ZH-programozási feladat

Határidő máj 26, 13:30 **Pont** 30 **Kérdések** 1
Elérhető máj 26, 10:00 - máj 26, 13:30 körülbelül 4 óra **Időkorlát** Nincs
Engedélyezett próbálkozások Korlátlan

Instrukciók

Programozási nyelvek (BSc, 18) Java zh programozási feladat

Szabályok

Az alábbi feladatleírásban azon nyelvi elemek (osztályok, metódusok stb) leírása szerepel, amelyeknek kötelező megjelennie a megoldásban. A megnevezetteken kívül további rejtett adattagok és metódusok, valamint nyilvános setter/getter metódusok felvétele megengedett. Ha a feladatleírásban meg van adva egy nyelvi elem neve, kötelező azt használni. A nyelvi szabályok mellett betartandóak a Java nyelv konvenciói is.

Ha bármilyen kérdés, észrevétel felmerül, azt az MS Teams alkalmazáson keresztül az oktatóknak vagy a gyakorlatvezetőnek kell jelezni, **nem** a diáktársaknak!

Az elkészített megoldásokat **generált fájlok (.class, .jar) nélkül** a Canvas felületen kérjük beadni. A **.jar fájlokat sem kérjük** beadni. Az elkészített megoldást csomagoljuk zip archívumba. A zip-fájlban a csomagoknak megfelelő könyvtárak is legyenek eltárolva. Linux alatt az alábbi paranccsal hozhatunk létre zip-fájlt:

A Canvas-be lehetőség van többször feltölteni és beadni megoldást. Az utoljára beadott megoldást fogjuk értékelni.

```
zip -r megoldas.zip *
```

Használható segédanyagok:

- **Letölthető zip fájl: Fájlok/base_calc.zip**, amely tartalmazza a junit tesztelő környezet jar fájljait.
- **Java dokumentáció (https://bead.inf.elte.hu/files/java)** .
- Legfeljebb egy üres lap és toll.

Ezt a kvízt ekkor zárolták: máj 26, 13:30 .

Próbálkozások naplója

Próbálkozás	Idő	Eredmény

	Próbálkozás	Idő	Eredmény
LEGUTOLSÓ	1. próbálkozás	207 perc	28.5 az összesen elérhető 30 pontból

Ezen próbálkozás eredménye: **28.5** az összesen elérhető 30 pontból
Beadva ekkor: máj 26, 13:27
Ez a próbálkozás ennyi időt vett igénybe: 207 perc

A feladat összefoglaló leírása

Ebben a feladatban megvalósítunk egy egyszerű táblázatkezelő programot (`calc`).

A táblázatkezelő programunkban sorok (rows) és oszlopok (cols) vannak, a táblázat elemeit **celláknak** (cells) nevezzük. Az oszlopokat az angol ABC nagybetűivel, a sorokat természetes számokkal indexeljük. Egyetlen adattípus használatát támogatjuk: nemnegatív egész szám.

Tekintsük a következő példát:

	A	B	C	D
0	6	5	11	5
1	2	6	8	4
2	2	9	11	5

Ebben a példában a táblázat 3 soros és 4 oszlopos. Egy konkrét cellára a cella nevével hivatkozhatunk: két index egymás mellé írva, például a B2 a `9`-et tartalmazó cellát jelenti.

Egyes cellák értékét más cellákból számolhatjuk. Például, a B2-es cella tartalma szám (a felhasználó inputja), a C0-ás cella tartalma szintén szám (a program outputja), azonban ezt a számot más cellákra hivatkozva számoltuk ki. A fenti példában a C és a D oszlopok így számolhatóak ki az A és B oszlopokból: `Ci=Ai+Bi` és `Di=Ci/2`, tehát a C oszlopban összegezzük az első két oszlopot, a D oszlopban átlagot számolunk (a táblázatkezelő programunk csak egészosztást támogat). Vegyük észre, hogy a C és D oszlopok képernyőre írásakor nem a bennük tárolt képlet íródik ki, hanem annak a végeredménye.

Cellanév és indexek (8.5 pont)

Írjon `calc.util.CellName` osztályt, melybe cellanevekkel kapcsolatos konstans és segédfüggvények fognak kerülni. Tárolja `colIndexes` osztályszintű,

módosíthatatlan sztringben a használható oszlopindexeket (`ABCDEFGHIJKLMNOPQRSTUVWXYZ`), a későbbiekben kényelmes lesz erre sztringként hivatkozni.

A felhasználó által megadott A1 alakú cellanevek ellenőrzésére és indexek használatára írjunk statikus segédfüggvényeket a `calc.util.CellName` osztályba.

