Beadott megoldások

Tárgy, csoport:	Programozási nyelvek II. JAVA (BSc)
Oktató:	Kozsik Tamás, Mészáros Mónika
Feladat:	1. beadandó: ATM
Határidő:	2017-10-31, 23:59:00

Nincsenek még beadott megoldások.

A feladat szövege

A feladat összefoglaló leírása

Ebben a feladatban egy bankrendszer leegyszerűsített működését fogjuk szimulálni. A szimulációban szerepelnek bankok és személyek és egy bankjegykiadó automata, amiből az emberek pénzt tudnak felvenni, vagy pénzt tudnak betenni oda.

A részfeladatok megoldása során ügyeljünk arra, hogy a megadottakon kívül egyetlen osztály se tartalmazzon más publikus metódust vagy adattagot, illetve egyik csomag se tartalmazzon más osztályokat! Ha az implementáció megköveteli, akkor az osztályok rejtett adattagokkal és metódusokkal szabadon bővíthetők. A megoldást egyetlen `.zip` állományként kell feltölteni, amely tartalmazza a csomagnak megfelelő könyvtárszerkezetben az összes forráskódot. A fordítás során keletkező `.class` állományokat viszont már nem szabad mellékelni! A fordításhoz legalább a Java Standard Edition 8 használata kötelező.

A feladathoz tartozik egy letölthető segédlet (http://people.inf.elte.hu/bonnie/java/bead/ATM.zip), ahol találunk egy minta bemeneti fájlt és egy ahhoz tartozó kimeneti fájlt, továbbá egyéb segítséget a teszteléshez.

A feladat megoldása során be kell tartani a kódolási konvenciókat (http://people.inf.elte.hu/bonnie/java/kodminoseg.html).
A kódolási konvenciók közül külön kiemelendő, hogy ha valamilyen feladat vagy ellenőrzés elvégzésére valamelyik osztály már tartalmaz megfelelő metódust, akkor azt kell használni, nem pedig újra leprogramozni (vagy átmásolni) az adott részt. (Bizonyos helyeken ezt a feladat külön ki is emeli, de erre külön figyelmeztetés nélkül is ügyelni kell.)

A feladat részletes ismertetése

`financial.bank.Bank`

Hozzuk létre a `financial.bank` csomagban a `Bank` osztályt, ami egy felsorolási típus, az egyes bankok rövidített neveit tartalmazza.

Az osztály lehetséges értékei legyenek a következők: `BB`, `OTP`, `Erste`, `CIB`, `Raiffeisen`, `Citibank`, `FHB`, `MKB` és `UniCredit`.

`financial.person.Customer`

Hozzuk létre a `financial.person` csomagban a `Customer` osztályt, ami egy folyószámlával rendelkező személyt valósít meg. A feladatban egy embernek pontosan egy banknál van folyószámlája.

Az osztálynak legyen négy rejtett adattagja:

- `name`: szöveges típusú, a személy nevét tárolja
- `birthYear`: egész szám, a személy születési éve
- `bank`: `Bank` típusú, azt tárolja, hogy melyik banknál van a személy (egyetlen) folyószámlája
- `amount`: egész szám, a személy aktuális egyenlege a folyószámláján

Készítsük el a következő metódusokat:

- Az osztálynak legyen egy rejtett konstruktora: `Customer(String name, int birthYear, Bank bank)`, amely ellenőrzés nélkül beállítja az adattagokat a megadott értékekre, a kezdeti egyenleg legyen `0`.
- Legyen egy publikus osztályszintű `Customer` visszatérési értékű metódus: `makeCustomer(String name, int birthYear, String bankName)`, ami ellenőrzi a paramétereket, és ha azok megfelelőek, akkor létrehozza a paramétereknek megfelelő `Customer` objektumot és visszaadja. Ha a paraméterek nem megfelelőek, akkor a metódus adjon vissza `null` referenciát. A metódusnak ellenőrzéskor a következőket kell megvizsgálnia:
  - A név csak az angol abc betűiből és szóközből állhat. A név egyes részeit (vezetéknevek, keresztnevek) pontosan egy szóköz választja el egymástól. A név legalább 2, legfeljebb 4 részből áll, melyben minden rész legalább három karakter hosszú, nagybetűvel kezdődik, utána viszont csak kisbetűből állhat. (segítség: használhatod a `String` osztály `split` és `charAt` metódusát, illetve a `Character` osztály `isUpperCase` és `isLowerCase` metódusát)
  - A születési évnek `1918` és `1998` közöttinek kell lennie (az `1918` és az `1998` értéket is felveheti).
  - A `bankName`-nek a Bank típus egyik lehetséges értékével kell egyeznie.
    (segítség: szöveges típusból egy adott felsorolási típusba tartozó objektummá konvertáláshoz használhatod az adott felsorolási típus ("automatikusan" létező) osztályszintű `valueOf` metódusát, amely `IllegalArgumentException` kivételt dob, ha a szöveg nem egyezik meg a felsorolási típus egyetlen lehetséges értékével sem)
- Az osztály tartalmazza a következő publikus "getter" metódusokat: `getName`, `getBank`, `getAmount`, amelyek visszaadják az adott értékeket
- Legyen egy publikus `void` visszatérési értékű, egész paraméterű `decreaseAmount` metódus, amely az adott értékkel csökkenti az egyenleget. Semmiféle ellenőrzést nem kell végezni.

