4. gyakorlat anyaga

4. Egységtesztelés: JUnit

Hasznos információk

Letöltések

A fájlok közöttt minden szükséges állomány megtalálható. A mai órára szükséges fájlok:

- JUnit
- CheckThat
- Tesztesetek fájljai

Fordítás és futtatás

Windows

Fordítás Windows alatt:

```
javac -cp '.;junit5all.jar;checkthat.jar' fordítandó/Fájl.java
```

Egy osztályon belüli tesztesetek futtatása:

```
java -jar junit5all.jar -cp '.;checkthat.jar' -c teljes.minősített.OsztályNév
```

Az összes teszteset futtatása a `classpath`-ban:

```
java -jar junit5all.jar -cp '.;checkthat.jar' --scan-classpath
```

Linux, MacOS

Fordítás Unix rendszereken:

```
javac -cp .:junit5all.jar:checkthat.jar fordítandó/Fájl.java
```

Egy osztályon belüli tesztesetek futtatása:

```
java -jar junit5all.jar -cp .:checkthat.jar -c teljes.minősített.OsztályNév
```

Az összes teszteset futtatása a `classpath`-ban:

```
java -jar junit5all.jar -cp .:checkthat.jar --scan-classpath
```

Demo

1. demó

Írjunk `fib()` metódust a `famous.sequence.Fibonacci` osztályba, amely egy `n` számot kap meg, és kiadja az `n`-edik Fibonacci-számot. Feltesszük, hogy az `n` bemenet kis, pozitív szám.

Teszteljük a következőképpen.

- A metódus kapjon `static` minősítőt.
- A mellékelt `famous.sequence.FibonacciStructureTest` használata.
  - Ezt a csomagjának megfelelő könyvtárba kell elhelyezni.
  - Ez a `CheckThat` eszközt használja, aminek szintén elérhetőnek kell lennie.
  - Fordítás és futtatás. A parancsokat lásd feljebb.
    - Először a `FibonacciStructureTest` osztályt fordítsuk.
    - Most a `Fibonacci.java` osztályt külön paranccsal kell lefordítani, mert a tesztelő nem hivatkozik rá közvetlenül, és így az előző fordításban a `javac` nem keresi meg.
    - Futtassuk a teszteket.
- Készítsünk `famous.sequence.FibonacciTest` tesztelőt.
  - Ebben legyen teszt néhány konkrét `n` értékre.
  - Próbáljuk ki a paraméterezett tesztelőt is.
  - Fordítás és futtatás.
    - A folyamat jobb megértése érdekében töröljük le a korábban elkészült összes `.class` fájlt. (A `.java` fájlok maradjanak meg!)
    - Fordítsuk le a `FibonacciTest` kódját. Ez a `Fibonacci.java` fájlt is megtalálja és lefordítja.
    - Futtassuk a teszteket.
- A mellékelt `famous.sequence.FibonacciTestSuite` használata.
  - Ezt is a csomagjának megfelelő könyvtárba kell elhelyezni.
  - Ismét töröljük le a korábban elkészült összes `.class` fájlt.
  - Fordítsuk le a `FibonacciTestSuite` kódját. Ezúttal `Fibonacci.java` és a két korábbi tesztelő is lefordul.
  - Futtassuk a teszteket.

Feladatok

1. feladat

Készítsünk háromszögszámokat kiszámító `static famous.sequence.TriangularNumbers.getTriangularNumber()` metódust.

- A tesztelő kód a `famous.sequence.TriangularNumbersTest` osztályba kerüljön.
- Ehhez is jár mellékelt strukturális tesztelő és suite is, ezeknek is rendben le kell futniuk.
  - A fájlokat a megfelelő könyvtárba kell elhelyezni, különben nem működnek.

Kipróbálandók a következő bemenetek.

- nulla
- egy
- a számra, a fiatal Gauss anekdotájában szerepel
- mínusz egy
- más negatív szám

A tesztelőbe írjunk szándékosan egy rossz tesztesetet is, direkt elrontott elvárt eredménnyel. Vizsgáljuk meg a futtatás eredményét, benne az összefoglalóval és a *stack trace* részleteivel.

Az osztályba kerüljön `getTriangularNumberAlternative()` is. Ez a képlettel dolgozzon, és adja ugyanazt a kimenetet, mint `getTriangularNumber()`.

- A tesztelő próbálja ezt is ki.

2. feladat

Készítsük el a `static math.operation.safe.Increment.increment()` metódust.

