Universitatea din București

Facultatea de Matematică și Informatică

Specializare: Calculatoare și Tehnologia Informației

Unit testing cu Java

Sursa cod testat: <https://github.com/AndritaLucianGabriel/TSS/tree/master>

Studenti

Oprea Ruxandra Ana-Maria

Buzatu Andreea Sabina

Andriță Lucian-Gabriel

București 2023

Table of Contents

[Teste functionale 2](#_Toc131001283)

[Partitionarea in categorii 2](#_Toc131001284)

[Codul testat: 2](#_Toc131001285)

[Domeniul de intrari: 3](#_Toc131001286)

[Domeniul de iesiri: 3](#_Toc131001287)

[Clasele de echivalenta: 4](#_Toc131001288)

Analiza valorilor de frontiera……………………………………………………………………………………………………………..15

[Codul testat: 2](#_Toc131001285)

[Domeniul de intrari: 3](#_Toc131001286)

[Domeniul de iesiri: 3](#_Toc131001287)

[Clasele de echivalenta: 4](#_Toc131001288)

Partitionarea de echivalenta……………………………………………………………………………………………………………….

[Codul testat: 2](#_Toc131001285)

[Domeniul de intrari: 3](#_Toc131001286)

[Domeniul de iesiri: 3](#_Toc131001287)

[Clasele de echivalenta: 4](#_Toc131001288)

[Teste structurale 15](#_Toc131001289)

[Acoperirea la nivel de instructiune 15](#_Toc131001290)

[Analiza set teste 17](#_Toc131001291)

# Teste functionale

## Partitionarea in categorii

### Codul testat:

|  |
| --- |
| public void currencyExchange(BankAccount bankAccount, String wantedCurrency) throws BankAccountException {  Timestamp.timestamp("Bank,currencyExchange");  String oldCurrency = bankAccount.getCurrency();  int c = 0, c1 = 0;  for (Map.Entry<Client, List<BankAccount>> x : this.clientBankAccountMap.entrySet()) {  for (BankAccount y : x.getValue()) {  if (Objects.equals(y.getIBAN(), bankAccount.getIBAN())) {  c++;  if (!bankAccount.getClosingDate().equals("-"))  c1++;  else {  boolean check = bankAccount.currencyExchange(wantedCurrency, getClientBankAccountMap());  if (check) {  System.out.println("Conversie realizata! (" + oldCurrency + "->" + wantedCurrency + ")");  } else {  System.out.println("Conversie esuata!");  }  }  }  }  }  if (c == 0)  System.out.println("Nu exista contul " + bankAccount.getIBAN());  else if (c1 != 0)  System.out.println("Nu se poate face conversia! Contul " + bankAccount.getIBAN() + " a fost inchis!");  } |

### Domeniul de intrari:

Nota: Nu se poate trimite null ca referinta pentru BankAccount deoarece parametrul trebuie sa implementeze acea clasa abstracta (altfel primesc eroare la compilare -ambigous-)

!! Fie multimea conturilor existente ale bancii = P !!