- Legyen egy publikus `void` visszatérési értékű, egész paraméterű `increaseAmount` metódus, amely az adott értékkel növeli az egyenleget.
- Legyen egy publikus szöveges visszatérési értékű, paraméter nélküli `toString` metódus, ami a következő formában adja vissza a személy szöveges reprezentációját: `Név: egyenleg`

financial.bank.ATM

Hozzuk létre a `financial.bank` csomagban az `ATM` osztályt, ami egy pénzkiadó automatát valósít meg.

Az osztálynak legyen két rejtett adattagja:

- `bank` : `Bank` típusú érték, azt tárolja, hogy melyik bankhoz tartozik az automata (az automatából más bank ügyfelei is kivehetnek pénzt, csak ők magasabb költséggel)
- `amount` : egész típusú érték, azt tárolja, hogy aktuálisan mennyi pénz van az automatában

Készítsük el a következő metódusokat:

- Az osztálynak legyen egy rejtett konstruktora: `ATM(Bank bank, int amount)` , amely a megadott adatokkal - ellenőrzés nélkül - létrehozza az objektumot.
- Legyen egy publikus osztályszintű, `ATM` visszatérési értékű metódus: `makeATM(String bankName, int amount)` , ami ellenőrzi a paramétereket, és amennyiben azok megfelelőek, akkor létrehozza és visszaadja az `ATM` objektumot. Ha a paraméterek nem megfelelőek, akkor a metódus adjon vissza `null` referenciát. Az ellenőrzéskor a következőket kell megvizsgálni:
  - A `bankName` -nek a `Bank` típus egyik lehetséges értékével kell egyeznie.
  - Az egyenlegnek pozitívnak kell lennie.
- Legyen egy publikus egész visszatérési értékű, paraméter nélküli `getAmount` metódus, amely visszaadja az automatában lévő pénz mennyiségét.
- Legyen egy publikus `void` visszatérési értékű, egész paraméterű `decreaseAmount` metódus, amely az adott értékkel csökkenti az egyenleget (semmiféle ellenőrzést nem kell végezni).
- Legyen egy publikus `void` visszatérési értékű, egész paraméterű `increaseAmount` metódus, az adott értékkel növeli az egyenleget.
- Legyen egy publikus egész visszatérési értékű `calculateFee(Bank bank, int value)` metódus, amely kiszámolja a pénzfelvételi díjat annak függvényében, hogy a felvevő folyószámlája melyik banknál van (`bank` paraméter) és mekkora összeget szeretne felvenni (`value` paraméter). A díjat a következőképpen kell számolni:
  - ha az automata a pénzfelvevő saját bankjának automatája, akkor a díj az összeg `1%` -a (felfelé kerekítve), de legalább `200 Ft` .
  - idegen bank esetén a díj az összeg `3%` -a (felfelé kerekítve), de legalább `500 Ft` .
  - (segítség: kerekítéshez használhatod a `Math` osztály `ceil` metódusát)

financial.Simulator

Hozzuk létre a `financial` csomagban az `Simulator` osztályt, ami a bankvilág szimulációját végzi.

Az osztálynak legyen három rejtett adattagja:

- `atm` : `ATM` típusú (a szimulációban egyetlen automata szerepel)
- `customers` : olyan adatszerkezet, amiben `Customer` típusú objektumokat lehet tárolni (ez lehet például tömb, `ArrayList` vagy `LinkedList` ; tömb esetén felhasználhatjuk, hogy maximum `50` felhasználó lehet a szimulációban)
- `pwLog` : `PrintWriter` típusú objektum, amelyet logfájl vezetéséhez használunk

Készítsük el a következő metódusokat:

- Az osztálynak legyen egy publikus konstruktora: `Simulator(String bankName, int initAmount, String outputFileName)`
  - Ha a paraméterek megfelelőek (lásd a `makeATM` metódusnál írtakat), akkor a konstruktor a hozza létre a megadott bankhoz tartozó automatát a megadott kezdeti összeggel. Ha az adatok nem megfelelőek, akkor a metódus dobjon `IllegalArgumentException` kivételt.
    (segítség: egy `ATM` objektum paramétereinek ellenőrzésére és helyes paraméterek esetén létrehozására már van megfelelő metódus az `ATM` osztályban, így kód-duplikáció helyett azt kell meghívni)
  - A `Customer` objektumokat tartalmazó adatszerkezet kezdetben ne tartalmazzon egyetlen objektumot sem.
  - A `PrintWriter` objektumot hozzuk létre a megadott fájlnévhez (ha a fájl nem hozható létre, vagy nem nyitható meg íráshoz, akkor a metódus engedje ki a keletkező kivételt).
- Legyen egy rejtett `Customer` visszatérési értékű szöveges paraméterű `getCustomerByName` metódus, amely visszaadja a `customers` adatszerkezetből annak a `Customer` objektumnak a referenciáját, amelynek a neve megegyezik a paraméterben megadott szöveggel. Feltételezhetjük, hogy egy név csak egyszer szerepel (ezt az `insertCustomer` metódus fogja biztosítani). Ha az adatszerkezetben nincs megfelelő objektum, akkor a metódus adjon vissza `null` referenciát.
  Az objektumot nem kell lemásolni, csak a referenciáját kell visszaadni.
- Legyen egy publikus `void` visszatérési értékű `insertCustomer(String customerName, int birthYear, String bankName)` metódus, amellyel egy `Customer` objektumot lehet felvinni a nyilvántartásba. A személy akkor vehető fel, ha ilyen nevű még nem szerepel az adatszerkezetben, továbbá ha a személy adatai megfelelőek (lásd a `makeCustomer` metódusnál írtakat). Ha a személy már szerepel a nyilvántartásban, vagy valamelyik paraméter nem megfelelő, akkor a `Customer` objektumokat tároló adatszerkezet maradjon változatlan.
  (segítség: használd a `makeCustomer` és `getCustomerByName` metódusokat)
- Legyen egy publikus `void` visszatérési értékű `withdrawCash(String customerName, int amount)` metódus, amellyel készpénzt lehet felvenni az automatából.
  - Ha ilyen nevű személy nem szerepel a nyilvántartásban (használd a `getCustomerByName` metódust), vagy ha az automatában nincs elég készpénz, vagy a felvenni kívánt összeg `0` vagy negatív, akkor a metódus nem csinál semmit.
  - Ha ilyen nevű személy szerepel a megfelelő adatszerkezetben és az automatában is van elég pénz, akkor a metódus kiszámítja a pénzfelvétel díját, majd ellenőrzi, hogy az adott személy folyószámláján van-e legalább akkora fedezet, mint amennyi a felvenni kívánt összeg és a pénzfelvételi díj együttesen. Ha a személy egyenlege nem elég nagy, akkor a metódus nem csinál semmit.

- Ha minden ellenőrzés sikeres, akkor a metódus az adott személy folyószámlájának egyenlegét az összeggel és a készpénzfelvételi díjjal csökkenti (mert a személynek a felvétel díját is meg kell fizetnie), az automatában lévő pénzösszeget viszont csak a felvenni kívánt összeggel csökkenti (mert az automata ténylegesen csak annyit ad ki).
- Ha történt készpénzfelvétel, akkor a metódus a logfájlba írja ki a felvevő szöveges reprezentációját. Ha nem történt készpénzfelvétel, akkor semmit se írjon a logfájlba.