Az `isCellNameValid()` függvény térjen vissza igazzal, ha a paraméterként kapott `cellName` egy megfelelő cellanév. Egy cellanév akkor megfelelő, ha az angol ABC nagybetűjével kezdődik (ezt lehet úgy ellenőrzni pl., hogy `colIndexes` tartalmazza-e `cellName` első betűjét), majd a betű után egy nemnegatív egész szám áll. (A cellanév tehát az angol ABC nagybetűjét csak az első helyen tartalmazhatja, utána pedig csak számjegyek következhetnek.) Ha `cellName` nem megfelelő, akkor a függvény visszatérési értéke hamis.

A táblázat tárolása kétdimenziós tömbbel fog történni, ezért egy cellanévről meg kell mondani, hogy a kétdimenziós tömb mely elemére hivatkozik. Például, ha a cella neve C1, akkor a sorindex 1, az oszlopindex 2 (az indexelés 0-tól kezdődik).

A `getRowIndexFromCellName()` függvény térjen vissza a paraméterként kapott `cellName` cellanévben tárolt sorindexszel.

A `getColIndexFromCellName()` függvény térjen vissza a paraméterként kapott `cellName` cellanévben tárolt oszlopindexszel. A cellanévben lévő oszlopok neveit a következőképpen kell indexszé (egész számmá) konvertálni: A -> 0, B -> 1, C -> 2, ...

Egyszerű megoldás erre, ha egy adott nagybetű esetén a `getColIndexFromCellName()` függvény visszatér a `CellName` osztály `colIndexes` sztringbeli indexével.

A `getRowIndexFromCellName()` és `getColIndexFromCellName()` függvények visszatérési típusa `int`, valamint `calc.util.SheetException` kivételt dobnak valamilyen informativ hibaüzenettel, ha a paraméterként kapott cellanév nem megfelelő. A `calc.util.SheetException` legyen egy ellenőrzött kivétel, amelyet lehet paraméter nélkül és paraméteresen is konstruálni (példányosítani). A paraméteres konstruktor meghívja a szülőosztály konstruktorát a paraméterként kapott hibaüzenettel (`String`).

A `tests.Tests` osztályban tesztelje fehérdoboz-teszteléssel az eddig megírt megoldást a következő szempontok szerint.

isCellNameValid()

- Szóközt tartalmazó cellanévre a visszatérési érték **hamis**.
- Kisbetűt tartalmazó cellanévre a visszatérési érték **hamis**.
- Nagybetűvel kezdődő, egész számmá nem konvertálható sorindexet tartalmazó cellanévre a visszatérési érték **hamis**.
- Nagybetűvel kezdődő, 10-nél kisebb pozitív sorindexre a visszatérési érték **igaz**.
- Nagybetűvel kezdődő, 9-nél nagyobb oszlopindexet tartalmazó cellanévre a visszatérési érték **igaz**.

getRowIndexFromCellName()

Egy ellenőrzés 10-nél kisebb és egy ellenőrzés 10-nél nagyobb sorindexre.

getColIndexFromCellName()

- Az A oszlop indexe 0.
- Egy A-nál későbbi betű indexe helyes.

Emlékeztető a JUnit használatához:

- Az `org.junit.Assert` osztályt (és/vagy annak statikus metódusait, pl. `assertEquals`) kell importálni, valamint az `org.junit.Test` annotációt.
- A JUnit futtatása, ha a tesztesetek osztálya a névtelen csomagba tartozó `SimpleTest` osztály, a következő parancsokkal történik.

Windows:

```
javac -cp .;junit-4.12.jar;hamcrest-core-1.3.jar SimpleTest.java
java -cp .;junit-4.12.jar;hamcrest-core-1.3.jar org.junit.runner.JUnitCore Simpl
eTest
```

Linux:

```
javac -cp .:junit-4.12.jar:hamcrest-core-1.3.jar SimpleTest.java
java -cp .:junit-4.12.jar:hamcrest-core-1.3.jar org.junit.runner.JUnitCore Simpl
eTest
```

Táblázat elemei (10 pont)

A megvalósításban minden cellában egy kiszámolható érték (`Evaluable`) található, ami egy `interface`. Egy kiszámolható érték állhat egyetlen számból (pl. B2), vagy lehet (más cellák és/vagy számokokból) számított (pl. D2). A

táblázat celláit a `Sheet` osztály fogja tárolni egy kétdimenziós tömbben.

Írjon `calc.Evaluable` néven `interface`-t, amely egyetlen metódust tartalmaz: `eval()` nevű, paraméterként `Sheet` táblázatot fogad, `int` visszatérési értékű metódus, amely `SheetException` kivételt dobhat. A metódust arra fogjuk használni, hogy egy cellába írt matematikai képlet végeredményét kiszámoljuk. A metódus törzsét a `Num` és `Equation` osztályok fogják implementálni. `Num` esetén ez könnyű, hiszen pl. a `6`-ot tartalmazó cella eredménye `6`, `Equation` esetében pl. a `6+5`-öt tartalmazó képlet végeredménye `11`.