* Un obiect ce implementeaza BankAccount

N\_1 = {x | x = DebitAccount ∈ P cu closingDate != "-"}

N\_2 = {x | x = DebitAccount ∈ P cu closingDate = "-"}

N\_3 = {x | x = SavingsAccount ∈ P cu closingDate != "-"}

N\_4 = {x | x = SavingsAccount ∈ P cu closingDate = "-"}

N\_5 = {x | x ∉ P}

* Un obiect de tip String

S\_1 = {y | y = null}

S\_2 = {y | y ∈ {"Lei", "Euro", "Dolari"}}

S\_3 = {y | y ∉ {"Lei", "Euro", "Dolari"}}

### Domeniul de iesiri:

C\_1(x, y) = {c | "Conversie realizata!"}

C\_2(x, y) = {c | "Conversie esuata!"}

C\_3(x, y) = {c | "Nu exista contul!"}

C\_4(x, y) = {c | "Nu se poate face conversia! Contul a fost inchis!"}

### Clasele de echivalenta:

C\_112 = {(x, y, c) | n ∈ N\_1, y ∈ S\_1, c ∈ C\_2}

C\_121 = {(x, y, c) | n ∈ N\_1, y ∈ S\_2, c ∈ C\_1}

C\_132 = {(x, y, c) | n ∈ N\_1, y ∈ S\_3, c ∈ C\_2}

C\_214 = {(x, y, c) | n ∈ N\_2, y ∈ S\_1, c ∈ C\_4}

C\_224 = {(x, y, c) | n ∈ N\_2, y ∈ S\_2, c ∈ C\_4}

C\_234 = {(x, y, c) | n ∈ N\_2, y ∈ S\_3, c ∈ C\_4}

C\_312 = {(x, y, c) | n ∈ N\_3, y ∈ S\_1, c ∈ C\_2}

C\_321 = {(x, y, c) | n ∈ N\_3, y ∈ S\_2, c ∈ C\_1}

C\_332 = {(x, y, c) | n ∈ N\_3, y ∈ S\_3, c ∈ C\_2}

C\_414 = {(x, y, c) | n ∈ N\_4, y ∈ S\_1, c ∈ C\_4}

C\_424 = {(x, y, c) | n ∈ N\_4, y ∈ S\_2, c ∈ C\_4}

C\_434 = {(x, y, c) | n ∈ N\_4, y ∈ S\_3, c ∈ C\_4}

C\_513 = {(x, y, c) | n ∈ N\_5, y ∈ S\_1, c ∈ C\_3}

C\_523 = {(x, y, c) | n ∈ N\_5, y ∈ S\_2, c ∈ C\_3}

C\_533 = {(x, y, c) | n ∈ N\_5, y ∈ S\_3, c ∈ C\_3}

|  |
| --- |
| @Test  public void testCurrencyExchange\_C112 () throws BankAccountException {  //Integrity checks  BankAccount tmp = getBankAccountFromMap(debitAccount1, bankClient, bank.getClientBankAccountMap());  Assert.assertEquals(debitAccount1, tmp);  // Prepping for the final checks  String wantedCurrency = null;  double valueAfterConversion = convert(debitAccount1, debitAccount1.getBalance(), debitAccount1.getCurrency(), wantedCurrency);  // Actual call  bank.currencyExchange(debitAccount1, wantedCurrency);  // Retrieving the modified account from the bank  tmp = getBankAccountFromMap(debitAccount1, bankClient, bank.getClientBankAccountMap());  // Verify the accounts' updated information  Assert.assertEquals(debitAccount1.getCurrency(), tmp.getCurrency());  Assert.assertEquals(valueAfterConversion, tmp.getBalance(), Math.ulp(valueAfterConversion));  // Check the messages. The logs are really detailed so we have to check if our string is in them  Assert.assertTrue(outputStreamCaptor.toString().trim().contains("Conversie esuata!"));  }  @Test  public void testCurrencyExchange\_C121 () throws BankAccountException {  //Integrity checks  BankAccount tmp = getBankAccountFromMap(debitAccount1, bankClient, bank.getClientBankAccountMap());  Assert.assertEquals(debitAccount1, tmp);  // Prepping for the final checks  String wantedCurrency = "Euro";  double valueAfterConversion = convert(debitAccount1, debitAccount1.getBalance(), debitAccount1.getCurrency(), wantedCurrency);  // Actual call  bank.currencyExchange(debitAccount1, wantedCurrency);  // Retrieving the modified account from the bank  tmp = getBankAccountFromMap(debitAccount1, bankClient, bank.getClientBankAccountMap());  // Verify the accounts' updated information  Assert.assertEquals(wantedCurrency, tmp.getCurrency());  Assert.assertEquals(valueAfterConversion, tmp.getBalance(), Math.ulp(valueAfterConversion));  // Check the messages. The logs are really detailed so we have to check if our string is in them  Assert.assertTrue(outputStreamCaptor.toString().trim().contains("Conversie realizata! (Lei->Euro)"));  }  @Test  public void testCurrencyExchange\_C132 () throws BankAccountException {  //Integrity checks  BankAccount tmp = getBankAccountFromMap(debitAccount1, bankClient, bank.getClientBankAccountMap());  Assert.assertEquals(debitAccount1, tmp);  // Prepping for the final checks  String wantedCurrency = "Yen";  double valueAfterConversion = convert(debitAccount1, debitAccount1.getBalance(), debitAccount1.getCurrency(), wantedCurrency);  // Actual call  bank.currencyExchange(debitAccount1, wantedCurrency);  // Retrieving the modified account from the bank  tmp = getBankAccountFromMap(debitAccount1, bankClient, bank.getClientBankAccountMap());  // Verify the accounts' updated information  Assert.assertEquals(debitAccount1.getCurrency(), tmp.getCurrency());  Assert.assertEquals(valueAfterConversion, tmp.getBalance(), Math.ulp(valueAfterConversion));  // Check the messages. The logs are really detailed so we have to check if our string is in them  Assert.assertTrue(outputStreamCaptor.toString().trim().contains("Conversie esuata!"));  }  @Test  public void testCurrencyExchange\_C214 () throws BankAccountException {  //Integrity checks  BankAccount tmp = getBankAccountFromMap(debitAccount2, bankClient, bank.getClientBankAccountMap());  Assert.assertEquals(debitAccount2, tmp);  // Prepping for the final checks  String wantedCurrency = null;  double valueAfterConversion = convert(debitAccount2, debitAccount2.getBalance(), debitAccount2.getCurrency(), wantedCurrency);  // Actual call  bank.currencyExchange(debitAccount2, wantedCurrency);  // Retrieving the modified account from the bank  tmp = getBankAccountFromMap(debitAccount2, bankClient, bank.getClientBankAccountMap());  // Verify the accounts' updated information  Assert.assertEquals(debitAccount2.getCurrency(), tmp.getCurrency());  Assert.assertEquals(valueAfterConversion, tmp.getBalance(), Math.ulp(valueAfterConversion));  // Check the messages. The logs are really detailed so we have to check if our string is in them  Assert.assertTrue(outputStreamCaptor.toString().trim().contains("Nu se poate face conversia! Contul " + tmp.getIBAN() + " a fost inchis!"));  }  @Test  public void testCurrencyExchange\_C224 () throws BankAccountException {  //Integrity checks  BankAccount tmp = getBankAccountFromMap(debitAccount2, bankClient, bank.getClientBankAccountMap());  Assert.assertEquals(debitAccount2, tmp);  // Prepping for the final checks  String wantedCurrency = "Euro";  double valueAfterConversion = convert(debitAccount2, debitAccount2.getBalance(), debitAccount2.getCurrency(), wantedCurrency);  // Actual call  bank.currencyExchange(debitAccount2, wantedCurrency);  // Retrieving the modified account from the bank  tmp = getBankAccountFromMap(debitAccount2, bankClient, bank.getClientBankAccountMap());  // Verify the accounts' updated information  Assert.assertEquals(debitAccount2.getCurrency(), tmp.getCurrency());  Assert.assertEquals(valueAfterConversion, tmp.getBalance(), Math.ulp(valueAfterConversion));  // Check the messages. The logs are really detailed so we have to check if our string is in them  Assert.assertTrue(outputStreamCaptor.toString().trim().contains("Nu se poate face conversia! Contul " + tmp.getIBAN() + " a fost inchis!"));  }  @Test  public void testCurrencyExchange\_C234 () throws BankAccountException {  //Integrity checks  BankAccount tmp = getBankAccountFromMap(debitAccount2, bankClient, bank.getClientBankAccountMap());  Assert.assertEquals(debitAccount2, tmp);  // Prepping for the final checks  String wantedCurrency = "Yen";  double valueAfterConversion = convert(debitAccount2, debitAccount2.getBalance(), debitAccount2.getCurrency(), wantedCurrency);  // Actual call  bank.currencyExchange(debitAccount2, wantedCurrency);  // Retrieving the modified account from the bank  tmp = getBankAccountFromMap(debitAccount2, bankClient, bank.getClientBankAccountMap());  // Verify the accounts' updated information  Assert.assertEquals(debitAccount2.getCurrency(), tmp.getCurrency());  Assert.assertEquals(valueAfterConversion, tmp.getBalance(), Math.ulp(valueAfterConversion));  // Check the messages. The logs are really detailed so we have to check if our string is in them  Assert.assertTrue(outputStreamCaptor.toString().trim().contains("Nu se poate face conversia! Contul " + tmp.getIBAN() + " a fost inchis!"));  }  @Test  public void testCurrencyExchange\_C312 () throws BankAccountException {  //Integrity checks  BankAccount tmp = getBankAccountFromMap(savingsAccount1, bankClient, bank.getClientBankAccountMap());  Assert.assertEquals(savingsAccount1, tmp);  // Prepping for the final checks  String wantedCurrency = null;  double valueAfterConversion = convert(savingsAccount1, savingsAccount1.getBalance(), savingsAccount1.getCurrency(), wantedCurrency);  // Actual call  bank.currencyExchange(savingsAccount1, wantedCurrency);  // Retrieving the modified account from the bank  tmp = getBankAccountFromMap(savingsAccount1, bankClient, bank.getClientBankAccountMap());  // Verify the accounts' updated information  Assert.assertEquals(savingsAccount1.getCurrency(), tmp.getCurrency());  Assert.assertEquals(valueAfterConversion, tmp.getBalance(), Math.ulp(valueAfterConversion));  // Check the messages. The logs are really detailed so we have to check if our string