

Java JUnit labor

Készítette: Budai Péter, BME IIT, 2024.

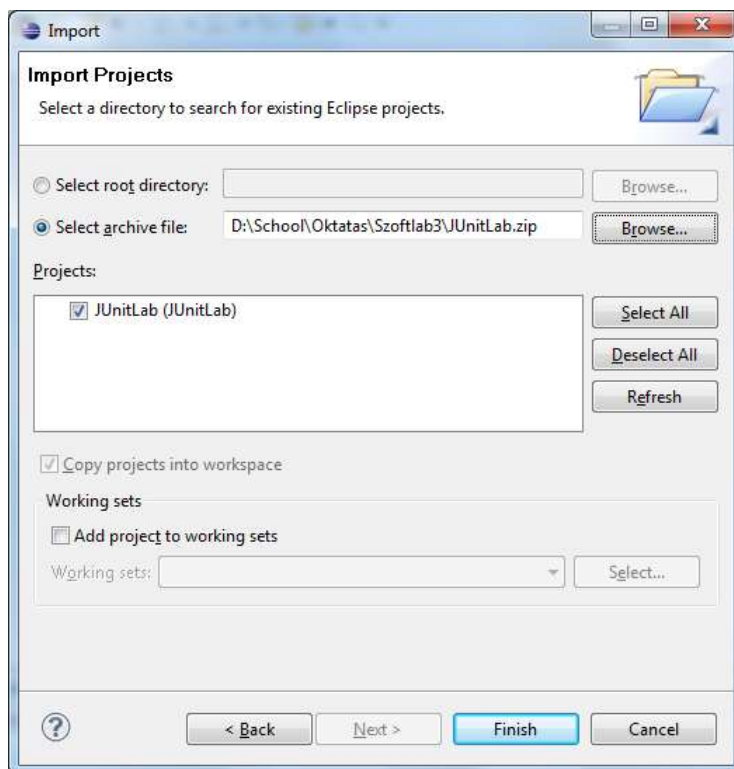
A feladatok megoldása előtt mindenképp ajánlatos végigolvasni és lépésről lépésre végigcsinálni a laborfeladathoz tartozó, és a tárgy honlapján megtalálható *JUnit tutorialt*.

A JUnit alapú egységteszteléshez felhasználandó annotációk leírásait az alábbi URL-en találja meg:
<https://junit.org/junit5/docs/current/api/index.html>

Többek között a parametrikus tesztelés használatára is mutat példát az alábbi *JUnit User guide*:
<https://junit.org/junit5/docs/current/user-guide/>

1 Projekt megnyitása

A tárgyhonlap eheti laborfeladatához tartozó oldaláról töltsse le a feladat megoldásához használandó Eclipse projekt vázat. Ez egy ZIP archívum, amelyet közvetlenül be lehet importálni az Eclipse-be. Ehhez válassza a **File/Import...** menüpontot, majd ott keresse ki a **General / Existing projects into workspace** lehetőséget! A felbukkanó ablakban tallózza ki a letöltött archívumot, majd kattintson a **Finish** gombra (lásd az alábbi ábrát).