- Legyen `static` minősítőjű.
- Bemenete egy `int`.

- Ha a bemenete a legnagyobb ábrázolható egész, azzal tér vissza, különben a bemeneténél eggyel nagyobb egésszel.
  - Erre így lehet hivatkozni: `Integer.A_MEGFELELŐ_ADATTAG_NEVE`.
  - A keresett név a Java API dokumentációból olvasható ki.

Ugyanebbe a csomagba készítsük el az `IncrementTest` osztályt az `increment()` működésének kipróbálására.

- Bemenete legyen nulla.
- Bemenete legyen a legkisebb ábrázolható egész.
- Bemenete legyen a legnagyobb ábrázolható egész.
- Bemenete legyen egy közepesen nagy, pozitív egész.
- Bemenete legyen egy közepesen nagy, negatív egész.
- Bemenete legyen `-1`.

Készítsünk egy hasonló tesztelő fájlt egy másik csomagba is, és próbáljuk ki ott is.

3. feladat

Készítsük el a mellékelt strukturális tesztelőben (`RecordLabelStructureTest`, `ArtistStructureTest`, `FanStructureTest`) leírt osztályokat a megadott szerkezettel.

Ebben a feladatban előre meg van adva néhány funkcionális teszt is. Ezeket a fájlokat bővítsük az alábbi tesztekkel.

- Tesztelendő, hogy létrehozás után mindegyik objektumra helyes értéket adnak a getterek.
  - A rajongónak, a rajongó művészének, és a rajongó művésze kiadójának a neve megfelelő kell, hogy legyen.
  - A rajongó létrehozás után közvetlenül még nem költött el pénzt.

A funkcionális tesztelőben az alábbi tesztek érhetők el a `Fan` osztályhoz. Itt a feladat a megvalósítás elkészítése.

- `buyMerchandise()` kiszámítja a rajongó által vásárolt termék árát.
  - A rajongó természetesen csak a kedvenc művészétől vesz termékeket.
  - Az alapár paraméterként van megadva, de ha a rajongó másokkal együtt veszi meg a terméket, az a darabárat fejenként 10%-kal, de legfeljebb 20%-kal csökkenti.
    - A további rajongókat deklaráljuk egy *vararg* paraméterben, így: `Fan... friends`.
    - Így hívható meg a metódus: `fan1.buyMerchandise(100, friend1, friend2, friend3)`, ahol mindegyik `friendN` egy `Fan` példány.
    - A költségcsökkentés mértéke akkor sem haladja meg a 20%-ot, ha kettőnél több extra rajongó csatlakozik.
  - A metódus visszatérési értéke a számított darabár.
    - Ezt az összeget elkölti a rajongó és mindegyik barátja.
    - Az összes költés felét megkapja a kiadó a `gotIncome()` segítségével.
- `favesAtSameLabel()` azt vizsgálja meg, hogy ugyanaz a művész-e két rajongó kedvence.
- Az alábbi négy metódus állítsa elő a rajongó szöveges reprezentációját. Ezek visszatérési értéke megegyezik, de más-más módon állítják elő azt.
  - A `toString1()` metódus a `+` operátorral fűz össze szövegrészleteket.
    - Ez hatékony, de macerás megírni és olvasni.
  - A `toString2()` metódus a `formatted` metódust használja ilyen módon: `"%s=%d".formatted("one hundred", 100)`.
    - Ezt könnyű olvasni.
  - A `toString3()` metódus ezt használja: `String.format("%s=%d", "one hundred", 100)`.
    - Ez az előzőhöz hasonló, kicsit régebbi stílusú.
  - A `toString4()` metódus egy `StringBuilder` példányt használ.
    - A `sb.append(123)` hívással lehet bővíteni a tartalmat.
    - A `sb.toString()` hívással lehet szövegként kinyerni a tartalmat.
    - Ez akkor hasznos, ha a kimenet hosszú és összetett.

4. feladat

A `math.operation.textual.Adder.addAsText()` metódus két `String` paramétert vár.

- Ha mindkettő értelmezhető egész számként, az eredmény az összegük szövegesen.
- Hasonlóan működik, ha a bemenetek értelmezhetőek lebegőpontos számként.
- Ha a bemenetek nem szám jellegűek, az eredmény a következő szöveg: `Operands are not numbers`.
- Az implementáció forráskódja **nem** elérhető, csak a `.class` fájl.
  - Ezt a megfelelő könyvtárba kell tenni.

