactionList is empty or if there is  no transaction in the given transaction id. |
| View the list of transaction for a given Account number | Bank | public List<Transaction> viewTransactionForAccount(List<Transaction> transactionList ,String accountNumber) | This method takes the transactionList and an accountNumber as argument. It should return the list of transaction for a given account number. If the transactionList is empty it should throw a user defined exception. | Throw a user defined exception “InvalidTransactionException” if the  transactionList is empty |
| View accout wise transaction performed for all Accounts | Bank | public Map<String,List<Transaction>>  viewTransactionAccountwise(List<Transaction> transactionList) | This method takes the transactionList as argument. It should return accoutwise transactions made for all accounts in the list. The return type is map, where the accountNumber is key and value is the List of Transaction performed on that account.  If the transactionList is empty it should throw a user defined exception. | Throw a user defined exception “InvalidTransactionException” if the  transactionList is empty. |
| View the number of transactions for each Account within a given range of date | Bank | public  Map<String,Integer> countTransactionsAccountwise(List<Transaction> transactionList , Date fromDate,Date toDate) | This method should return the number of transactions performed on each accountNumber in the transactionList for a given dateRange(From fromDate to toDate). It takes the transactionList, fromDate and toDate as arguments and returns a Map with key as accountNumber and value as count of transactions within the given date range.                          If the transactionList is empty it should throw a user defined exception. | Throw a user defined exception “InvalidTransactionException” if the  transactionList is empty. |

You need to write Junit test for the Bank class.

**Testing Scenarios:**

You are provided with a class  “BankTest”  to do this testing.

**Note:**

To perform testing the transactionList should contain objects of Transaction.

To do this, in BankTest  class you are provided with a setup method.  Use this method to populate the static variable transactionList  in BankTest class.  That is, create few objects for  Transaction,  populate the transactionList given in BankTest class with these objects and use that list to test the methods  in Bank  class that needs a transaction list to be passed as attribute.

The below are the test methods to be implemented in BankTest class.

|  |  |
| --- | --- |
| **Test Method** | **Scenarios / Responsibilities** |
| test11ValidateTransactionTypeWhenCredit | This method should test the validateTransactionType method when “Credit” is passed as parameter |
| test12ValidateTransactionTypeWhenDebit | This method should test the validateTransactionType method when “Debit” is passed as parameter. |
| test13ValidateTransactionTypeWhenInvalid | This method should test the validateTransactionType method when invalid value is passed as parameter  validateTransactionType is expected  to throw InvalidTransactionException when type is invalid.  Write JUnit to test for the exception thrown  either by using appropriate annotation or by using try catch block. |
| test14ViewTransactionForValidId | This method should test the correctness of  viewTransaction method for  an existing transaction id .  Perform testing for the correctness of the value returned. |
| test15ViewTransactionForInvalidId | This method should test the correctness of  viewTransaction method for a non existing transaction id.  Perform testing for the correctness of the value returned.  viewTransaction method is expected  to throw InvalidTransactionException when transaction id does not exist.  Write JUnit to test for the exception thrown  either by using appropriate annotation or by using try catch block |
| test16ViewTransactionForAccount | This method should test the correctness of  viewTransactionForAccount method.  Perform testing for the correctness of the value returned. |
| test17ViewTransactionForAccountForEmptyList | This method should test the correctness of viewTransactionForAccount method for an empty transactionList.  viewTransactionForAccount method is expected  to throw InvalidTransactionException when transaction list is empty.  Write JUnit to test for the exception thrown  either by using appropriate annotation or by using try catch block |
| test18ViewTransactionAccountwise | This method should test the correctness of viewTransactionAccountwise method.  Perform testing for the correctness of the value returned. |
| test19ViewTransactionAccountwiseForEmptyList | This method should test the correctness of viewTransactionAccountwise method for an empty transactionList.  viewTransactionAccountwise method is expected  to throw InvalidTransactionException when transaction list is empty.  Write JUnit to test for the exception thrown  either by using appropriate annotation or by using try catch block |
| test20CountTransactionsAccountwise | This method should test the correctness of countTransactionsAccountwise method.  Perform testing for the correctness of the value returned. |
| test21CountTransactionsAccountwiseForEmptyList | This method should test the correctness of countTransactionsAccountwise method for an empty transactionList.  countTransactionsAccountwise  method is expected  to throw InvalidTransactionException when transaction list is empty.  Write JUnit to test for the exception thrown  either by using appropriate annotation or by using try catch block |

Implement the test methods and provide the needed annotation to all the methods in BankTest class.