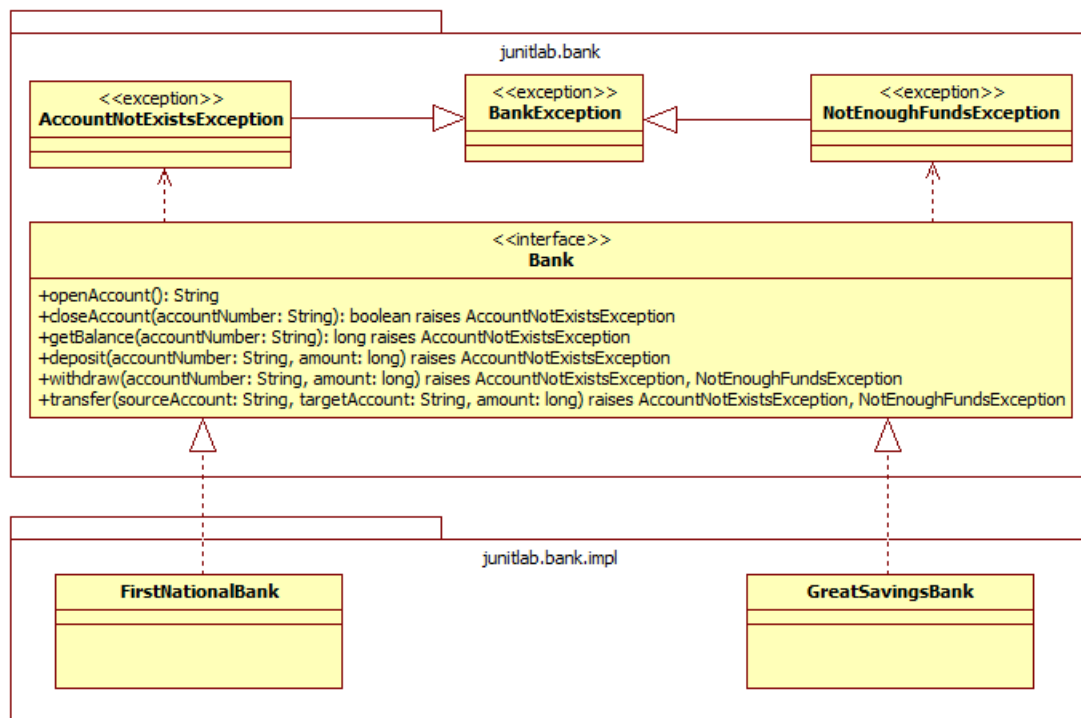


A beimportált projektben négy Java fájl és egy összecsomagolt **.jar** osztálykönyvtár található. Az első forrásfájl (**Bank.java**) egy elképzelt banki szolgáltatás interfészét (**Bank**) tartalmazza. A bankban

számlát nyithatunk, vagy megszüntethetjük azt, lekérdezhetjük az egyenlegünket, pénzt fizethetünk be, illetve ki, valamint átutalhatunk egy összeget egyik számláról egy másikra. *Az átutalásnál tetszőleges összeget megadhatunk, de pénzfelvétel vagy befizetés esetén a bank kerek 100-as értékre kerekít.*

A különböző műveletek kivételeket (**AccountNotExistsException**, **NotEnoughFundsException**) dobhatnak, ha nem létező számlaszámot adunk meg vagy például nincs elég pénz a számlánkon a tranzakció elvégzéséhez. Ezeket a kivétel osztályokat (illetve a közös őssztályukat, a **BankException** tartalmazza a többi Java forrásfájl.

A library-ként hivatkozott **.jar** állományban a fent leírt **Bank** interfészt megvalósító két különböző implementációs osztály (**FirstNationalBank** és **GreatSavingsBank**) található, ezeket kell majd a feladat során tesztelni.



Az eddig leírtakat a fenti osztálydiagram foglalja össze. Tanulmányozza át a **Bank** interfész metódusain szereplő **Javadoc** kommenteket, melyek pontról pontra leírják, hogy az egyes műveleteknek hogyan is kellene működniük! Ez azért fontos, mert ez alapján kell majd végezni a tesztelést!

2 Egyszerű tesztesetek készítése

(Lásd a JUnit tutorial dokumentum 2. fejezetét)

Készítse elő a projektet arra, hogy JUnit tesztek lehessen rajta végezni. Ehhez először is adja hozzá a **JUnit5 Library**-t a projekthez, majd hozzon létre új forrásmappát (**test**) a teszt osztályok számára, ahogy az a leírásban is szerepel.