- Legyen egy publikus `void` visszatérési értékű `depositCash(String customerName, int amount)` metódus, amellyel pénzt lehet betenni az automatába. Ha ilyen nevű személy nem szerepel a nyilvántartásban, vagy az összeg `0` vagy negatív, akkor a metódus ne csináljon semmit. Ellenkező esetben mind az automatában lévő pénzösszeget, mind az adott személy folyószámláján lévő egyenleget növelje meg az adott értékkel (a pénz betételének nincs díja). Ha történt készpénz betét, akkor a metódus a logfájlba írja ki a betevő személy szöveges reprezentációját. Ellenkező esetben semmit se írjon a logfájlba.
- Legyen egy publikus `void` visszatérési értékű `simulate(String inputFileName)` metódus, amely a megadott nevű inputfájl alapján szimulációt hajt végre a bank-világban.
  - A metódus nyissa meg olvasásra a megadott inputfájlt (ha a fájl nem létezik, vagy nem nyitható meg olvasáshoz, akkor a metódus engedje ki a keletkező kivételt).
  - Ha sikerült megnyitni a fájlt, akkor a metódus feladata, hogy azt feldolgozza.
  - Az inputfájl minden sora egy műveletet tartalmaz.
  - A művelet két részből áll, a részeket kettőspont választja el egymástól.
  - A művelet első része 3 értéket vehet fel:
    - `REG` : ebben az esetben a művelet egy személy felvétele a nyilvántartásba. Ekkor a művelet második fele a személy adatait tartalmazza vesszővel elválasztva: `név,születésiév,banknév` sorrendben. Ha a személy adatai megfelelőek, és még nem szerepel a nyilvántartásban, akkor vegyük fel. (segítség: az ellenőrzéshez és a felvitelhez használd az `insertCustomer` metódust)
    - `GET` : ebben az esetben a művelet a készpénzfelvétel. A művelet második része a felvevő nevét és a felvenni kívánt összeget tartalmazza vesszővel elválasztva. Ha az adatok megfelelőek, akkor történjen meg a készpénz felvétel.
      (segítség: használd a `withdrawCash` metódust)
    - `PUT` : ebben az esetben a művelet a készpénz betétele az automatába. Ekkor a művelet második része a betevő személy nevét és a betenni kívánt összeget tartalmazza vesszővel elválasztva. Ha az adatok megfelelőek, akkor történjen meg a készpénz betétel.
      (segítség: használd a `depositCash` metódust)
  - Ha bármelyik input-sor hibás, tehát nem a megfelelő tartalmú és/vagy mennyiségű adatot tartalmazza, akkor a sort figyelmen kívül kell hagyni, és a feldolgozás folytatódjon a következő sorral. Rossz lehet például az a sor, amelyik nem pontosan egy kettőspontot tartalmaz, vagy aminél a kettőspont előtt nem a három megadott parancs egyike szerepel, vagy amelyik nem számot tartalmaz ott, ahol számot várunk, vagy ahol nincs meg az összes szükséges adat (például a felvételnél hiányzik az összeg), stb.
  - Ha az inputfájl feldolgozása során olvasási hiba történik, akkor a metódus engedje ki a keletkező kivételt.
- Legyen egy publikus `void` visszatérési értékű, paraméter nélküli `close` metódus, amely lezárja a logfájlhoz létrehozott `PrintWriter` objektumot.

`main.Main`

Az eddigi osztályokból most már össze tudunk állítani egy parancssorból is önállóan futtatható programot, amely a parancssori paramétereknek megfelelően elkészíti a szimuláció kezdeti világát, majd egy megadott inputfájlnak megfelelően végrehajtja a szimulációt. Az eredményeket egy megadott outputfájlba menti.

Hozzuk létre a `main` csomagban a `Main` osztályt, amelyben legyen egy főprogram! A következőképpen kell viselkednie:

- A főprogram három paramétert vár: `banknév inputfájl outputfájl`
- Ha nincs megfelelő számú paraméter, akkor írjon ki egy hibaüzenetet és fejeződjön be.
- Ha három paramétert kapott, akkor a banknévvel, `1 000 000` Ft kezdeti egyenleggel és az outputfájl nevével hozza létre a `Simulator` objektumot. Ha létrehozáskor kivétel keletkezett, akkor a program egy megfelelő hibaüzenet kiírása után szabályosan fejeződön be.
- Ha létrejött a `Simulator` objektum, akkor hívja meg annak `simulate` metódusát (átadva neki az inputfájlt). Végül hívja meg az objektum `close` metódusát. Ha a metódusok végrehajtása során kivétel keletkezik, akkor a program egy megfelelő hibaüzenet kiírása után szabályosan fejeződön be.

Tesztelés

A feladathoz tartozó letölthető segédlet (http://people.inf.elte.hu/bonnie/java/bead/ATM.zip)-ben található `input.txt` -t másoljuk a megfelelő helyre. Fordítsuk le a programot és indítsuk el a következő paraméterezéssel:

```
java main.Main BB input.txt output.txt
```

Ezután a keletkező `output.txt` fájlt vessük össze a becsomagolt állományban lévővel, hogy lássuk, hogy a várt eredmény jött-e ki.

A tömörített állományban található `magyarazat.txt` nevű fájl is, ami az `input.txt` sorait tartalmazza magyarázatokkal kiegészítve: minden sornál szerepel a program elvárt viselkedése.

Figyelem: a helyes kimeneti fájl még nem feltétlenül jelenti azt, hogy a megoldás helyes.

Jó munkát!