is in them  Assert.assertTrue(outputStreamCaptor.toString().trim().contains("Conversie esuata!"));  }  @Test  public void testCurrencyExchange\_C321 () throws BankAccountException {  //Integrity checks  BankAccount tmp = getBankAccountFromMap(savingsAccount1, bankClient, bank.getClientBankAccountMap());  Assert.assertEquals(savingsAccount1, tmp);  // Prepping for the final checks  String wantedCurrency = "Lei";  double valueAfterConversion = convert(savingsAccount1, savingsAccount1.getBalance(), savingsAccount1.getCurrency(), wantedCurrency);  // Actual call  bank.currencyExchange(savingsAccount1, wantedCurrency);  // Retrieving the modified account from the bank  tmp = getBankAccountFromMap(savingsAccount1, bankClient, bank.getClientBankAccountMap());  // Verify the accounts' updated information  Assert.assertEquals(savingsAccount1.getCurrency(), tmp.getCurrency());  Assert.assertEquals(valueAfterConversion, tmp.getBalance(), Math.ulp(valueAfterConversion));  // Check the messages. The logs are really detailed so we have to check if our string is in them  Assert.assertTrue(outputStreamCaptor.toString().trim().contains("Conversie realizata!"));  }  @Test  public void testCurrencyExchange\_C332 () throws BankAccountException {  //Integrity checks  BankAccount tmp = getBankAccountFromMap(savingsAccount1, bankClient, bank.getClientBankAccountMap());  Assert.assertEquals(savingsAccount1, tmp);  // Prepping for the final checks  String wantedCurrency = "Yen";  double valueAfterConversion = convert(savingsAccount1, savingsAccount1.getBalance(), savingsAccount1.getCurrency(), wantedCurrency);  // Actual call  bank.currencyExchange(savingsAccount1, wantedCurrency);  // Retrieving the modified account from the bank  tmp = getBankAccountFromMap(savingsAccount1, bankClient, bank.getClientBankAccountMap());  // Verify the accounts' updated information  Assert.assertEquals(savingsAccount1.getCurrency(), tmp.getCurrency());  Assert.assertEquals(valueAfterConversion, tmp.getBalance(), Math.ulp(valueAfterConversion));  // Check the messages. The logs are really detailed so we have to check if our string is in them  Assert.assertTrue(outputStreamCaptor.toString().trim().contains("Conversie esuata!"));  }  @Test  public void testCurrencyExchange\_C414 () throws BankAccountException {  //Integrity checks  BankAccount tmp = getBankAccountFromMap(savingsAccount2, bankClient, bank.getClientBankAccountMap());  Assert.assertEquals(savingsAccount2, tmp);  // Prepping for the final checks  String wantedCurrency = null;  double valueAfterConversion = convert(savingsAccount2, savingsAccount2.getBalance(), savingsAccount2.getCurrency(), wantedCurrency);  // Actual call  bank.currencyExchange(savingsAccount2, wantedCurrency);  // Retrieving the modified account from the bank  tmp = getBankAccountFromMap(savingsAccount2, bankClient, bank.getClientBankAccountMap());  // Verify the accounts' updated information  Assert.assertEquals(savingsAccount2.getCurrency(), tmp.getCurrency());  Assert.assertEquals(valueAfterConversion, tmp.getBalance(), Math.ulp(valueAfterConversion));  // Check the messages. The logs are really detailed so we have to check if our string is in them  Assert.assertTrue(outputStreamCaptor.toString().trim().contains("Nu se poate face conversia! Contul " + tmp.getIBAN() + " a fost inchis!"));  }  @Test  public void testCurrencyExchange\_C424 () throws BankAccountException {  //Integrity checks  BankAccount tmp = getBankAccountFromMap(savingsAccount2, bankClient, bank.getClientBankAccountMap());  Assert.assertEquals(savingsAccount2, tmp);  // Prepping for the final checks  String wantedCurrency = "Euro";  double valueAfterConversion = convert(savingsAccount2, savingsAccount2.getBalance(), savingsAccount2.getCurrency(), wantedCurrency);  // Actual call  bank.currencyExchange(savingsAccount2, wantedCurrency);  // Retrieving the modified account from the bank  tmp = getBankAccountFromMap(savingsAccount2, bankClient, bank.getClientBankAccountMap());  // Verify the accounts' updated information  Assert.assertEquals(savingsAccount2.getCurrency(), tmp.getCurrency());  Assert.assertEquals(valueAfterConversion, tmp.getBalance(), Math.ulp(valueAfterConversion));  // Check the messages. The logs are really detailed so we have to check if our string is in them  Assert.assertTrue(outputStreamCaptor.toString().trim().contains("Nu se poate face conversia! Contul " + tmp.getIBAN() + " a fost inchis!"));  }  @Test  public void testCurrencyExchange\_C434 () throws BankAccountException {  //Integrity checks  BankAccount tmp = getBankAccountFromMap(savingsAccount2, bankClient, bank.getClientBankAccountMap());  Assert.assertEquals(savingsAccount2, tmp);  // Prepping for the final checks  String wantedCurrency = "Yen";  double valueAfterConversion = convert(savingsAccount2, savingsAccount2.getBalance(), savingsAccount2.getCurrency(), wantedCurrency);  // Actual call  bank.currencyExchange(savingsAccount2, wantedCurrency);  // Retrieving the modified account from the bank  tmp = getBankAccountFromMap(savingsAccount2, bankClient, bank.getClientBankAccountMap());  // Verify the accounts' updated information  Assert.assertEquals(savingsAccount2.getCurrency(), tmp.getCurrency());  Assert.assertEquals(valueAfterConversion, tmp.getBalance(), Math.ulp(valueAfterConversion));  // Check the messages. The logs are really detailed so we have to check if our string is in them  Assert.assertTrue(outputStreamCaptor.toString().trim().contains("Nu se poate face conversia! Contul " + tmp.getIBAN() + " a fost inchis!"));  }  @Test  public void testCurrencyExchange\_C513 () throws BankAccountException {  //Integrity checks  BankAccount tmp = getBankAccountFromMap(nonExistingAccount, bankClient, bank.getClientBankAccountMap());  Assert.assertEquals(nonExistingAccount, tmp);  // Prepping for the final checks  String wantedCurrency = null;  double valueAfterConversion = convert(nonExistingAccount, nonExistingAccount.getBalance(), nonExistingAccount.getCurrency(), wantedCurrency);  // Actual call  bank.currencyExchange(nonExistingAccount, wantedCurrency);  // Retrieving the modified account from the bank  tmp = getBankAccountFromMap(nonExistingAccount, bankClient, bank.getClientBankAccountMap());  // Verify the accounts' updated information  Assert.assertEquals(nonExistingAccount.getCurrency(), tmp.getCurrency());  Assert.assertEquals(valueAfterConversion, tmp.getBalance(), Math.ulp(valueAfterConversion));  // Check the messages. The logs are really detailed so we have to check if our string is in them  Assert.assertTrue(outputStreamCaptor.toString().trim().contains("Nu exista contul " + tmp.getIBAN()));  }  @Test  public void testCurrencyExchange\_C523 () throws BankAccountException {  //Integrity checks  BankAccount tmp = getBankAccountFromMap(nonExistingAccount, bankClient, bank.getClientBankAccountMap());  Assert.assertEquals(nonExistingAccount, tmp);  // Prepping for the final checks  String wantedCurrency = "Lei";  double valueAfterConversion = convert(nonExistingAccount, nonExistingAccount.getBalance(), nonExistingAccount.getCurrency(), wantedCurrency);  // Actual call  bank.currencyExchange(nonExistingAccount, wantedCurrency);  // Retrieving the modified account from the bank  tmp = getBankAccountFromMap(nonExistingAccount, bankClient, bank.getClientBankAccountMap());  // Verify the accounts' updated information  Assert.assertEquals(nonExistingAccount.getCurrency(), tmp.getCurrency());  Assert.assertEquals(valueAfterConversion, tmp.getBalance(), Math.ulp(valueAfterConversion));  // Check the messages. The logs are really detailed so we have to check if our string is in them  Assert.assertTrue(outputStreamCaptor.toString().trim().contains("Nu exista contul " + tmp.getIBAN()));  }  @Test  public void testCurrencyExchange\_C533 () throws BankAccountException {  //Integrity checks  BankAccount tmp = getBankAccountFromMap(nonExistingAccount, bankClient, bank.getClientBankAccountMap());  Assert.assertEquals(nonExistingAccount, tmp);  // Prepping for the final checks  String wantedCurrency = "Yen";  double valueAfterConversion = convert(nonExistingAccount, nonExistingAccount.getBalance(), nonExistingAccount.getCurrency(), wantedCurrency);  // Actual call  bank.currencyExchange(nonExistingAccount, wantedCurrency);  // Retrieving the modified account from the bank  tmp = getBankAccountFromMap(nonExistingAccount, bankClient, bank.getClientBankAccountMap());  // Verify the accounts' updated information  Assert.assertEquals(nonExistingAccount.getCurrency(), tmp.getCurrency());  Assert.assertEquals(valueAfterConversion, tmp.getBalance(), Math.ulp(valueAfterConversion));  // Check the messages. The logs are really detailed so we have to check if our string is in them  Assert.assertTrue(outputStreamCaptor.toString().trim().contains("Nu exista contul " + tmp.getIBAN()));  } |