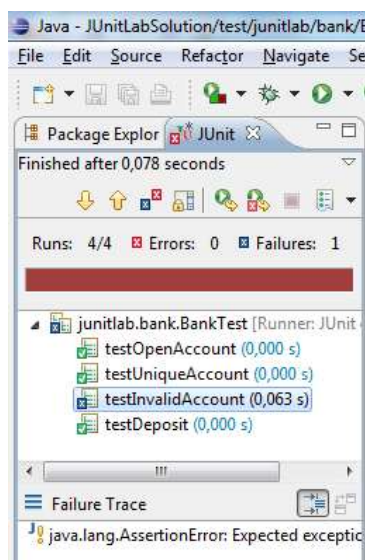
*Ebben és a következő feladatokban a **FirstNationalBank** implementációs osztályt teszteljük, a **GreatSavingsBank** osztállyal egyelőre ne foglalkozzon!*

Ezután hozzon létre egy teszt osztályt **junitlab.bank.BankTest** néven és készítsen bele teszt metódusokat a számlanyitás (**openAccount**) és egyenleglekérdezés (**getBalance**) és befizetés (**deposit**) műveletek tesztelésére! Az alábbiakat kell ellenőriznie egy-egy teszt metódussal:

- **testOpenAccount**: A megnyitott számla létezik, és az új számla egyenlege valóban nulla (0).
- **testUniqueAccount**: Az egymás után megnyitott számlák számlaszámai valóban különböznek.
- **testInvalidAccount**: Nem létező számlaszám esetén valóban az elvárt kivételt (**AccountNotExistsException**) dobja-e az egyenleg lekérdező metódus.
- **testDeposit**: Fizessen be 2000 forintot egy új számlára, majd ellenőrizze, hogy a számlán valóban a várt összeg szerepel-e.

A teszt metódusokban ne szerepeljenek a kimenetre író utasítások, és használja az **org.junit.jupiter.api.Assertions** osztály megfelelő statikus metódusait! Ne kezeljen semmilyen kivételt a teszt metódusokban, hagyja ezt a keretrendszerre!

Ha mindent jól csinált, akkor a négy tesztesetből három sikeresen lefut. Mi volt a hiba a negyedik esetén?



3 Tesztinicializálás (fixture) készítése

(Lásd a JUnit tutorial dokumentum 3. fejezetét)

A további tesztekhez már előkészített számlákra lesz szükség. Hozzon létre egy új teszt osztályt (**junitlab.bank.BankTestFixture**), mely minden teszt metódus lefuttatása előtt az alábbi tesztkörnyezetet (text fixture) készíti elő!

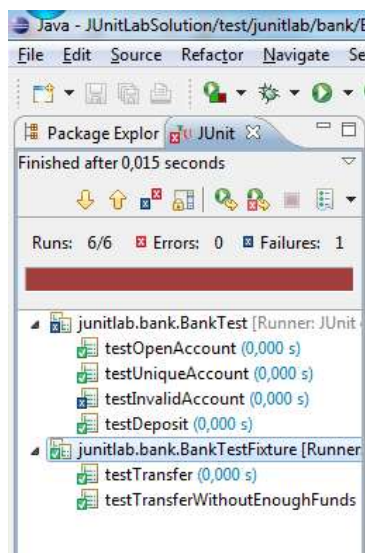
- Létrehoz, és tagváltozóként eltárol egy **Bank** objektumpéldányt (jelen esetben egy **FirstNationalBank** példányt)
- A bank objektum segítségével megnyit két számlát:
 - az egyik számlán 1500 forintot,
 - a másikon pedig 12000 forintot helyez el.

A számlaszámokat tárolja el tagváltozóként, hogy a teszt metódusok hozzáférhessenek a számlákhoz!

