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)

[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()));  } |

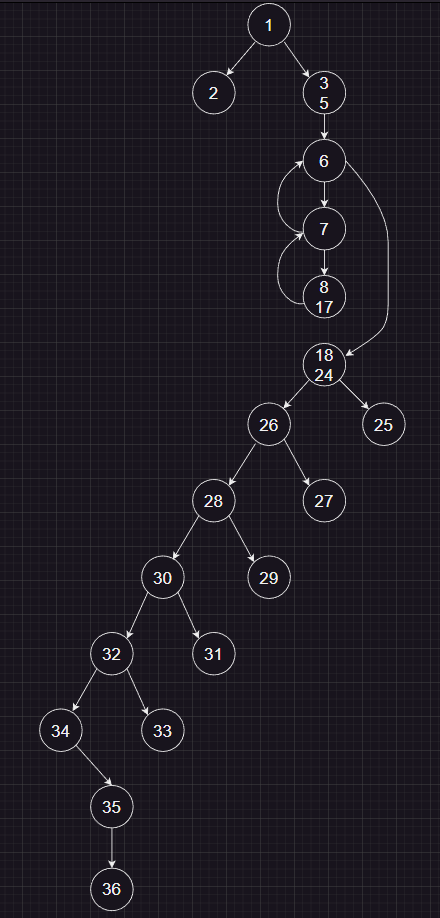
# Teste structurale

## Acoperirea la nivel de instructiune

Nodurile sunt scoate 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 de 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"));  } |

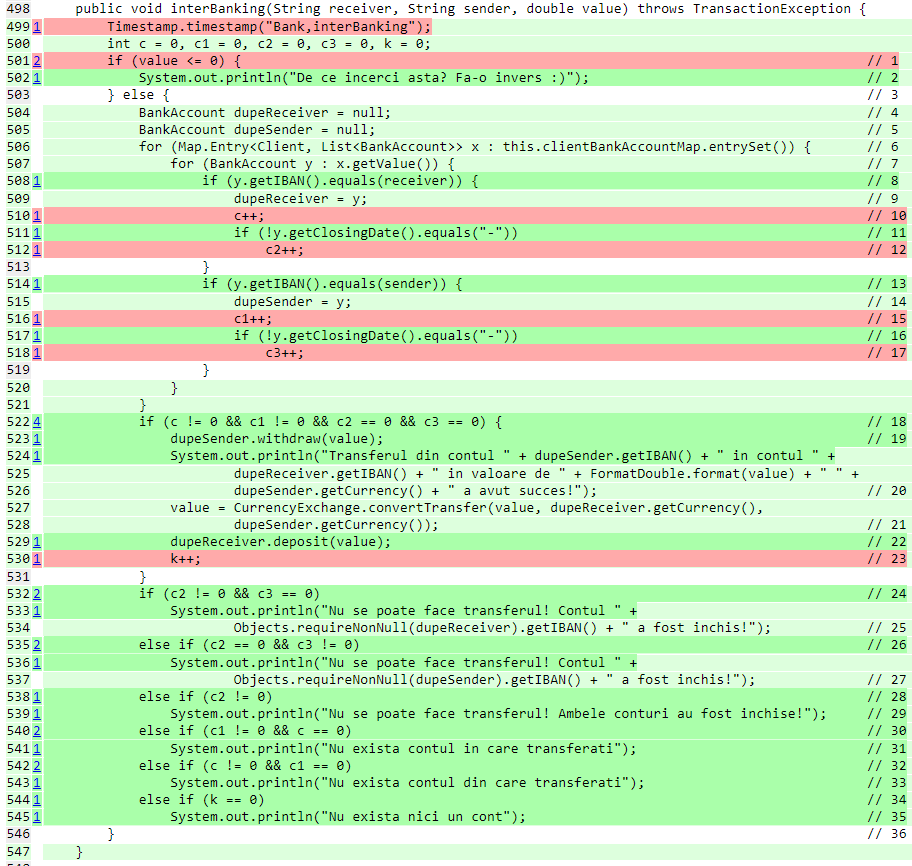
# Analiza set teste

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