## Analiza valorilor de frontiera

### Codul testat:

|  |
| --- |
| public void payLoan(BankAccount bankAccount, Loan loan, double value) throws TransactionException {  Timestamp.timestamp("Bank,payLoan");  Client dummy = new Client();  Loan dupe = new Loan();  Loan.setCounterLoanID(Loan.getCounterLoanID() - 1);  int c = 0, c1 = 0, c2 = 0;  for (Map.Entry<Client, List<Loan>> x : this.clientLoanMap.entrySet())  if (x.getValue().contains(loan)) {  dummy = x.getKey();  c++;  for (Loan z : x.getValue())  if (z.equals(loan))  dupe = z;  }  for (Map.Entry<Client, List<BankAccount>> y : this.clientBankAccountMap.entrySet())  if (y.getValue().contains(bankAccount)) {  c1++;  if (!(y.getKey().equals(dummy))) {  System.out.println("Cele doua produse nu apartin aceluiasi client");  } else {  if (!bankAccount.getClosingDate().equals("-"))  c2++;  else {  if (value < 0) {  System.out.println("Nu se poate efectua plata unei sume negative!");  } else {  if (value > bankAccount.getBalance()) {  System.out.println("Fonduri insuficiente!");  } else {  bankAccount.withdraw(CurrencyExchange.convertTransfer(value, bankAccount.getCurrency(), loan.getCurrency()));  System.out.print("Clientul a platit!");  dupe.payMonthlyRate(value, dummy.getCnp());  }  }  }  }  }  if (c == 0 && c1 != 0 )  System.out.println("Imprumutul nu exista");  else if (c != 0 && c1 == 0)  System.out.println("Contul nu exista.");  else if (c == 0)  System.out.println("Nici contul, nici imprumutul nu exista!");  else if (c2 != 0)  System.out.println("Nu se poate plati rata! Contul a fost inchis");  } |

### Domeniul de intrari:

Nota: Nu se poate trimite null ca referinta pentru BankAccount deoarece parametrul trebuie sa implementeze acea clasa abstracta (altfel primesc eroare la compilare -ambigous-)

!! Fie multimea conturilor existente ale bancii = P !!

* Un obiect ce implementeaza BankAccount

N\_1 = {x | x = DebitAccount ∈ P cu closingDate != "-"}

N\_2 = {x | x = DebitAccount ∈ P cu closingDate = "-"}

N\_3 = {x | x ∉ P}

* Un obiect ce implementeaza Loan

L\_1 = {y | y exista cu acelasi client ca x}

L\_2 = {y | y exista cu client diferit fata de x }

L\_3 = {y | y nu exista}

* O valoare pentru rata lunara de tip double

R\_1 = {z | z = 0 sau z = x.balance cu x ∈ P}

R\_2 = {z | z = -1}

R\_3 = {z | z = x.balance + 1 cu x ∈ P}

### Domeniul de iesiri:

C\_1 (x, y, z) = {c | "Cele doua produse nu apartin aceluiasi client!"}

C\_2 (x, y, z) = {c | "Nu se poate efectua plata unei sume negative!"}

C\_3 (x, y, z) = {c | "Fonduri insuficiente!"}

C\_4 (x, y, z) = {c | "Clientul a platit!"}

C\_5 (x, y, z) = {c | "Imprumutul nu exista!"}

C\_6 (x, y, z) = {c | "Contul nu exista!"}

C\_7 (x, y, z) = {c | "Nici contul, nici imprumutul nu exista!"}

C\_8 (x, y, z) = {c | "Nu se poate plati rata! Contul a fost inchis!"}

### Clasele de echivalenta:

C\_1118 = {(x, y, z, c) | n ∈ N\_1, y ∈ L\_1, z ∈ R\_1, c ∈ C\_8}  
C\_1128 = {(x, y, z, c) | n ∈ N\_1, y ∈ L\_1, z ∈ R\_2, c ∈ C\_8}  
C\_1138 = {(x, y, z, c) | n ∈ N\_1, y ∈ L\_1, z ∈ R\_3, c ∈ C\_8}  
C\_1211 = {(x, y, z, c) | n ∈ N\_1, y ∈ L\_2, z ∈ R\_1, c ∈ C\_1}  
C\_1221 = {(x, y, z, c) | n ∈ N\_1, y ∈ L\_2, z ∈ R\_2, c ∈ C\_1}  
C\_1231 = {(x, y, z, c) | n ∈ N\_1, y ∈ L\_2, z ∈ R\_3, c ∈ C\_1}  
C\_1315 = {(x, y, z, c) | n ∈ N\_1, y ∈ L\_3, z ∈ R\_1, c ∈ C\_5}  
C\_1325 = {(x, y, z, c) | n ∈ N\_1, y ∈ L\_3, z ∈ R\_1, c ∈ C\_5}  
C\_1335 = {(x, y, z, c) | n ∈ N\_1, y ∈ L\_3, z ∈ R\_1, c ∈ C\_5}  
  
C\_2114 = {(x, y, z, c) | n ∈ N\_2, y ∈ L\_1, z ∈ R\_1, c ∈ C\_4}  
C\_2122 = {(x, y, z, c) | n ∈ N\_2, y ∈ L\_1, z ∈ R\_2, c ∈ C\_2}  
C\_2133 = {(x, y, z, c) | n ∈ N\_2, y ∈ L\_1, z ∈ R\_3, c ∈ C\_3}  
C\_2211 = {(x, y, z, c) | n ∈ N\_2, y ∈ L\_2, z ∈ R\_1, c ∈ C\_1}  
C\_2221 = {(x, y, z, c) | n ∈ N\_2, y ∈ L\_2, z ∈ R\_2, c ∈ C\_1}  
C\_2231 = {(x, y, z, c) | n ∈ N\_2, y ∈ L\_2, z ∈ R\_3, c ∈ C\_1}  
C\_2315 = {(x, y, z, c) | n ∈ N\_2, y ∈ L\_3, z ∈ R\_1, c ∈ C\_5}  
C\_2325 = {(x, y, z, c) | n ∈ N\_2, y ∈ L\_3, z ∈ R\_2, c ∈ C\_5}  
C\_2335 = {(x, y, z, c) | n ∈ N\_2, y ∈ L\_3, z ∈ R\_3, c ∈ C\_5}  
  
C\_3116 = {(x, y, z, c) | n ∈ N\_3, y ∈ L\_1, z ∈ R\_1, c ∈ C\_6}  
C\_3126 = {(x, y, z, c) | n ∈ N\_3, y ∈ L\_1, z ∈ R\_2, c ∈ C\_6}  
C\_3136 = {(x, y, z, c) | n ∈ N\_3, y ∈ L\_1, z ∈ R\_3, c ∈ C\_6}  
C\_3216 = {(x, y, z, c) | n ∈ N\_3, y ∈ L\_2, z ∈ R\_1, c ∈ C\_6}  
C\_3226 = {(x, y, z, c) | n ∈ N\_3, y ∈ L\_2, z ∈ R\_2, c ∈ C\_6}  
C\_3236 = {(x, y, z, c) | n ∈ N\_3, y ∈ L\_2, z ∈ R\_3, c ∈ C\_6}  
C\_3317 = {(x, y, z, c) | n ∈ N\_3, y ∈ L\_3, z ∈ R\_1, c ∈ C\_7}  
C\_3327 = {(x, y, z, c) | n ∈ N\_3, y ∈ L\_3, z ∈ R\_2, c ∈ C\_7}  
C\_3337 = {(x, y, z, c) | n ∈ N\_3, y ∈ L\_3, z ∈ R\_3, c ∈ C\_7}