Ezek után készítse el ebben az új teszt osztályban a számlák közti átutalás (**transfer**) műveletének tesztjét! A következőket kell ellenőriznie egy-egy teszt metódusban:

- **testTransfer**: Utaljon át 3456 forintot a második számláról az elsőre, és ellenőrizze, hogy a számlákon valóban az elvárt összegek (4956 és 8544 forint) szerepelnek-e az átutalás után.
- **testTransferWithoutEnoughFunds**: Utaljon át 3456 forintot az első számláról a másodikra, és ellenőrizze, hogy valóban az elvárt kivétel (**NotEnoughFundsException**) keletkezik-e!

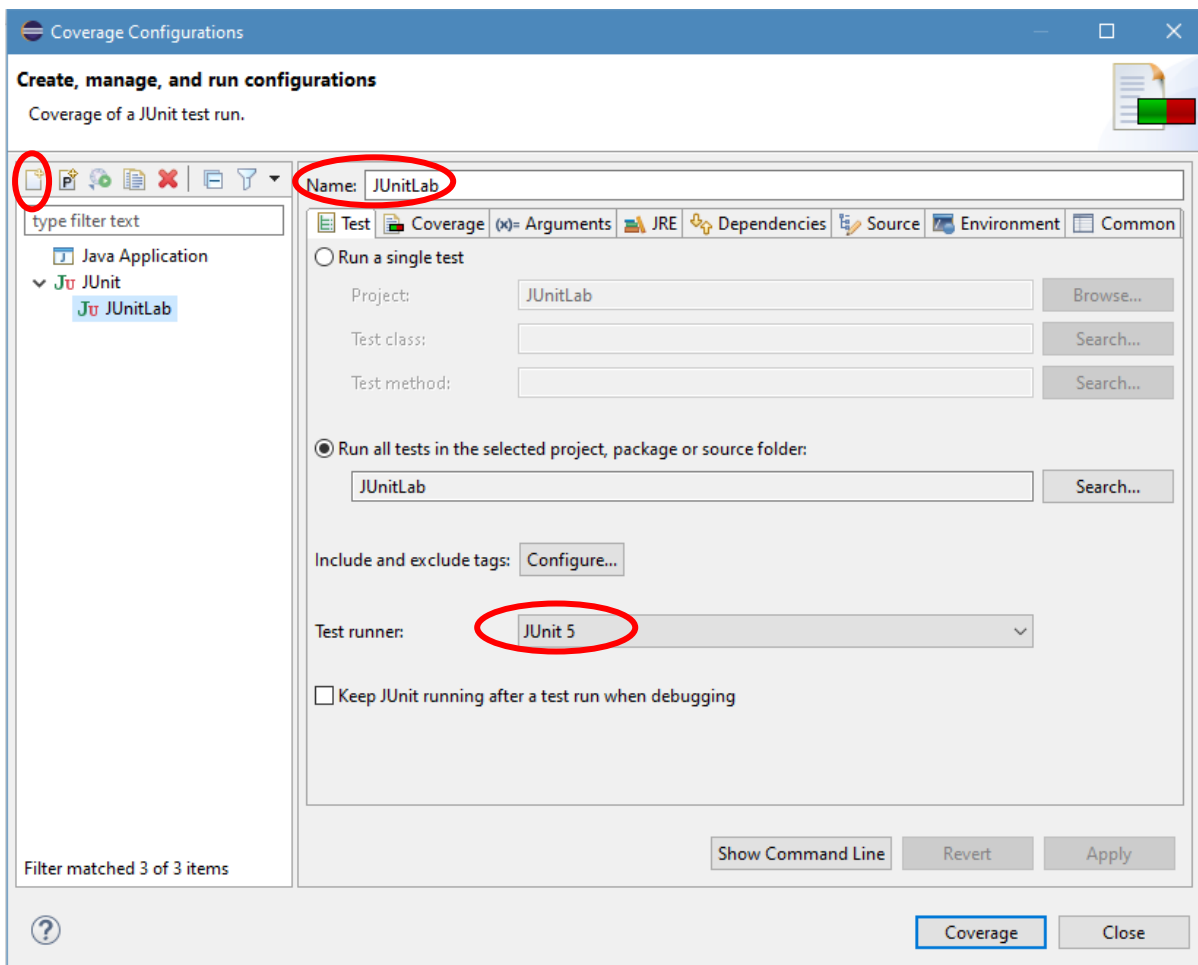
A teszt metódusokban használja az **org.junit.jupiter.api.Assertions** osztály megfelelő statikus metódusait, valamint a metódusok törzsében ne kezeljen le semmilyen kivételt, hagyja ezt a keretrendszerre! Ha mindent jól csinált, a tesztek sikeresen lefutnak:



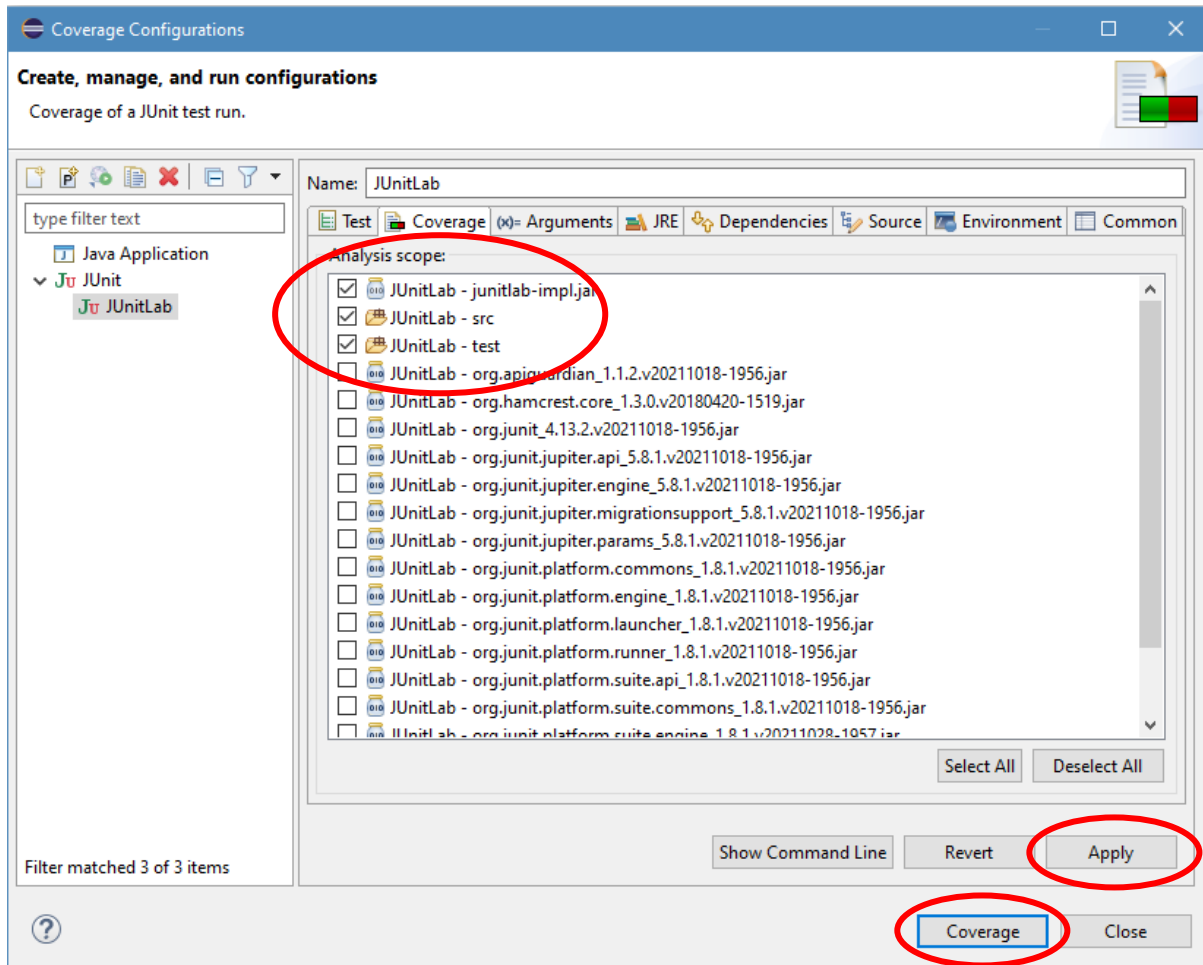
4 Kódlefedettség-vizsgálat

(Lásd a JUnit tutorial dokumentum 4. fejezetét)

Ahhoz, hogy a kódlefedettség vizsgálat a hivatkozott **junitlab-impl.jar** osztálykönyvtárat is érintse, először hozzon létre egy új *Coverage Configuration*-t. Ehhez kattintson jobb egérgombbal a projekt nevére, majd válassza a **Coverage As/Coverage Configurations...** lehetőséget. A megjelenő ablakban az alábbi ábrán megjelölt gomb segítségével hozzon létre egy új **Launch Configuration**-t a **JUnit** csoporton belül. A **Test** fülön nevezze el a konfigurációt, a **Test runner** opciónál válassza a **JUnit 5** lehetőséget.



A **Coverage** fülön az **Analysis Scope** listában pedig pipálja be, hogy az **src** és **test** forrásmappán kívül a **lib/junitlab-impl.jar** osztálykönyvtár