Also this class is provided with  the annotation, so that the test methods are executed in ascending order of the test method names.

You are provided with a Main class with the main method to check the correctness of the test methods written in BankTest class.

Having completed writing the test methods, uncomment the code in Main class and execute the main method.

### **Automatic evaluation**[**[-]**](javascript:void(0);)

**Proposed grade: 100.0 / 100**  
**Result Description**  
[[-]](javascript:void(0);)**Grading and Feedback**

*Writing JUnit for a Utility class - 60.0 / 60.0(Success)*

*Test Coverage - 40 / 40.0(Success)*

Test Case Passed

#### **BankTransaction/src/com/bank/exception/InvalidTransactionException.java**

1 *package* com.bank.exception;

2

3 *public* *class* InvalidTransactionException *extends* Exception {

4

5 *public* InvalidTransactionException(String msg) {

6 *super*(msg);

7 }

8 }

9

#### **BankTransaction/src/com/bank/model/Transaction.java**

1 *package* com.bank.model;

2

3 *import* java.util.Date;

4

5 *public* *class* Transaction {

6 *private* *int* transactionId;

7 *private* Date transactionDate;

8 *private* String accountNumber;

9 *private* String transactionType;

10 *private* *double* amount;

11

12

13 *public* Transaction(){

14

15 }

16

17 *public* Transaction(*int* transactionId, Date transactionDate, String accountNumber, String transactionType,

18 *double* amount) {

19 *this*.transactionId = transactionId;

20 *this*.transactionDate = transactionDate;

21 *this*.accountNumber = accountNumber;

22 *this*.transactionType = transactionType;

23 *this*.amount = amount;

24 }

25

26

27 *public* *int* getTransactionId() {

28 *return* transactionId;

29 }

30 *public* *void* setTransactionId(*int* transactionId) {

31 *this*.transactionId = transactionId;

32 }

33 *public* Date getTransactionDate() {

34 *return* transactionDate;

35 }

36 *public* *void* setTransactionDate(Date transactionDate) {

37 *this*.transactionDate = transactionDate;

38 }

39 *public* String getAccountNumber() {

40 *return* accountNumber;

41 }

42 *public* *void* setAccountNumber(String accountNumber) {

43 *this*.accountNumber = accountNumber;

44 }

45 *public* String getTransactionType() {

46 *return* transactionType;

47 }

48 *public* *void* setTransactionType(String transactionType) {

49 *this*.transactionType = transactionType;

50 }

51 *public* *double* getAmount() {

52 *return* amount;

53 }

54 *public* *void* setAmount(*double* amount) {

55 *this*.amount = amount;

56 }

57

58

59

60 }

61

#### **BankTransaction/src/com/bank/skeleton/SkeletonValidator.java**

1 *package* com.bank.skeleton;

2

3 *import* java.lang.reflect.Method;

4 *import* java.util.logging.Level;

5 *import* java.util.logging.Logger;

6

7 /\*\*

8 \* This class is used to verify if the Code Skeleton is intact

9 \* and not modified by participants thereby ensuring smooth auto

10 \* evaluation

11 \*/

12

13 *public* *class* SkeletonValidator {

14

15 *int* count=0;

16 *int* count1=0;

17 *public* SkeletonValidator() {

18 validateClassName("com.bank.util.Bank");

19 validateClassName("com.bank.model.Transaction");

20 validateClassName("com.bank.exception.InvalidTransactionException");

21 validateClassName("com.bank.test.BankTest");

22

23 validateMethodSignature(

24 "validateTransactionType:boolean,viewTransaction:com.bank.model.Transaction,viewTransactionForAccount:java.util.List,"