Ugyanebbe a csomagba készítsük el az `AdderTest` osztályt, ami a következőket próbálja ki.

- `wrongInput`: legalább az egyik paraméter nem szöveges
- `addZero`: egy számhoz nullát adunk hozzá
  - Mindkét oldal lehessen a nulla.
  - Kipróbálandó: a nulla vagy a szám lebegőpontos.
- `add`: összeadunk két számot
  - Az összeadás mindkét sorrendben ugyanazt az eredményt adja.
  - Kipróbálandó: mindkét szám egész; mindkettő lebegőpontos; egyik ilyen, másik olyan.
- `addCommutative`: két szám összeadva a megadott sorrendben és ahhoz képest fordítva ugyanazt az eredményt adja.

Megengedett egyszerűsítés.

- Egy metódusba szabad egynél több `assertX` hívást is írni.
- Ez realisztikus kódban nem jó választás.
- Létezik erre jobb megoldás (`assertAll`), de az túlmutat a kurzus keretein.

A fentieket a következő módokon lehet bővíteni.

- Néhány (vagy akár mindegyik) feltétel több paraméterrel is kipróbálható (`@ParameterizedTest`).
- A `wrongInput` teszt kerüljön át az `AdderInvalidTest.java` fájlba. Egy test suite hívja meg mindkét osztályt.

Gyakorló feladatok

1. gyakorló feladat

Implementálja a Caesar-kódolást a `cipher.CaesarCipher` osztályban a következőek szerint.

- Az eltolási távolságot konstruktor paraméterként kapja meg (`shift`) és tárolja el.
- Az `encrypt()` metódus egy szöveget kap meg, és az alábbiak szerint, karakterenként kódolja el.
  - Ha a karakter nem betű (`'a'` és `'z'` közötti karakter), akkor megtartjuk. Különben a `shift` pozícióval odébb levő karaktert adjuk a kimenethez.
    - Például: `'d'` hárommal eltolva `'g'` lesz.
    - Figyelem: az eltolás átfordulhat: `'z'` kettő távolsággal eltolva `'b'` lesz.
    - Figyelem: az eltolás lehet negatív, és akkor is átfordulhat: `'c'` `-5` távolsággal eltolva `'x'` lesz.
  - További részletek.
    - Az eredmény szöveget `+` operátorral összefűzve is elő lehet állítani, de még jobb `StringBuilder` használatával.
    - Az eltolás után típuskényszerítést kell használni: `(char)shiftedChar`.
- A `decrypt()` metódus legyen az `encrypt()` ellentéte.

A `cipher.CaesarCipherTest` osztály a következőket tesztelje.

- `noShift`: ha `shift` értéke `0`, `encrypt()` és `decrypt()` egyaránt az eredeti szöveget adja
- `encryptBy`: adott szöveg elkódolása adott `shift` távolsággal
  - üres szöveg (a `textBlock` részre `''` alakban írható le)

    - egybetűs szöveg
    - hosszabb szöveg
    - nem csak kisbetűket tartalmazó szöveg
- `inverseOperation`: két, egymással ellentétes hatású művelet meghívása egymás után, az eredmény az eredeti szöveg
    - `encrypt()`, aztán `decrypt()` ugyanolyan `shift` távolsággal
    - `decrypt()`, aztán `encrypt()` ugyanolyan `shift` távolsággal
    - `encrypt()` `shift` távolsággal, aztán `encrypt()` `-shift` távolsággal
    - `encrypt()` `-shift` távolsággal, aztán `encrypt()` `shift` távolsággal
    - `decrypt()` `shift` távolsággal, aztán `decrypt()` `-shift` távolsággal
    - `decrypt()` `-shift` távolsággal, aztán `decrypt()` `shift` távolsággal

2. feladat

a

Írjunk `static math.operation.Power.power()` metódust, amely természetes számok hatványozását valósítja meg (a hatvány alap és kitevő is természetes szám). Bár az eredmény mindig egész lesz, a visszatérési érték legyen `double`, mert nemsokára továbbfejlesztjük ezt a függvényt. A 0 nulladik hatványa most legyen 1.

Tesztelje a `power()` függvényt.

b

Fejlesszük tovább a `power()` függvényt úgy, hogy negatív egész kitevővel is helyesen működjön.

Próbálja ki, hogy az a feladatban megírt unit tesztek továbbra is sikeresen lefutnak-e.

Bővítse a unit teszteket negatív kitevő helyességét ellenőrző esetekkel.