|  |
| --- |
| @Test public void testPayLoan\_C1118() throws TransactionException {   double value = 156788765;   BankAccount tmp = getBankAccountFromMap(debitAccount2, bankClient, bank.getClientBankAccountMap());  double balance1 = tmp.getBalance();  double balance2 = bankLoan1.getValue();   bank.payLoan(debitAccount2, bankLoan1, value);   Assert.*assertEquals*(debitAccount2.getBalance(), balance1,0);  Assert.*assertEquals*(bankLoan1.getValue(), balance2,0);   Assert.*assertTrue*(outputStreamCaptor.toString().trim().contains("Nu se poate plati rata! Contul a fost inchis")); }  @Test public void testPayLoan\_C1128() throws TransactionException {   double value = -1;   BankAccount tmp = getBankAccountFromMap(debitAccount2, bankClient, bank.getClientBankAccountMap());  double balance1 = tmp.getBalance();  double balance2 = bankLoan1.getValue();   bank.payLoan(debitAccount2, bankLoan1, value);   Assert.*assertEquals*(debitAccount2.getBalance(), balance1,0);  Assert.*assertEquals*(bankLoan1.getValue(), balance2,0);   Assert.*assertTrue*(outputStreamCaptor.toString().trim().contains("Nu se poate plati rata! Contul a fost inchis")); }  @Test public void testPayLoan\_C1138() throws TransactionException {   BankAccount tmp = getBankAccountFromMap(debitAccount2, bankClient, bank.getClientBankAccountMap());  double balance1 = tmp.getBalance();  double balance2 = bankLoan1.getValue();  double value = balance1 + 1;   bank.payLoan(debitAccount2, bankLoan1, value);   Assert.*assertEquals*(debitAccount2.getBalance(), balance1,0);  Assert.*assertEquals*(bankLoan1.getValue(), balance2,0);   Assert.*assertTrue*(outputStreamCaptor.toString().trim().contains("Nu se poate plati rata! Contul a fost inchis")); }  @Test public void testPayLoan\_C1211() throws TransactionException {   BankAccount tmp = getBankAccountFromMap(debitAccount2, bankClient, bank.getClientBankAccountMap());  double balance1 = tmp.getBalance();  double balance2 = bankLoan2.getValue();  double value = balance1;   bank.payLoan(debitAccount2, bankLoan2, value);   Assert.*assertEquals*(debitAccount2.getBalance(), balance1,0);  Assert.*assertEquals*(bankLoan2.getValue(), balance2,0);   Assert.*assertTrue*(outputStreamCaptor.toString().trim().contains("Cele doua produse nu apartin aceluiasi client")); }  @Test public void testPayLoan\_C1221() throws TransactionException {   BankAccount tmp = getBankAccountFromMap(debitAccount2, bankClient, bank.getClientBankAccountMap());  double balance1 = tmp.getBalance();  double balance2 = bankLoan2.getValue();  double value = -1;   bank.payLoan(debitAccount2, bankLoan2, value);   Assert.*assertEquals*(debitAccount2.getBalance(), balance1,0);  Assert.*assertEquals*(bankLoan2.getValue(), balance2,0);   Assert.*assertTrue*(outputStreamCaptor.toString().trim().contains("Cele doua produse nu apartin aceluiasi client")); }  @Test public void testPayLoan\_C1231() throws TransactionException {   BankAccount tmp = getBankAccountFromMap(debitAccount2, bankClient, bank.getClientBankAccountMap());  double balance1 = tmp.getBalance();  double balance2 = bankLoan2.getValue();  double value = balance1 + 1;   bank.payLoan(debitAccount2, bankLoan2, value);   Assert.*assertEquals*(debitAccount2.getBalance(), balance1,0);  Assert.*assertEquals*(bankLoan2.getValue(), balance2,0);   Assert.*assertTrue*(outputStreamCaptor.toString().trim().contains("Cele doua produse nu apartin aceluiasi client")); }  @Test public void testPayLoan\_C1315() throws TransactionException {   BankAccount tmp = getBankAccountFromMap(debitAccount2, bankClient, bank.getClientBankAccountMap());  double balance1 = tmp.getBalance();  double balance2 = nonExistingBankLoan.getValue();  double value = balance1;   bank.payLoan(debitAccount2, nonExistingBankLoan, value);   Assert.*assertEquals*(debitAccount2.getBalance(), balance1,0);  Assert.*assertEquals*(nonExistingBankLoan.getValue(), balance2,0);   Assert.*assertTrue*(outputStreamCaptor.toString().trim().contains("Imprumutul nu exista")); }  @Test public void testPayLoan\_C1325() throws TransactionException {   BankAccount tmp = getBankAccountFromMap(debitAccount2, bankClient, bank.getClientBankAccountMap());  double balance1 = tmp.getBalance();  double balance2 = nonExistingBankLoan.getValue();  double value = -1;   bank.payLoan(debitAccount2, nonExistingBankLoan, value);   Assert.*assertEquals*(debitAccount2.getBalance(), balance1,0);  Assert.*assertEquals*(nonExistingBankLoan.getValue(), balance2,0);   Assert.*assertTrue*(outputStreamCaptor.toString().trim().contains("Imprumutul nu exista")); }  @Test public void testPayLoan\_C1335() throws TransactionException {   BankAccount tmp = getBankAccountFromMap(debitAccount2, bankClient, bank.getClientBankAccountMap());  double balance1 = tmp.getBalance();  double balance2 = nonExistingBankLoan.getValue();  double value = balance1 + 1;   bank.payLoan(debitAccount2, nonExistingBankLoan, value);   Assert.*assertEquals*(debitAccount2.getBalance(), balance1,0);  Assert.*assertEquals*(nonExistingBankLoan.getValue(), balance2,0);   Assert.*assertTrue*(outputStreamCaptor.toString().trim().contains("Imprumutul nu exista")); }  @Test public void testPayLoan\_C2114() throws ProviderException, TransactionException {   double value = 156788765;   BankAccount tmp = getBankAccountFromMap(debitAccount1, bankClient, bank.getClientBankAccountMap());  double balance1 = tmp.getBalance();  double balance2 = bankLoan1.getValue();   bank.payLoan(debitAccount1, bankLoan1, value);   Assert.*assertNotEquals*(debitAccount1.getBalance(), balance1);  Assert.*assertNotEquals*(bankLoan1.getValue(), balance2);   Assert.*assertTrue*(outputStreamCaptor.toString().trim().contains("Clientul a platit")); }  @Test public void testPayLoan\_C2122() throws ProviderException, TransactionException {   double value = -1;   BankAccount tmp = getBankAccountFromMap(debitAccount1, bankClient, bank.getClientBankAccountMap());  double balance1 = tmp.getBalance();  double balance2 = bankLoan1.getValue();   bank.payLoan(debitAccount1, bankLoan1, value);   Assert.*assertEquals*(debitAccount1.getBalance(), balance1,0);  Assert.*assertEquals*(bankLoan1.getValue(), balance2,0);   Assert.*assertTrue*(outputStreamCaptor.toString().trim().contains("Nu se poate efectua plata unei sume negative")); }  @Test public void testPayLoan\_C2133() throws ProviderException, TransactionException {   BankAccount tmp = getBankAccountFromMap(debitAccount1, bankClient, bank.getClientBankAccountMap());  double balance1 = tmp.getBalance();  double balance2 = bankLoan1.getValue();  double value = balance1 + 1;   bank.payLoan(debitAccount1, bankLoan1, value);   Assert.*assertEquals*(debitAccount1.getBalance(), balance1,0);  Assert.*assertEquals*(bankLoan1.getValue(), balance2,0);   Assert.*assertTrue*(outputStreamCaptor.toString().trim().contains("Fonduri insuficiente")); }  @Test public void testPayLoan\_C2211() throws ProviderException, TransactionException {   BankAccount tmp = getBankAccountFromMap(debitAccount1, bankClient, bank.getClientBankAccountMap());  double balance1 = tmp.getBalance();  double balance2 = bankLoan2.getValue();  double value = balance1;   bank.payLoan(debitAccount1, bankLoan2, value);   Assert.*assertEquals*(debitAccount1.getBalance(), balance1,0);  Assert.*assertEquals*(bankLoan2.getValue(), balance2,0);   Assert.*assertTrue*(outputStreamCaptor.toString().trim().contains("Cele doua produse nu apartin aceluiasi client")); }  @Test public void testPayLoan\_C2221() throws ProviderException, TransactionException {   BankAccount tmp = getBankAccountFromMap(debitAccount1, bankClient, bank.getClientBankAccountMap());  double balance1 = tmp.getBalance();  double balance2 = bankLoan2.getValue();  double value = -1;   bank.payLoan(debitAccount1, bankLoan2, value);   Assert.*assertEquals*(debitAccount1.getBalance(), balance1,0);  Assert.*assertEquals*(bankLoan2.getValue(), balance2,0);   Assert.*assertTrue*(outputStreamCaptor.toString().trim().contains("Cele doua produse nu apartin aceluiasi client")); }  @Test public void testPayLoan\_C2231() throws ProviderException, TransactionException {   BankAccount tmp = getBankAccountFromMap(debitAccount1, bankClient, bank.getClientBankAccountMap());  double balance1 = tmp.getBalance();  double balance2 = bankLoan2.getValue();  double value = balance1 + 1;   bank.payLoan(debitAccount1, bankLoan2, value);   Assert.*assertEquals*(debitAccount1.getBalance(), balance1,0);  Assert.*assertEquals*(bankLoan2.getValue(), balance2,0);   Assert.*assertTrue*(outputStreamCaptor.toString().trim().contains("Cele doua produse nu apartin aceluiasi client")); }  @Test public void testPayLoan\_C2315() throws ProviderException, TransactionException {   BankAccount tmp = getBankAccountFromMap(debitAccount1, bankClient, bank.getClientBankAccountMap());  double balance1 = tmp.getBalance();  double balance2 = nonExistingBankLoan.getValue();  double value = balance1;   bank.payLoan(debitAccount1, nonExistingBankLoan, value);   Assert.*assertEquals*(debitAccount1.getBalance(), balance1,0);  Assert.*assertEquals*(nonExistingBankLoan.getValue(), balance2,0);   Assert.*assertTrue*(outputStreamCaptor.toString().trim().contains("Imprumutul nu exista")); }  @Test public void testPayLoan\_C2325() throws ProviderException, TransactionException {   BankAccount tmp = getBankAccountFromMap(debitAccount1, bankClient, bank.getClientBankAccountMap());  double balance1 = tmp.getBalance();  double balance2 = nonExistingBankLoan.getValue();  double value = -1;   bank.payLoan(debitAccount1, nonExistingBankLoan, value);   Assert.*assertEquals*(debitAccount1.getBalance(), balance1,0);  Assert.*assertEquals*(nonExistingBankLoan.getValue(), balance2,0);   Assert.*assertTrue*(outputStreamCaptor.toString().trim().contains("Imprumutul nu exista")); }  @Test public void testPayLoan\_C2335() throws ProviderException, TransactionException {   BankAccount tmp = getBankAccountFromMap(debitAccount1, bankClient, bank.getClientBankAccountMap());  double balance1 = tmp.getBalance();  double balance2 = nonExistingBankLoan.getValue();  double value = balance1 + 1;   bank.payLoan(debitAccount1, nonExistingBankLoan, value);   Assert.*assertEquals*(debitAccount1.getBalance(), balance1,0);  Assert.*assertEquals*(nonExistingBankLoan.getValue(), balance2,0);   Assert.*assertTrue*(outputStreamCaptor.toString().trim().contains("Imprumutul nu exista")); }  @Test public void testPayLoan\_C3116() throws ProviderException, TransactionException {   BankAccount tmp = getBankAccountFromMap(nonExistingAccount, bankClient, bank.getClientBankAccountMap());  double balance1 = tmp.getBalance();  double balance2 = bankLoan1.getValue();  double value = balance1;   bank.payLoan(nonExistingAccount, bankLoan1, value);   Assert.*assertEquals*(nonExistingAccount.getBalance(), balance1,0);  Assert.*assertEquals*(bankLoan1.getValue(), balance2,0);   Assert.*assertTrue*(outputStreamCaptor.toString().trim().contains("Contul nu exista")); }  @Test public void testPayLoan\_C3126() throws ProviderException, TransactionException {   BankAccount tmp = getBankAccountFromMap(nonExistingAccount, bankClient, bank.getClientBankAccountMap());  double balance1 = tmp.getBalance();  double balance2 = bankLoan1.getValue();  double value = -1;   bank.payLoan(nonExistingAccount, bankLoan1, value);   Assert.*assertEquals*(nonExistingAccount.getBalance(), balance1,0);  Assert.*assertEquals*(bankLoan1.getValue(), balance2,0);   Assert.*assertTrue*(outputStreamCaptor.toString().trim().contains("Contul nu exista")); }  @Test public void testPayLoan\_C3136() throws ProviderException, TransactionException {   BankAccount tmp = getBankAccountFromMap(nonExistingAccount, bankClient, bank.getClientBankAccountMap());  double balance1 = tmp.getBalance();  double balance2 = bankLoan1.getValue();  double value = balance1 + 1;   bank.payLoan(nonExistingAccount, bankLoan1, value);   Assert.*assertEquals*(nonExistingAccount.getBalance(), balance1,0);  Assert.*assertEquals*(bankLoan1.getValue(), balance2,0);   Assert.*assertTrue*(outputStreamCaptor.toString().trim().contains("Contul nu exista")); }  @Test public void testPayLoan\_C3216() throws ProviderException, TransactionException {   BankAccount tmp = getBankAccountFromMap(nonExistingAccount, bankClient, bank.getClientBankAccountMap());  double balance1 = tmp.getBalance();  double balance2 = bankLoan2.getValue();  double value = balance1;   bank.payLoan(nonExistingAccount, bankLoan2, value);   Assert.*assertEquals*(nonExistingAccount.getBalance(), balance1,0);  Assert.*assertEquals*(bankLoan2.getValue(), balance2,0);   Assert.*assertTrue*(outputStreamCaptor.toString().trim().contains("Contul nu exista")); }  @Test public void testPayLoan\_C3226() throws ProviderException, TransactionException {   BankAccount tmp = getBankAccountFromMap(nonExistingAccount, bankClient, bank.getClientBankAccountMap());  double balance1 = tmp.getBalance();  double balance2 = bankLoan2.getValue();  double value = -1;   bank.payLoan(nonExistingAccount, bankLoan2, value);   Assert.*assertEquals*(nonExistingAccount.getBalance(), balance1,0);  Assert.*assertEquals*(bankLoan2.getValue(), balance2,0);   Assert.*assertTrue*(outputStreamCaptor.toString().trim().contains("Contul nu exista")); }  @Test public void testPayLoan\_C3236() throws ProviderException, TransactionException {   BankAccount tmp = getBankAccountFromMap(nonExistingAccount, bankClient, bank.getClientBankAccountMap());  double balance1 = tmp.getBalance();  double balance2 = bankLoan2.getValue();  double value = balance1 + 1;   bank.payLoan(nonExistingAccount, bankLoan2, value);   Assert.*assertEquals*(nonExistingAccount.getBalance(), balance1,0);  Assert.*assertEquals*(bankLoan2.getValue(), balance2,0);   Assert.*assertTrue*(outputStreamCaptor.toString().trim().contains("Contul nu exista")); }  @Test public void testPayLoan\_C3317() throws ProviderException, TransactionException {   BankAccount tmp = getBankAccountFromMap(nonExistingAccount, bankClient, bank.getClientBankAccountMap());  double balance1 = tmp.getBalance();  double balance2 = nonExistingBankLoan.getValue();  double value = balance1;   bank.payLoan(nonExistingAccount, nonExistingBankLoan, value);   Assert.*assertEquals*(nonExistingAccount.getBalance(), balance1,0);  Assert.*assertEquals*(nonExistingBankLoan.getValue(), balance2,0);   Assert.*assertTrue*(outputStreamCaptor.toString().trim().contains("Nici contul, nici imprumutul nu exista!")); }  @Test public void testPayLoan\_C3327() throws ProviderException, TransactionException {   BankAccount tmp = getBankAccountFromMap(nonExistingAccount, bankClient, bank.getClientBankAccountMap());  double balance1 = tmp.getBalance();  double balance2 = nonExistingBankLoan.getValue();  double value = -1;   bank.payLoan(nonExistingAccount, nonExistingBankLoan, value);   Assert.*assertEquals*(nonExistingAccount.getBalance(), balance1,0);  Assert.*assertEquals*(nonExistingBankLoan.getValue(), balance2,0);   Assert.*assertTrue*(outputStreamCaptor.toString().trim().contains("Nici contul, nici imprumutul nu exista!")); }  @Test public void testPayLoan\_C3337() throws ProviderException, TransactionException {   BankAccount tmp = getBankAccountFromMap(nonExistingAccount, bankClient, bank.getClientBankAccountMap());  double balance1 = tmp.getBalance();  double balance2 = nonExistingBankLoan.getValue();  double value = balance1 + 1;   bank.payLoan(nonExistingAccount, nonExistingBankLoan, value);   Assert.*assertEquals*(nonExistingAccount.getBalance(), balance1,0);  Assert.*assertEquals*(nonExistingBankLoan.getValue(), balance2,0);   Assert.*assertTrue*(outputStreamCaptor.toString().trim().contains("Nici contul, nici imprumutul nu exista!")); }  } |