elemeit is figyelembe vegye (lásd következő oldali ábra). Végül az **Apply** gombbal mentse el a változtatásokat és a **Coverage** gombbal indítsa el a mérést!



Milyen eredményeket kapott? Próbálja meg duplakattintással megnyitni a **Coverage** nézetből a **FirstNationalBank.class** állományt! Azt kell tapasztalnia, hogy a forráskód helyett csak egy **Source not found** üzenet, és egy halom JVM bytecode fogadja.

BankTest.java BankTestFixture.java FirstNationalBank.class

Class File Editor

Source not found

The JAR file junitlab-impl.jar has no source attachment.
You can attach the source by clicking Attach Source below:

[Attach Source...](#)

Problems Javadoc Declaration Console Debug Expressions Coverage Search

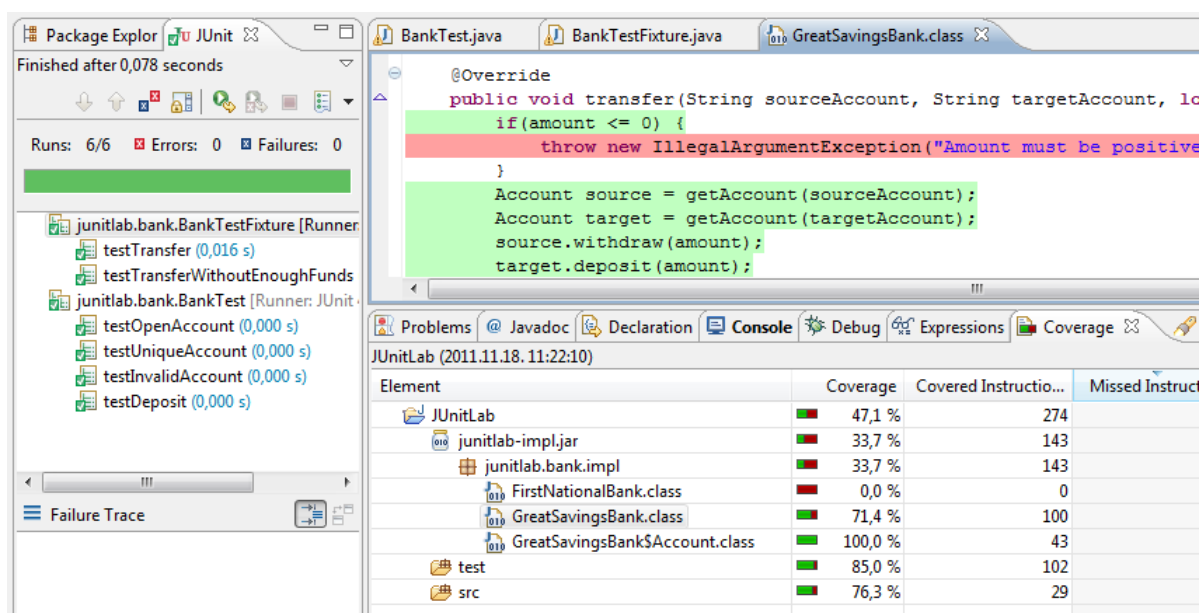
JUnitLab (2011.11.18. 11:10:03)