25 + "viewTransactionAccountwise:java.util.Map,countTransactionsAccountwise:java.util.Map",

26 "com.bank.util.Bank");

27 validateMethodSignature(

28 "test11ValidateTransactionTypeWhenCredit:void,test12ValidateTransactionTypeWhenDebit:void,test13ValidateTransactionTypeWhenInvalid:void,test14ViewTransactionForValidId:void,test15ViewTransactionForInvalidId:void,"

29 + "test16ViewTransactionForAccount:void,test17ViewTransactionForAccountForEmptyList:void,test18ViewTransactionAccountwise:void,test19ViewTransactionAccountwiseForEmptyList:void,"

30 + "test20CountTransactionsAccountwise:void,test21CountTransactionsAccountwiseForEmptyList:void",

31 "com.bank.test.BankTest");

32 *if*(count==0 && count1==0)

33 System.out.println("All class and method names are correct");

34 *else* *if*(count>0)

35 System.out.println("Please check the correctness of class names");

36 *else*

37 System.out.println("Please check the correctness of method names");

38 }

39

40 *private* *static* *final* Logger LOG = Logger.getLogger("SkeletonValidator");

41

42 *protected* *final* *boolean* validateClassName(String className) {

43

44 *boolean* iscorrect = *false*;

45 *try* {

46 Class.forName(className);

47 iscorrect = *true*;

48 LOG.info("Class Name " + className + " is correct");

49

50 } *catch* (ClassNotFoundException e) {

51 count++;

52 LOG.log(Level.SEVERE, "You have changed either the " + "class name/package. Use the correct package "

53 + "and class name as provided in the skeleton");

54

55 } *catch* (Exception e) {

56 count++;

57 LOG.log(Level.SEVERE,

58 "There is an error in validating the " + "Class Name. Please manually verify that the "

59 + "Class name is same as skeleton before uploading");

60 }

61 *return* iscorrect;

62 }

63

64 *protected* *final* *void* validateMethodSignature(String methodWithExcptn, String className) {

65 Class cls = *null*;

66 *try* {

67

68 String[] actualmethods = methodWithExcptn.split(",");

69 *boolean* errorFlag = *false*;

70 String[] methodSignature;

71 String methodName = *null*;

72 String returnType = *null*;

73

74 *for* (String singleMethod : actualmethods) {

75 *boolean* foundMethod = *false*;

76 methodSignature = singleMethod.split(":");

77

78 methodName = methodSignature[0];

79 returnType = methodSignature[1];

80 cls = Class.forName(className);

81 Method[] methods = cls.getMethods();

82 *for* (Method findMethod : methods) {

83 *if* (methodName.equals(findMethod.getName())) {

84 foundMethod = *true*;

85 *if* (!(findMethod.getReturnType().getName().equals(returnType))) {

86 errorFlag = *true*;

87 count1++;

88 LOG.log(Level.SEVERE, " You have changed the " + "return type in '" + methodName

89 + "' method. Please stick to the " + "skeleton provided");

90

91 } *else* {

92 LOG.info("Method signature of " + methodName + " is valid");

93 }

94

95 }

96 }

97 *if* (!foundMethod) {

98 errorFlag = *true*;

99 count1++;

100 LOG.log(Level.SEVERE, " Unable to find the given public method " + methodName

101 + ". Do not change the " + "given public method name. " + "Verify it with the skeleton");

102 }

103

104 }

105 *if* (!errorFlag) {

106 LOG.info("Method signature is valid");

107 }

108

109 } *catch* (Exception e) {

110 count1++;

111 LOG.log(Level.SEVERE,

112 " There is an error in validating the " + "method structure. Please manually verify that the "

113 + "Method signature is same as the skeleton before uploading");

114 }

115 }

116

117 }

118

#### **BankTransaction/src/com/bank/test/BankTest.java**

1 *package* com.bank.test;

2

3 *import* java.util.ArrayList;

4 *import* java.util.List;

5 *import* java.util.Map;

6 *import* java.util.Date;

7 *import* java.util.LinkedHashMap;