## Partitionarea de echivalenta

### Codul testat:

|  |
| --- |
| public void paymentUtilies(String Sender, String Receiver, double value) throws ProviderException, TransactionException {  Timestamp.*timestamp*("Bank,paymentUtilies");  int c = 0, c1 = 0;  for (Map.Entry<Client, List<BankAccount>> x : this.clientBankAccountMap.entrySet()) {  for (BankAccount y : x.getValue()) {  if (y.getIBAN().equals(Sender)) {  c++;  if (!y.getClosingDate().equals("-"))  c1++;  else {  System.*out*.print("\tClientul " + x.getKey().getFirstName() + " " + x.getKey().getLastName());  y.paymentUtilies(Receiver, value);  }  }  }  }  if (c == 0)  System.*out*.println("Nu exista contul " + Sender);  else if (c1 != 0)  System.*out*.println("Nu se poate plati providerul! Contul " + Sender + " a fost inchis!"); } |

### Domeniul de intrari:

Nota: Nu se poate trimite null ca referinta pentru BankAccount deoarece parametrul trebuie sa implementeze acea clasa abstracta (altfel primesc eroare la compilare -ambigous-)

!! Fie multimea conturilor existente ale bancii = P !!

* Un obiect ce implementeaza BankAccount

N\_1 = {x | x = DebitAccount ∈ P cu closingDate != "-"}

N\_2 = {x | x = DebitAccount ∈ P cu closingDate = "-"}

N\_3 = {x | x ∉ P}

* Un obiect Provider

L\_1 = {y | y exista}

L\_2 = {y | y nu exista}

* O valoare de tip double pentru rata lunara

R\_1 = {z | x ∈ P}

R\_2 = {z | z < 0}

R\_3 = {z | z > x.balance cu x ∈ P}

### Domeniul de iesiri:

### C\_1(x, y, z) = {c | "Payment ok"}

### C\_2(x, y, z) = {c | "Nu exista contul " + Sender"}

### C\_3(x, y, z) = {c | "Fonduri insuficiente!"}

### C\_4(x, y, z) = {c | "Nu se poate plati providerul! Contul " + Sender + " a fost inchis!"}

### C\_5(x, y, z) = {c | "Nu exista providerul cu " + Receiver + ".""}

### C\_6(x, y, z) = {c | "Suma invalida"}

### Clasele de echivalenta:

C\_1111 = {(x, y, z, c) | n ∈ N\_1, y ∈ L\_1, z ∈ R\_1, c ∈ C\_1}

C\_1126 = {(x, y, z, c) | n ∈ N\_1, y ∈ L\_1, z ∈ R\_2, c ∈ C\_6 }

C\_1133 = {(x, y, z, c) | n ∈ N\_1, y ∈ L\_1, z ∈ R\_3, c ∈ C\_3}

C\_1215 = {(x, y, z, c) | n ∈ N\_1, y ∈ L\_2, z ∈ R\_1, c ∈ C\_5}

C\_1225 = {(x, y, z, c) | n ∈ N\_1, y ∈ L\_2, z ∈ R\_2, c ∈ C\_3}

C\_1235 = {(x, y, z, c) | n ∈ N\_1, y ∈ L\_2, z ∈ R\_3, c ∈ C\_3}

C\_2114 = {(x, y, z, c) | n ∈ N\_2, y ∈ L\_1, z ∈ R\_1, c ∈ C\_4}

C\_2124 = {(x, y, z, c) | n ∈ N\_2, y ∈ L\_1, z ∈ R\_2, c ∈ C\_4}

C\_2134 = {(x, y, z, c) | n ∈ N\_2, y ∈ L\_1, z ∈ R\_3, c ∈ C\_4}

C\_2214 = {(x, y, z, c) | n ∈ N\_2, y ∈ L\_2, z ∈ R\_1, c ∈ C\_4}

C\_2224 = {(x, y, z, c) | n ∈ N\_2, y ∈ L\_2, z ∈ R\_2, c ∈ C\_4}

C\_2234 = {(x, y, z, c) | n ∈ N\_2, y ∈ L\_2, z ∈ R\_3, c ∈ C\_4}

C\_3112 = {(x, y, z, c) | n ∈ N\_3, y ∈ L\_1, z ∈ R\_1, c ∈ C\_2}

C\_3122 = {(x, y, z, c) | n ∈ N\_3, y ∈ L\_1, z ∈ R\_2, c ∈ C\_2}

C\_3132 = {(x, y, z, c) | n ∈ N\_3, y ∈ L\_1, z ∈ R\_3, c ∈ C\_2}

C\_3212 = {(x, y, z, c) | n ∈ N\_3, y ∈ L\_2, z ∈ R\_1, c ∈ C\_2}

C\_3222 = {(x, y, z, c) | n ∈ N\_3, y ∈ L\_2, z ∈ R\_2, c ∈ C\_2}

C\_3232 = {(x, y, z, c) | n ∈ N\_3, y ∈ L\_2, z ∈ R\_3, c ∈ C\_2}

|  |
| --- |
| @Test  public void testPaymentUtilities\_C1111() throws ProviderException, TransactionException {  String sender = "RO59RZBR0000065122344800";  String receiver = "RO59RZBR0000068222375802";  double value = 1;  BankAccount tmp = getBankAccountFromMap(debitAccount1, bankClient, bank.getClientBankAccountMap());  double balance1 = tmp.getBalance();  int index = ToProviders.providerList.indexOf(provider);  bank.paymentUtilies(sender, receiver, value);  double balance2 = ToProviders.providerList.get(index).getBalance();  Assert.assertNotEquals(debitAccount1.getBalance(), balance1);  Assert.assertNotEquals(debitAccount2.getBalance(), balance2);  Assert.assertTrue(outputStreamCaptor.toString().trim().contains("Payment ok"));  }  @Test  public void testPaymentUtilities\_C1133() throws ProviderException, TransactionException {  String sender = "RO59RZBR0000065122344800";  String receiver= "RO59RZBR0000068222375802";  double value = 23456787;  BankAccount tmp = getBankAccountFromMap(debitAccount1, bankClient, bank.getClientBankAccountMap());  double balance1 = tmp.getBalance();  int index= ToProviders.providerList.indexOf(provider);  bank.paymentUtilies(sender,receiver,value);  double balance2 = ToProviders.providerList.get(index).getBalance();  Assert.assertNotEquals(debitAccount1.getBalance(), balance1 );  Assert.assertNotEquals(debitAccount2.getBalance(), balance2);  Assert.assertTrue(outputStreamCaptor.toString().trim().contains("Fonduri insuficiente"));  }  @Test  public void testPaymentUtilities\_C1126() throws ProviderException, TransactionException {  String sender = "RO59RZBR0000065122344800";  String receiver= "RO59RZBR0000068222375802";  double value = -1;  bank.paymentUtilies(sender,receiver,value);  Assert.assertTrue(outputStreamCaptor.toString().trim().contains("Suma invalida"));  }  @Test  public void testPaymentUtilities\_C1215() throws ProviderException, TransactionException {  String sender = "RO59RZBR0000065122344800";  String receiver= "RO59RZBR000006822237580";  double value = 1;  BankAccount tmp = getBankAccountFromMap(debitAccount1, bankClient, bank.getClientBankAccountMap());  double balance1 = tmp.getBalance();  int index= ToProviders.providerList.indexOf(provider);  bank.paymentUtilies(sender,receiver,value);  double balance2 = ToProviders.providerList.get(index).getBalance();  Assert.assertEquals(debitAccount1.getBalance(), balance1, 0);  Assert.assertEquals(debitAccount2.getBalance(), balance2, 0);  Assert.assertTrue(outputStreamCaptor.toString().trim().contains("Nu exista providerul cu " + receiver + "."));  }  @Test  public void testPaymentUtilities\_C1225() throws ProviderException, TransactionException {  String sender = "RO59RZBR0000065122344800";  String receiver= "RO59RZBR000006822237580";  double value = -1;  BankAccount tmp = getBankAccountFromMap(debitAccount1, bankClient, bank.getClientBankAccountMap());  double balance1 = tmp.getBalance();  int index= ToProviders.providerList.indexOf(provider);  bank.paymentUtilies(sender,receiver,value);  double balance2 = ToProviders.providerList.get(index).getBalance();  Assert.assertEquals(debitAccount1.getBalance(), balance1, 0);  Assert.assertEquals(debitAccount2.getBalance(), balance2, 0);  Assert.assertTrue(outputStreamCaptor.toString().trim().contains("Nu exista providerul cu " + receiver + "."));  }  @Test  public void testPaymentUtilities\_C1235() throws ProviderException, TransactionException {  String sender = "RO59RZBR0000065122344800";  String receiver= "RO59RZBR000006822237580";  double value = 234567894;  BankAccount tmp = getBankAccountFromMap(debitAccount1, bankClient, bank.getClientBankAccountMap());  double balance1 = tmp.getBalance();  int index= ToProviders.providerList.indexOf(provider);  bank.paymentUtilies(sender,receiver,value);  double balance2 = ToProviders.providerList.get(index).getBalance();  Assert.assertEquals(debitAccount1.getBalance(), balance1, 0);  Assert.assertEquals(debitAccount2.getBalance(), balance2, 0);  Assert.assertTrue(outputStreamCaptor.toString().trim().contains("Nu exista providerul cu " + receiver + "."));  }  @Test  public void testPaymentUtilities\_C2114() throws ProviderException, TransactionException {  String sender = "RO69RZBR0000068222375804";  String receiver= "RO59RZBR0000068222375802";  double value = 1;  BankAccount tmp = getBankAccountFromMap(debitAccount1, bankClient, bank.getClientBankAccountMap());  double balance1 = tmp.getBalance();  int index= ToProviders.providerList.indexOf(provider);  bank.paymentUtilies(sender,receiver,value);  double balance2 = ToProviders.providerList.get(index).getBalance();  Assert.assertEquals(debitAccount1.getBalance(), balance1, 0);  Assert.assertEquals(debitAccount2.getBalance(), balance2, 0);  Assert.assertTrue(outputStreamCaptor.toString().trim().contains("Nu se poate plati providerul! Contul " + sender + " a fost inchis!"));  }  @Test  public void testPaymentUtilities\_C2124() throws ProviderException, TransactionException {  String sender = "RO69RZBR0000068222375804";  String receiver= "RO59RZBR0000068222375802";  double value = -1;  BankAccount tmp = getBankAccountFromMap(debitAccount1, bankClient, bank.getClientBankAccountMap());  double balance1 = tmp.getBalance();  int index= ToProviders.providerList.indexOf(provider);  bank.paymentUtilies(sender,receiver,value);  double balance2 = ToProviders.providerList.get(index).getBalance();  Assert.assertEquals(debitAccount1.getBalance(), balance1, 0);  Assert.assertEquals(debitAccount2.getBalance(), balance2, 0);  Assert.assertTrue(outputStreamCaptor.toString().trim().contains("Nu se poate plati providerul! Contul " + sender + " a fost inchis!"));  }  @Test  public void testPaymentUtilities\_C2134() throws ProviderException, TransactionException {  String sender = "RO69RZBR0000068222375804";  String receiver= "RO59RZBR0000068222375802";  double value = 234567890;  BankAccount tmp = getBankAccountFromMap(debitAccount1, bankClient, bank.getClientBankAccountMap());  double balance1 = tmp.getBalance();  int index= ToProviders.providerList.indexOf(provider);  bank.paymentUtilies(sender,receiver,value);  double balance2 = ToProviders.providerList.get(index).getBalance();  Assert.assertEquals(debitAccount1.getBalance(), balance1, 0);  Assert.assertEquals(debitAccount2.getBalance(), balance2, 0);  Assert.assertTrue(outputStreamCaptor.toString().trim().contains("Nu se poate plati providerul! Contul " + sender + " a fost inchis!"));  }  @Test  public void testPaymentUtilities\_C2214() throws ProviderException, TransactionException {  String sender = "RO69RZBR0000068222375804";  String receiver= "RO59RZBR0000068222375802";  double value = 1;  BankAccount tmp = getBankAccountFromMap(debitAccount1, bankClient, bank.getClientBankAccountMap());  double balance1 = tmp.getBalance();  int index= ToProviders.providerList.indexOf(provider);  bank.paymentUtilies(sender,receiver,value);  double balance2 = ToProviders.providerList.get(index).getBalance();  Assert.assertEquals(debitAccount1.getBalance(), balance1, 0);  Assert.assertEquals(debitAccount2.getBalance(), balance2, 0);  Assert.assertTrue(outputStreamCaptor.toString().trim().contains("Nu se poate plati providerul! Contul " + sender + " a fost inchis!"));  }  @Test  public void testPaymentUtilities\_C2224() throws ProviderException, TransactionException {  String sender = "RO69RZBR0000068222375804";  String receiver= "RO59RZBR0000068222375802";  double value = -1;  BankAccount tmp = getBankAccountFromMap(debitAccount1, bankClient, bank.getClientBankAccountMap());  double balance1 = tmp.getBalance();  int index= ToProviders.providerList.indexOf(provider);  bank.paymentUtilies(sender,receiver,value);  double balance2 = ToProviders.providerList.get(index).getBalance();  Assert.assertEquals(debitAccount1.getBalance(), balance1, 0);  Assert.assertEquals(debitAccount2.getBalance(), balance2, 0);  Assert.assertTrue(outputStreamCaptor.toString().trim().contains("Nu se poate plati providerul! Contul " + sender + " a fost inchis!"));  }  @Test  public void testPaymentUtilities\_C2234() throws ProviderException, TransactionException {  String sender = "RO69RZBR0000068222375804";  String receiver= "RO59RZBR0000068222375802";  double value = 234567787;  BankAccount tmp = getBankAccountFromMap(debitAccount1, bankClient, bank.getClientBankAccountMap());  double balance1 = tmp.getBalance();  int index= ToProviders.providerList.indexOf(provider);  bank.paymentUtilies(sender,receiver,value);  double balance2 = ToProviders.providerList.get(index).getBalance();  Assert.assertEquals(debitAccount1.getBalance(), balance1, 0);  Assert.assertEquals(debitAccount2.getBalance(), balance2, 0);  Assert.assertTrue(outputStreamCaptor.toString().trim().contains("Nu se poate plati providerul! Contul " + sender + " a fost inchis!"));  }  @Test  public void testPaymentUtilities\_C3112() throws ProviderException, TransactionException {  String sender = "RO69RZBR00000682223758045";  String receiver= "RO59RZBR0000068222375802";  double value = 1;  BankAccount tmp = getBankAccountFromMap(debitAccount1, bankClient, bank.getClientBankAccountMap());  double balance1 = tmp.getBalance();  int index= ToProviders.providerList.indexOf(provider);  bank.paymentUtilies(sender,receiver,value);  double balance2 = ToProviders.providerList.get(index).getBalance();  Assert.assertEquals(debitAccount1.getBalance(), balance1, 0);  Assert.assertEquals(debitAccount2.getBalance(), balance2, 0);  Assert.assertTrue(outputStreamCaptor.toString().trim().contains("Nu exista contul " + sender));  }  @Test  public void testPaymentUtilities\_C3122() throws ProviderException, TransactionException {  String sender = "RO69RZBR00000682223758045";  String receiver= "RO59RZBR0000068222375802";  double value = -1;  BankAccount tmp = getBankAccountFromMap(debitAccount1, bankClient, bank.getClientBankAccountMap());  double balance1 = tmp.getBalance();  int index= ToProviders.providerList.indexOf(provider);  bank.paymentUtilies(sender,receiver,value);  double balance2 = ToProviders.providerList.get(index).getBalance();  Assert.assertEquals(debitAccount1.getBalance(), balance1, 0);  Assert.assertEquals(debitAccount2.getBalance(), balance2, 0);  Assert.assertTrue(outputStreamCaptor.toString().trim().contains("Nu exista contul " + sender));  }  @Test  public void testPaymentUtilities\_C3132() throws ProviderException, TransactionException {  String sender = "RO69RZBR00000682223758045";  String receiver= "RO59RZBR0000068222375802";  double value = 234567787;  BankAccount tmp = getBankAccountFromMap(debitAccount1, bankClient, bank.getClientBankAccountMap());  double balance1 = tmp.getBalance();  int index= ToProviders.providerList.indexOf(provider);  bank.paymentUtilies(sender,receiver,value);  double balance2 = ToProviders.providerList.get(index).getBalance();  Assert.assertEquals(debitAccount1.getBalance(), balance1, 0);  Assert.assertEquals(debitAccount2.getBalance(), balance2, 0);  Assert.assertTrue(outputStreamCaptor.toString().trim().contains("Nu exista contul " + sender));  }  @Test  public void testPaymentUtilities\_C3212() throws ProviderException, TransactionException {  String sender = "RO69RZBR00000682223758045";  String receiver= "RO59RZBR0000068222375809";  double value = 1;  BankAccount tmp = getBankAccountFromMap(debitAccount1, bankClient, bank.getClientBankAccountMap());  double balance1 = tmp.getBalance();  int index= ToProviders.providerList.indexOf(provider);  bank.paymentUtilies(sender,receiver,value);  double balance2 = ToProviders.providerList.get(index).getBalance();  Assert.assertEquals(debitAccount1.getBalance(), balance1, 0);  Assert.assertEquals(debitAccount2.getBalance(), balance2, 0);  Assert.assertTrue(outputStreamCaptor.toString().trim().contains("Nu exista contul " + sender));  }  @Test  public void testPaymentUtilities\_C3222() throws ProviderException, TransactionException {  String sender = "RO69RZBR00000682223758045";  String receiver= "RO59RZBR0000068222375809";  double value = -1;  BankAccount tmp = getBankAccountFromMap(debitAccount1, bankClient, bank.getClientBankAccountMap());  double balance1 = tmp.getBalance();  int index= ToProviders.providerList.indexOf(provider);  bank.paymentUtilies(sender,receiver,value);  double balance2 = ToProviders.providerList.get(index).getBalance();  Assert.assertEquals(debitAccount1.getBalance(), balance1, 0);  Assert.assertEquals(debitAccount2.getBalance(), balance2, 0);  Assert.assertTrue(outputStreamCaptor.toString().trim().contains("Nu exista contul " + sender));  }  @Test  public void testPaymentUtilities\_C3232() throws ProviderException, TransactionException {  String sender = "RO69RZBR00000682223758045";  String receiver= "RO59RZBR0000068222375809";  double value = 787678767;  BankAccount tmp = getBankAccountFromMap(debitAccount1, bankClient, bank.getClientBankAccountMap());  double balance1 = tmp.getBalance();  int index= ToProviders.providerList.indexOf(provider);  bank.paymentUtilies(sender,receiver,value);  double balance2 = ToProviders.providerList.get(index).getBalance();  Assert.assertEquals(debitAccount1.getBalance(), balance1, 0);  Assert.assertEquals(debitAccount2.getBalance(), balance2, 0);  Assert.assertTrue(outputStreamCaptor.toString().trim().contains("Nu exista contul " + sender));  } |