Element	Coverage	Covered Instruction...	Missed Instructions	Total Instructions
JUnitLab	45,7 %	266	316	582
junitlab-impl.jar	32,3 %	137	287	424
junitlab.bank.impl	32,3 %	137	287	424
GreatSavingsBank.class	0,0 %	0	140	140
FirstNationalBank.class	56,8 %	137	104	241
GreatSavingsBank\$Account.class	0,0 %	0	43	43
src	47,4 %	18	20	38
test	92,5 %	111	9	120

5 A tesztelt implementációs osztály cseréje

Módosítsa a 2. és 3. feladatban elkészített teszt osztályokat (**BankTest** és **BankTestFixture**) úgy, hogy a **FirstNationalBank** osztály helyett ezentúl a **GreatSavingsBank** osztály példányain végezzék el ugyanazokat a teszteket!

Futtassa le újra a 4. feladatban elkészített **Coverage Configuration**-t! Ha jól csinált mindent, akkor a most tesztelt implementációs osztály az összes teszten sikeresen átmegy. Sőt, mivel a **GreatSavingsBank** osztályhoz a forráskód is csatolva van, a **Coverage** ablakból duplakattintással megnyitva a **GreatSavingsBank.class** állományt, meg is tudja tekinteni azokat a forrássorokat, melyeket érintette a tesztelés:



The screenshot shows an IDE with the following components:

- Package Explorer:** Shows the test suite structure:
 - junitlab.bank.BankTestFixture [Runner: JUnitLab]
 - testTransfer (0,016 s)
 - testTransferWithoutEnoughFunds
 - junitlab.bank.BankTest [Runner: JUnitLab]
 - testOpenAccount (0,000 s)
 - testUniqueAccount (0,000 s)
 - testInvalidAccount (0,000 s)
 - testDeposit (0,000 s)
- Code Editor:** Shows the **BankTest.java** file with the following code:


```
@Override
public void transfer(String sourceAccount, String targetAccount, long amount) {
    if (amount <= 0) {
        throw new IllegalArgumentException("Amount must be positive");
    }
    Account source = getAccount(sourceAccount);
    Account target = getAccount(targetAccount);
    source.withdraw(amount);
    target.deposit(amount);
}
```
- Coverage View:** Shows the coverage results for the test suite:

Element	Coverage	Covered Instruction...	Missed Instruct...
JUnitLab	47,1 %	274	
junitlab-impl.jar	33,7 %	143	
junitlab.bank.impl	33,7 %	143	
FirstNationalBank.class	0,0 %	0	
GreatSavingsBank.class	71,4 %	100	
GreatSavingsBank\$Account.class	100,0 %	43	
test	85,0 %	102	
src	76,3 %	29	

6 Paraméteres tesztelés

(Lásd a JUnit tutorial dokumentum 5. fejezetét)

Készítsen egy új teszt osztályt (**junitlab.bank.BankParamTest**), mely képes a bank kerekítési algoritmusának paraméteres tesztelésére! Ehhez egy tesztnek két paramétert kell kapnia: a kerekítés nélküli (**amount**) és a kerekített (**rounded**) összeg. Készítse el a tesztmetódusokat úgy, hogy ezt a két argumentumot várják, és tárolja el őket egy-egy változóban!