8

9 *import* *static* org.junit.Assert.assertArrayEquals;

10 *import* *static* org.junit.Assert.assertEquals;

11 *import* *static* org.junit.Assert.assertTrue;

12

13 *import* java.text.ParseException;

14 *import* java.text.SimpleDateFormat;

15

16 *import* org.junit.BeforeClass;

17 *import* org.junit.FixMethodOrder;

18 *import* org.junit.Rule;

19 *import* org.junit.Test;

20 *import* org.junit.rules.ExpectedException;

21 *import* org.junit.runners.MethodSorters;

22

23 *import* com.bank.exception.InvalidTransactionException;

24 *import* com.bank.model.Transaction;

25 *import* com.bank.util.Bank;

26

27 @FixMethodOrder(MethodSorters.NAME\_ASCENDING)

28 *public* *class* BankTest {

29

30 @Rule

31 *public* ExpectedException exceptionRule=ExpectedException.none();

32 *private* *static* List<Transaction> transactionList = *new* ArrayList<Transaction>();

33 *private* *static* Bank bankObj;

34 *private* *static* Transaction trans1;

35 *private* *static* Transaction trans2;

36 *private* *static* Transaction trans3;

37 *private* *static* Transaction trans4;

38

39 @BeforeClass

40 *public* *static* *void* setUp() throws Exception {

41

42 bankObj = *new* Bank();

43

44 //Create few Transaction objects and add to transactionList.

45 //Use that list to the test all the methods in Bank class that requires a list of transaction

46 trans2=*new* Transaction(101,*new* SimpleDateFormat("dd/MM/yy").parse("12/03/1999"),"10586465332566","Credit",6000);

47 trans3=*new* Transaction(102,*new* SimpleDateFormat("dd/MM/yy").parse("12/03/1999"),"10586465332566","Debit",6000);

48 trans4=*new* Transaction(103,*new* SimpleDateFormat("dd/MM/yy").parse("1/03/1999"),"10586465332567","Debit",6000);

49

50 }

51

52 //Test the validateTransactionType method when the value is Credit

53 @Test

54 *public* *void* test11ValidateTransactionTypeWhenCredit() throws InvalidTransactionException {

55 //fill code here

56 assertTrue(bankObj.validateTransactionType("Credit"));

57 }

58

59 //Test the validateTransactionType method when the value is Debit

60 @Test

61 *public* *void* test12ValidateTransactionTypeWhenDebit() throws InvalidTransactionException {

62 //fill code here

63 assertTrue(bankObj.validateTransactionType("Debit"));

64

65 }

66

67 //Test the validateTransactionType method when the value is invalid

68 @Test

69 *public* *void* test13ValidateTransactionTypeWhenInvalid() throws InvalidTransactionException {

70 //fill code here

71 exceptionRule.expect(InvalidTransactionException.*class*);

72 exceptionRule.expectMessage("Transaction type is invalid");

73 bankObj.validateTransactionType("deb");

74 }

75

76 //Test the viewTransaction method For Valid Id

77 @Test

78 *public* *void* test14ViewTransactionForValidId() throws InvalidTransactionException {

79 //fill code here

80 List<Transaction> t=*new* ArrayList<>();

81 t.add(trans2);

82 t.add(trans3);

83 assertEquals(trans2,(bankObj.viewTransaction(t,101)));

84 }

85

86 //Test the viewTransaction method For Invalid Id

87 @Test

88 *public* *void* test15ViewTransactionForInvalidId() throws InvalidTransactionException{

89 List<Transaction> t=*new* ArrayList<>();

90 t.add(trans2);

91 t.add(trans3);

92 exceptionRule.expect(InvalidTransactionException.*class*);

93 exceptionRule.expectMessage("Transaction ID is invalid");

94 assertEquals(trans2,bankObj.viewTransaction(t,105));

95 }

96

97 //Test the viewTransactionForAccount method

98 @Test

99 *public* *void* test16ViewTransactionForAccount() throws InvalidTransactionException {

100 //fill code here

101 List<Transaction> t=*new* ArrayList<>();

102 t.add(trans2);