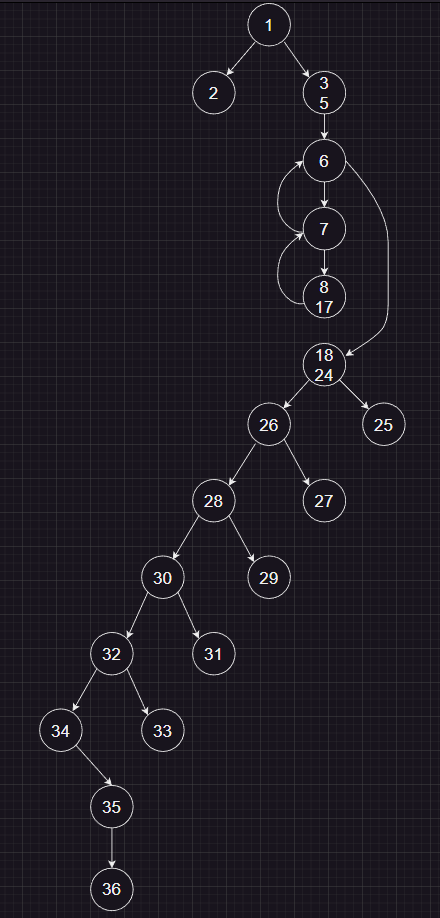
# Teste structurale

## Acoperirea la nivel de instructiune

Nodurile sunt scoase din mainClasses.Bank.interBanking.

Structura nume la teste: testInterBanking\_Nod1Nod2Nod3...\_NodIndividual\_NodIndividual\_...

Pt noduri cu separarea dintre noduri se va face cu un x. (Avem noduri cu 2 cifre si e mai lizibil)

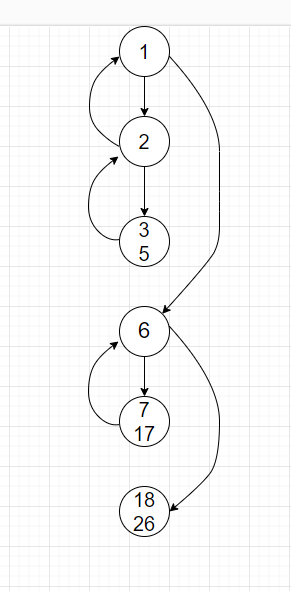


|  |
| --- |
| @Test  public void testInterBanking\_1\_2 () throws TransactionException {  bank.interBanking(debitAccount1.getIBAN(), savingsAccount1.getIBAN(), -10);  // Check if the accounts are still the same  BankAccount tmp1 = getBankAccountFromMap(debitAccount1, bankClient, bank.getClientBankAccountMap());  BankAccount tmp2 = getBankAccountFromMap(savingsAccount1, bankClient, bank.getClientBankAccountMap());  Assert.assertEquals(debitAccount1, tmp1);  Assert.assertEquals(savingsAccount1, tmp2);  // Check the messages. The logs are really detailed so we have to check if our string is in them  Assert.assertTrue(outputStreamCaptor.toString().trim().contains("De ce incerci asta? Fa-o invers :)"));  }  @Test  public void testInterBanking\_1\_3x5\_6\_7\_8x17\_18x24\_26\_28\_30\_32\_34\_36 () throws TransactionException {  double value = 100;  double initialSenderBalance = debitAccount1.getBalance();  double initialReceiverBalance = savingsAccount1.getBalance();  // receiver  bank.interBanking(getBankAccountFromMap(savingsAccount1, bankClient, bank.getClientBankAccountMap()).getIBAN(),  // sender  getBankAccountFromMap(debitAccount1, bankClient, bank.getClientBankAccountMap()).getIBAN(), value);  // We need to factor the conversion in  double toGiveToReceiver = CurrencyExchange.convertTransferWithoutText(value, savingsAccount1.getCurrency(), debitAccount1.getCurrency());  // Check the balances  Assert.assertEquals(initialSenderBalance - value, getBankAccountFromMap(debitAccount1, bankClient, bank.getClientBankAccountMap()).getBalance(), Math.ulp(getBankAccountFromMap(debitAccount1, bankClient, bank.getClientBankAccountMap()).getBalance()));  Assert.assertEquals(initialReceiverBalance + toGiveToReceiver, getBankAccountFromMap(savingsAccount1, bankClient, bank.getClientBankAccountMap()).getBalance(), Math.ulp(getBankAccountFromMap(savingsAccount1, bankClient, bank.getClientBankAccountMap()).getBalance()));  // Check the messages. The logs are really detailed so we have to check if our string is in them  Assert.assertTrue(outputStreamCaptor.toString().trim().contains("Transferul din contul " + debitAccount1.getIBAN() + " in contul " +  savingsAccount1.getIBAN() + " in valoare de " + FormatDouble.format(value) + " " +  debitAccount1.getCurrency() + " a avut succes!"));  }  @Test  public void testInterBanking\_1\_3x5\_6\_7\_8x17\_18x24\_25 () throws TransactionException {  double value = 100;  double initialSenderBalance = debitAccount1.getBalance();  double initialReceiverBalance = savingsAccount2.getBalance();  // receiver  bank.interBanking(getBankAccountFromMap(savingsAccount2, bankClient, bank.getClientBankAccountMap()).getIBAN(),  // sender  getBankAccountFromMap(debitAccount1, bankClient, bank.getClientBankAccountMap()).getIBAN(), value);  // Check the balances  Assert.assertEquals(initialSenderBalance, getBankAccountFromMap(debitAccount1, bankClient, bank.getClientBankAccountMap()).getBalance(), Math.ulp(getBankAccountFromMap(debitAccount1, bankClient, bank.getClientBankAccountMap()).getBalance()));  Assert.assertEquals(initialReceiverBalance, getBankAccountFromMap(savingsAccount2, bankClient, bank.getClientBankAccountMap()).getBalance(), Math.ulp(getBankAccountFromMap(savingsAccount2, bankClient, bank.getClientBankAccountMap()).getBalance()));  // Check the messages. The logs are really detailed so we have to check if our string is in them  Assert.assertTrue(outputStreamCaptor.toString().trim().contains("Nu se poate face transferul! Contul " + savingsAccount2.getIBAN() + " a fost inchis!"));  }  @Test  public void testInterBanking\_1\_3x5\_6\_7\_8x17\_18x24\_26\_27 () throws TransactionException {  double value = 100;  double initialSenderBalance = debitAccount2.getBalance();  double initialReceiverBalance = savingsAccount1.getBalance();  // receiver  bank.interBanking(getBankAccountFromMap(savingsAccount1, bankClient, bank.getClientBankAccountMap()).getIBAN(),  // sender  getBankAccountFromMap(debitAccount2, bankClient, bank.getClientBankAccountMap()).getIBAN(), value);  // Check the balances  Assert.assertEquals(initialSenderBalance, getBankAccountFromMap(debitAccount2, bankClient, bank.getClientBankAccountMap()).getBalance(), Math.ulp(getBankAccountFromMap(debitAccount2, bankClient, bank.getClientBankAccountMap()).getBalance()));  Assert.assertEquals(initialReceiverBalance, getBankAccountFromMap(savingsAccount1, bankClient, bank.getClientBankAccountMap()).getBalance(), Math.ulp(getBankAccountFromMap(savingsAccount1, bankClient, bank.getClientBankAccountMap()).getBalance()));  // Check the messages. The logs are really detailed so we have to check if our string is in them  Assert.assertTrue(outputStreamCaptor.toString().trim().contains("Nu se poate face transferul! Contul " + debitAccount2.getIBAN() + " a fost inchis!"));  }  @Test  public void testInterBanking\_1\_3x5\_6\_7\_8x17\_18x24\_26\_28\_29 () throws TransactionException {  double value = 100;  double initialSenderBalance = debitAccount2.getBalance();  double initialReceiverBalance = savingsAccount2.getBalance();  // receiver  bank.interBanking(getBankAccountFromMap(savingsAccount2, bankClient, bank.getClientBankAccountMap()).getIBAN(),  // sender  getBankAccountFromMap(debitAccount2, bankClient, bank.getClientBankAccountMap()).getIBAN(), value);  // Check the balances  Assert.assertEquals(initialSenderBalance, getBankAccountFromMap(debitAccount2, bankClient, bank.getClientBankAccountMap()).getBalance(), Math.ulp(getBankAccountFromMap(debitAccount2, bankClient, bank.getClientBankAccountMap()).getBalance()));  Assert.assertEquals(initialReceiverBalance, getBankAccountFromMap(savingsAccount2, bankClient, bank.getClientBankAccountMap()).getBalance(), Math.ulp(getBankAccountFromMap(savingsAccount2, bankClient, bank.getClientBankAccountMap()).getBalance()));  // Check the messages. The logs are really detailed so we have to check if our string is in them  Assert.assertTrue(outputStreamCaptor.toString().trim().contains("Nu se poate face transferul! Ambele conturi au fost inchise!"));  }  @Test  public void testInterBanking\_1\_3x5\_6\_7\_8x17\_18x24\_26\_30\_31 () throws TransactionException {  double value = 100;  double initialSenderBalance = debitAccount1.getBalance();  double initialReceiverBalance = nonExistingAccount.getBalance();  // receiver  bank.interBanking(nonExistingAccount.getIBAN(),  // sender  getBankAccountFromMap(debitAccount1, bankClient, bank.getClientBankAccountMap()).getIBAN(), value);  // Check the balances  Assert.assertEquals(initialSenderBalance, getBankAccountFromMap(debitAccount1, bankClient, bank.getClientBankAccountMap()).getBalance(), Math.ulp(getBankAccountFromMap(debitAccount1, bankClient, bank.getClientBankAccountMap()).getBalance()));  Assert.assertEquals(initialReceiverBalance, getBankAccountFromMap(nonExistingAccount, bankClient, bank.getClientBankAccountMap()).getBalance(), Math.ulp(getBankAccountFromMap(nonExistingAccount, bankClient, bank.getClientBankAccountMap()).getBalance()));  // Check the messages. The logs are really detailed so we have to check if our string is in them  Assert.assertTrue(outputStreamCaptor.toString().trim().contains("Nu exista contul in care transferati"));  }  @Test  public void testInterBanking\_1\_3x5\_6\_7\_8x17\_18x24\_26\_30\_32\_33 () throws TransactionException {  double value = 100;  double initialSenderBalance = nonExistingAccount.getBalance();  double initialReceiverBalance = debitAccount1.getBalance();  // receiver  bank.interBanking(debitAccount1.getIBAN(),  // sender  getBankAccountFromMap(nonExistingAccount, bankClient, bank.getClientBankAccountMap()).getIBAN(), value);  // Check the balances  Assert.assertEquals(initialSenderBalance, getBankAccountFromMap(nonExistingAccount, bankClient, bank.getClientBankAccountMap()).getBalance(), Math.ulp(getBankAccountFromMap(nonExistingAccount, bankClient, bank.getClientBankAccountMap()).getBalance()));  Assert.assertEquals(initialReceiverBalance, getBankAccountFromMap(debitAccount1, bankClient, bank.getClientBankAccountMap()).getBalance(), Math.ulp(getBankAccountFromMap(debitAccount1, bankClient, bank.getClientBankAccountMap()).getBalance()));  // Check the messages. The logs are really detailed so we have to check if our string is in them  Assert.assertTrue(outputStreamCaptor.toString().trim().contains("Nu exista contul din care transferati"));  }  @Test  public void testInterBanking\_1\_3x5\_6\_7\_8x17\_18x24\_26\_30\_32\_34\_35\_36 () throws TransactionException {  double value = 100;  double initialSenderBalance = nonExistingAccount.getBalance();  double initialReceiverBalance = nonExistingAccount.getBalance();  // receiver  bank.interBanking(nonExistingAccount.getIBAN(),  // sender  getBankAccountFromMap(nonExistingAccount, bankClient, bank.getClientBankAccountMap()).getIBAN(), value);  // Check the balances  Assert.assertEquals(initialSenderBalance, getBankAccountFromMap(nonExistingAccount, bankClient, bank.getClientBankAccountMap()).getBalance(), Math.ulp(getBankAccountFromMap(nonExistingAccount, bankClient, bank.getClientBankAccountMap()).getBalance()));  Assert.assertEquals(initialReceiverBalance, getBankAccountFromMap(nonExistingAccount, bankClient, bank.getClientBankAccountMap()).getBalance(), Math.ulp(getBankAccountFromMap(nonExistingAccount, bankClient, bank.getClientBankAccountMap()).getBalance()));  // Check the messages. The logs are really detailed so we have to check if our string is in them  Assert.assertTrue(outputStreamCaptor.toString().trim().contains("Nu exista nici un cont"));  } |