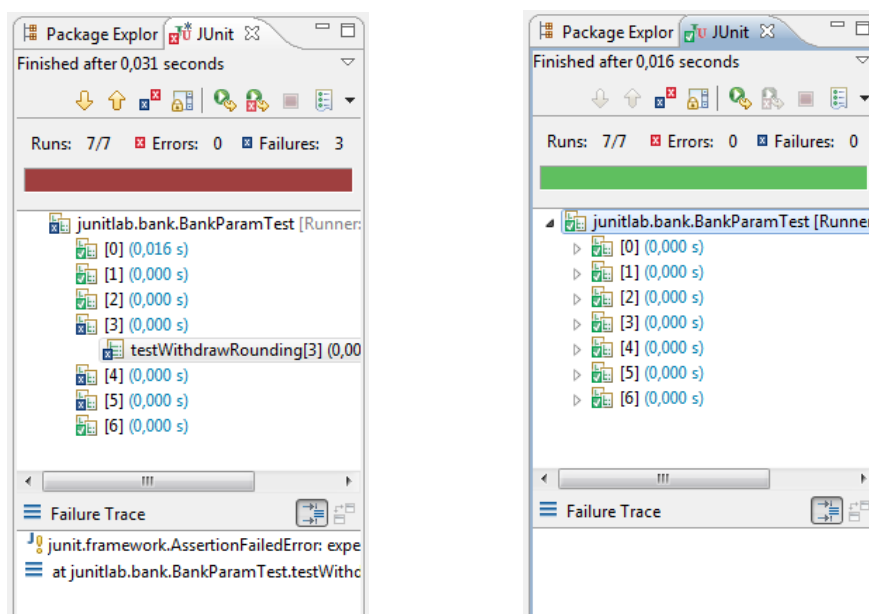
Írja meg az adatsorokat előállító statikus függvényt, mely az alábbi kerekítetlen értékeket (és természetesen a hozzájuk tartozó kerekített értékeket) tartalmazó paraméterpárok listáját állítja elő:

1100, 1101, 1149, 1150, 1151, 1199, 1200

Készítsen egy teszt metódust (**testWithdrawRounding**) ebben az új teszt osztályban, mely a bankok által a pénzfelvétel (**withdraw**) műveletkor végrehajtott kerekítés helyességét ellenőrzi! A metódus hozzon létre egy új bank példányt, benne egy új számlát, amin helyezzen el 10000 forintot. Ezek után próbáljon felvenni a teszt paraméterében kapott kerekítetlen (**amount**) összegnek megfelelő forintot a számláról, majd ellenőrizze, hogy valóban a szintén paraméterül kapott helyes kerekített értékkel (**rounded**) csökkent-e a számla egyenlege.

A teszt metódusban használja az **org.junit.jupiter.api.Assertions** osztály megfelelő statikus metódusait, a metódusok törzsében pedig ne kezeljen le semmilyen kivételt, hagyja ezt a keretrendszerre!

Kétszer is futtassa le a tesztet, először a **FirstNationalBank** osztály egy példányát, másodszor pedig a **GreatSavingsBank** osztály egy példányát tesztelve! Az alábbi ábrához hasonló eredményeket kell kapnia:



7 További hiányzó tesztesetek elkészítése

A **GreatSavingsBank** forráskódjának ismeretében készítsen további teszteseteket úgy, hogy a **GreatSavingsBank** osztály minden kódsora le legyen tesztelve (teljes, vagy legalábbis közel 100%-os forráskód lefedettséget érjen el). Ebben a feladatban szabad kezet kap, létrehozhat új teszt osztályt vagy bővíthet egy meglévőt újabb metódusokkal.