103 t.add(trans3);

104 t.add(trans4);

105 List<Transaction> t1=*new* ArrayList<>();

106 t1.add(trans2);

107 t1.add(trans3);

108 assertEquals(t1,bankObj.viewTransactionForAccount(t,"10586465332566"));

109 }

110

111

112 //Test the viewTransactionForAccount method when list is empty

113 @Test

114 *public* *void* test17ViewTransactionForAccountForEmptyList() throws InvalidTransactionException {

115 //fill code here

116 List<Transaction> t=*new* ArrayList<Transaction>();

117 exceptionRule.expect(InvalidTransactionException.*class*);

118 exceptionRule.expectMessage("Transaction list is empty");

119 assertEquals(t,bankObj.viewTransactionForAccount(t,"10586465332566"));

120

121

122 }

123

124 //Test the viewTransactionAccountwise method

125 @Test

126 *public* *void* test18ViewTransactionAccountwise() throws InvalidTransactionException {

127 //fill code here

128 Map<String,List<Transaction>> result= *new* LinkedHashMap<>();

129 List<Transaction> t2=*new* ArrayList<>();

130 t2.add(trans2);

131 t2.add(trans3);

132 t2.add(trans4);

133 List<Transaction> t=*new* ArrayList<>();

134 t.add(trans2);

135 t.add(trans3);

136 List<Transaction> t1=*new* ArrayList<>();

137 t1.add(trans4);

138 result.put("10586465332566",t);

139 result.put("10586465332567",t1);

140 assertEquals(result,bankObj.viewTransactionAccountwise(t2));

141 }

142

143 //Test the viewTransactionAccountwise method when list is empty

144 @Test

145 *public* *void* test19ViewTransactionAccountwiseForEmptyList() throws InvalidTransactionException {

146 //fill code here

147 Map<String,List<Transaction>> result=*new* LinkedHashMap<>();

148 List<Transaction> t2=*new* ArrayList<>();

149 exceptionRule.expect(InvalidTransactionException.*class*);

150 exceptionRule.expectMessage("Transaction list is empty");

151 assertEquals(result,bankObj.viewTransactionAccountwise(t2));

152 }

153

154 //Test the countTransactionsAccountwise method

155 @Test

156 *public* *void* test20CountTransactionsAccountwise() throws InvalidTransactionException,ParseException {

157 //fill code here

158 Map<String,Integer> result=*new* LinkedHashMap<>();

159 List<Transaction> t2=*new* ArrayList<>();

160 t2.add(trans2);

161 t2.add(trans3);

162 t2.add(trans4);

163 result.put("10586465332566", 2);

164 result.put("10586465332567", 1);

165 assertEquals(result,bankObj.countTransactionsAccountwise(t2,*new* SimpleDateFormat("dd/MM/yyyy").parse("1/02/1999"),*new* SimpleDateFormat("dd/MM/yy").parse("13/03/1999")));

166

167 }

168

169 //Test the countTransactionsAccountwise method when list is empty

170 @Test

171 *public* *void* test21CountTransactionsAccountwiseForEmptyList() throws InvalidTransactionException,ParseException {

172 //fill code here

173 Map<String,List<Transaction>> result =*new* LinkedHashMap<>();

174 List<Transaction> t2=*new* ArrayList<>();

175 exceptionRule.expect(InvalidTransactionException.*class*);

176 exceptionRule.expectMessage("Transaction list is empty");

177 assertEquals(result,bankObj.countTransactionsAccountwise(t2,*new* SimpleDateFormat("dd/MM/yyyy").parse("1/02/1999"),*new* SimpleDateFormat("dd/MM/yyyy").parse("13/03/1999")));

178 }

179 }

180

#### **BankTransaction/src/com/bank/test/Main.java**

1 *package* com.bank.test;

2

3 *import* org.junit.runner.JUnitCore;

4 *import* org.junit.runner.Result;

5

6 *import* com.bank.skeleton.SkeletonValidator;

7

8

9 *public* *class* Main {

10

11 *public* *static* *void* main(String[] args) {

12

13 SkeletonValidator validator = *new* SkeletonValidator();

14

15 /\*Result result = JUnitCore.runClasses(BankTest.class);