## Acoperirea la nivel de decizie/ramura

Nodurile sunt scoase din mainClasses.Bank.addLoan.

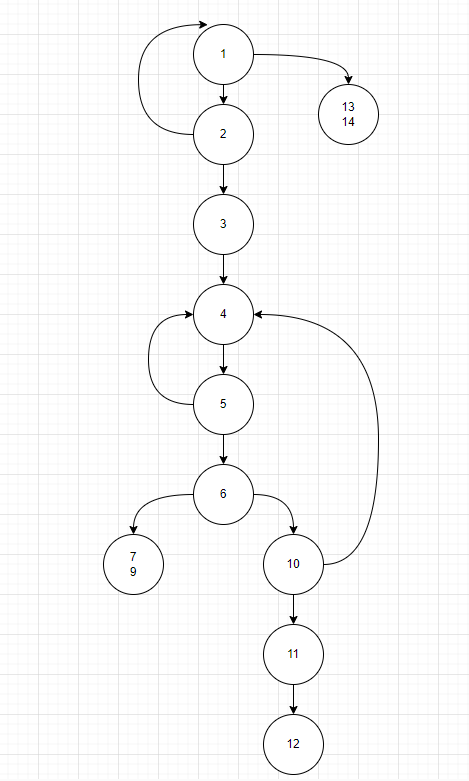
Structura nume la teste: testAddLoanWithExisisting/NonexistingLoan/Client



|  |
| --- |
| @Test public void testAddLoanWithNonExistingLoan () throws TransactionException {   bank.addLoan(bankClient, nonExistingBankLoan);   Assert.*assertTrue*(bank.getClientLoanMap().containsKey(bankClient));  Assert.*assertEquals*(bank.getClientLoanMap().get(bankClient).stream().filter(s -> s.getLoanID()==(nonExistingBankLoan).getLoanID()).collect(Collectors.*toList*()).get(0), nonExistingBankLoan); }  @Test public void testAddLoanWithAlreadyExistingLoan () throws TransactionException {   bank.addLoan(bankClient, bankLoan1);  Assert.*assertTrue*(outputStreamCaptor.toString().trim().contains("Imprumutul exista deja pentru client!")); }  @Test public void testAddLoanWithNonExistingClient () throws TransactionException {   bank.addLoan(bankClient2, bankLoan1);  Assert.*assertFalse*(bank.getClientLoanMap().containsKey(bankClient2)); } |

## Acoperirea la nivel de conditie

Nodurile sunt scoase din mainClasses.Bank.addCard.



|  |
| --- |
| @Test  public void testAddCard\_bankAccount\_nonexistent\_card\_ok () throws TransactionException, CardException {  Card card = new Card("5213512152346781", 905, "25-09-2020");  // receiver  bank.addCard(nonExistingAccount,card);  Assert.assertFalse(nonExistingAccount.getCardList().contains(card));  Assert.assertTrue(outputStreamCaptor.toString().trim().contains("Contul " + nonExistingAccount.getIBAN() + " nu exista."));  }  @Test  public void testAddCard\_bankAccount\_closed\_card\_ok () throws TransactionException, CardException {  Card card = new Card("5213512152346781", 905, "25-09-2020");  // receiver  bank.addCard(debitAccount2,card);  Assert.assertFalse(debitAccount2.getCardList().contains(card));  Assert.assertTrue(outputStreamCaptor.toString().trim().contains("Contul " + debitAccount2.getIBAN() + " a fost inchis deja, va rugam nu adaugati carduri."));  }  @Test  public void testAddCard\_bankAccount\_ok\_card\_existent () throws TransactionException, CardException {  Card card = new Card("5213512152346781", 905, "25-09-2020");  // receiver  bank.addCard(debitAccount1,card);  Assert.assertTrue(debitAccount1.getCardList().contains(card));  Assert.assertTrue(outputStreamCaptor.toString().trim().contains(""));  }  @Test  public void testAddCard\_bankAccount\_ok\_card\_ok () throws TransactionException, CardException {  Card card = new Card("5213512152346781", 905, "25-09-2020");  Card card1 = new Card("5603512157346791", 509, "05-10-2020");  // receiver  bank.addCard(debitAccount1,card1);  Assert.assertTrue(debitAccount1.getCardList().contains(card1));  Assert.assertTrue(outputStreamCaptor.toString().trim().contains(""));  } |

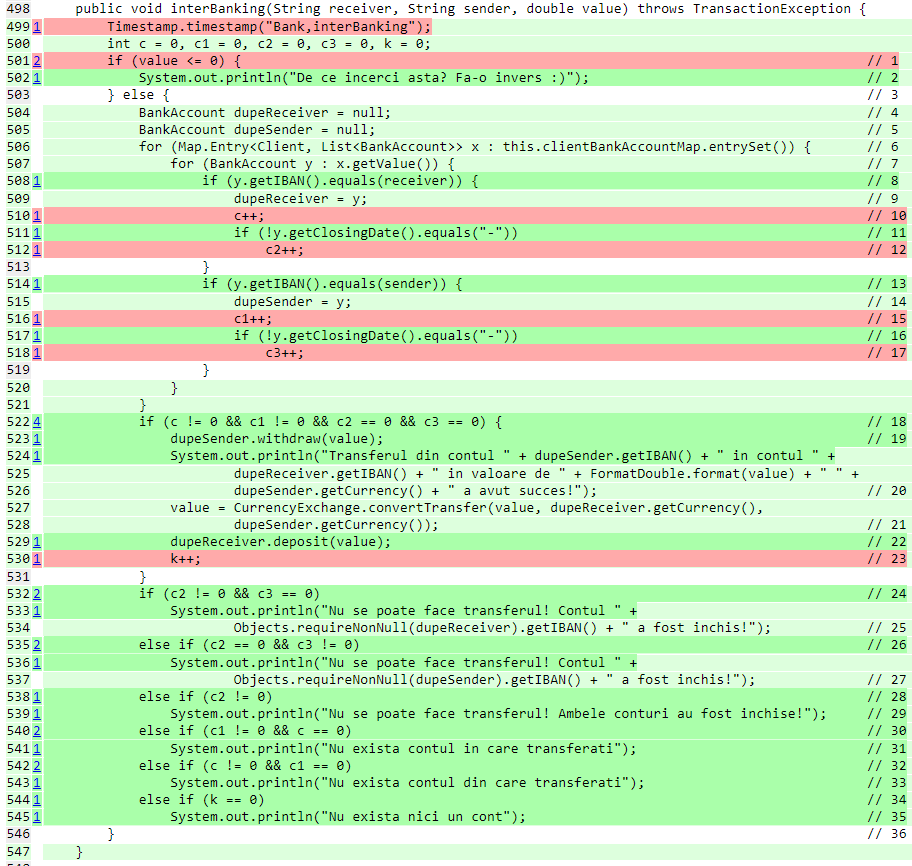
# Analiza set teste

Nevand alte variante pt a omori mutantii pe testele de mai sus (am putut doar unul), sunt nevoit sa fac niste teste pe o alta metoda pentru a putea indeplini cerinta. Metoda se afla in mainClasses.Loan

|  |
| --- |
| @Test  public void testPayMonthlyRate () throws LoanException {  // Testul nu are logica dpdv al functionalitatii bancii, el fiind facut sa punctam cerinta  // Monthly rate ul aici va fi 200lei/2luni = 100 lei per luna  Loan loan = new Loan(200, "Lei", "Imprumut pentru nevoi personale", "03-12-2021", 2);  loan.payMonthlyRate(100, bankClient.getCnp());  // Din imprumut mai raman 100lei  Assert.assertEquals(100, loan.getValue(), Math.ulp(100));  } |

## Raport PiTest

1. Inainte de eliminarea mutantilor



Testul creat