16 if (result.getFailureCount() == 0) {

17 System.out.println("All Test cases Cleared");

18 } else {

19 System.out.println("One or more test case(s) failed");

20 System.out.println("===============================");

21 result.getFailures().forEach(x -> System.out.println(x.getMessage()));

22 }

23 \*/

24

25 }

26

27 }

28

#### **BankTransaction/src/com/bank/util/Bank.java**

1 *package* com.bank.util;

2

3 *import* java.util.\*;

4

5 *import* com.bank.exception.InvalidTransactionException;

6 *import* com.bank.model.Transaction;

7

8 *public* *class* Bank {

9

10 *public* *boolean* validateTransactionType(String transactionType) throws InvalidTransactionException {

11 *if*(transactionType.equals("Debit") || transactionType.equals("Credit"))

12 *return* *true*;

13 *else*

14 *throw* *new* InvalidTransactionException("Transaction type is invalid");

15 }

16

17 *public* Transaction viewTransaction(List<Transaction> transactionList,*int* transactionId) throws InvalidTransactionException{

18 *if*(transactionList.size()==0){

19 *throw* *new* InvalidTransactionException("Transaction list is empty");

20 }

21 *else*{

22 *for*(Transaction t : transactionList){

23 *if*(t.getTransactionId()==transactionId){

24 *return* t;

25 }

26 }

27 *throw* *new* InvalidTransactionException("Transaction ID is invalid");

28 }

29 }

30

31 *public* List<Transaction> viewTransactionForAccount(List<Transaction> transactionList ,String accountNumber) throws InvalidTransactionException {

32 *if*(transactionList.size()==0){

33 *throw* *new* InvalidTransactionException("Transaction list is empty");

34 }

35 *else*{

36 List<Transaction> result = *new* ArrayList<>();

37 *for*(Transaction t : transactionList){

38 *if*(t.getAccountNumber().equals(accountNumber)){

39 result.add(t);

40 }

41 }

42 *return* result;

43 }

44

45 }

46

47

48 *public* Map<String,List<Transaction>> viewTransactionAccountwise(List<Transaction> transactionList) throws InvalidTransactionException {

49 *if*(transactionList.size()==0){

50 *throw* *new* InvalidTransactionException("Transaction list is empty");

51 }

52 *else* {

53 Map<String,List<Transaction>> result = *new* LinkedHashMap<>();

54

55 *for*(Transaction t : transactionList){

56 *if*(!result.containsKey(t.getAccountNumber())){

57 result.put(t.getAccountNumber(),*new* ArrayList<Transaction>());

58 }

59 List<Transaction> temp=result.get(t.getAccountNumber());

60 temp.add(t);

61 result.put(t.getAccountNumber(), temp);

62 }

63 *return* result;

64 }

65 }

66

67 *public* Map<String,Integer> countTransactionsAccountwise(List<Transaction> transactionList , Date fromDate,Date toDate) throws InvalidTransactionException {

68 *if*(transactionList.size()==0){

69 *throw* *new* InvalidTransactionException("Transaction list is empty");

70 }

71 *else* {

72 Map<String,Integer> result = *new* LinkedHashMap<>();

73

74 *for*(Transaction t : transactionList){

75 *if*(t.getTransactionDate().after(fromDate) && t.getTransactionDate().before(toDate)){

76 *if*(!result.containsKey(t.getAccountNumber())){

77 result.put(t.getAccountNumber(),1);

78 }

79 *else*

80 result.put(t.getAccountNumber(), result.get(t.getAccountNumber())+1);

81 }

82

83 }

84 *return* result;

85 }

86 }

87 }

## Grade

Reviewed on Monday, 3 May 2021, 11:19 PM by Automatic grade  
**Grade** 100 / 100  
**Assessment report**  
[[-]](javascript:void(0);)**Grading and Feedback**

*Writing JUnit for a Utility class - 60.0 / 60.0(Success)*

*Test Coverage - 40 / 40.0(Success)*

Test Case Passed