Írjon `calc.Num` néven osztályt, amely megvalósítja a `Evaluable` `interface`-t, és egy egész szám tárolására alkalmas. Írjon paraméteres konstruktort a `Num` osztályhoz. Mivel negatív egészek tárolását nem támogatjuk, ezért a konstruktor negatív szám esetén dobjon `IllegalArgumentException` kivételt. A `Num` osztály implementálja az `eval()` függvényt úgy, hogy visszatér a tárolt számmal (a paraméterként kapott `Sheet` táblázatot nem is használja, így egy `Num` példányosításakor paraméternek adjunk meg `null` referenciát).

Írjon `calc.Equation` néven osztályt, amely megvalósítja a `Evaluable` `interface`-t. Az `Equation` osztály paraméteres konstruktora sztringként kap egy olyan képletet, amely kizárólag cellaneveket, nemnegatív egész számokat (ezek lehetnek operandusok) és a négy alapműveletet tartalmazhatja (`+ - * /`, ezek közül kerülhet ki az operátor). (Tehát zárójeleket sem tartalmazhat!) A sztringbeli képlet alakja: **operandus1 operátor operandus2**. A konstruktor ellenőrizze a sztringként kapott képletet, hogy megfelelő-e. Amennyiben nem megengedett karaktert tartalmaz, vagy nem tartalmaz operátort, dobjon `IllegalArgumentException` kivételt. Az `Equation` osztály sztringként tárolja a kapott képlet két operandusát az `operand1` és `operand2` adattagokban, és karakterként (`Character`) az operátort (műveleti jelet), ami egyetlen karakter (`operator` adattag). Egy képlet elemeinek elkülönítésére rengeteg módszer létezik, azonban ne időzzünk ezen túl sokat, hiszen tudjuk, hogy a képlet **operandus1 operátor operandus2** alakú. Kényelmes megoldás például az, ha ellenőrzés során, amikor karakterenként bejárjuk a sztring karaktereit, ha műveleti jellel találkozunk, azt rögtön elmentjük az `operator` adattagba, így az ellenőrzés után, a műveleti jel ismeretében részsztringekre vágjuk a paraméterként kapott sztringet (`split()`). Az `Equation` osztály `eval()` függvénye a táblázat ismeretében kiértékeli a tárolt képlet eredményét, azonban ezt a metódust később valósítjuk meg (a *Táblázat elemei* részfeladat lefordításához itt ideiglenesen visszatérhetünk 0-val).

Írjon `calc.Sheet` néven osztályt, amely egy táblázatot reprezentál. Egy `Sheet` konstruálásakor meg kell adni a sorok számát (`numOfRows`), az oszlopok számát (`numOfCols`), és tárolja `Evaluable`-ök ilyen méretű kétdimenziós

tömbjét. A `Sheet` osztályt paraméteres konstruktora fogadja paraméterként a sorok és oszlopok darabszámát. Mivel az oszlopokat az angol ABC negybetűivel indexeljük, ezért az oszlopok száma korlátozott. Amennyiben túl nagy az oszlopok darabszáma, vagy nem pozitív méretet adnak meg, a függvény dobjon `java.lang.IllegalArgumentException` kivételt.

A `tests.Tests` osztályban tesztelje fehérdoboz-teszteléssel az eddig megírt megoldást a következő szempontok szerint.

- Egy `Num(x)` konstruálás után a `Num` példány `eval(null)` metódusa `x`-el tér vissza.
- Az `Equation` konstruktora szóközt tartalmazó sztringre `IllegalArgumentException` kivételt dob.
- Az `Equation` konstruktora kisbetűvel megadott cellanévre `IllegalArgumentException` kivételt dob.
- Az `Equation` konstruktora nem megengedett karakterre `IllegalArgumentException` kivételt dob.

Táblázat feltöltése (11.5 pont)

Írjon `insertToSheet()` metódust a `Sheet` osztályhoz, amely `cellName` sztringet és egy `Evaluable` referencát kap paraméterként, és a kapott `Evaluable` referenciát tárolja a `Sheet` táblázatot reprezentáló tömbjében. Írjon `getFromSheet()` metódust, amely egy `cellName`-re visszatér az adott cellában lévő `Evaluable` referenciával. A tömb indexeinek megkeresésére használja a *Cellanév és indexek* részfeladatban megírt segédfüggvényeket.

Írjunk egy statikus segédfüggvényt `constructIntFromOperandStr()` néven. Ez a függvény egy egy operandust sztringként (`operandStr`) és egy `Sheet` referenciát kap paraméterként, és `int` a függvény visszatérési értéke. A függvény feladata, hogy `operandStr`-ből egész számot csináljon, attól függően, hogy cellanév vagy egész szám található benne. A segédfüggvény működése:

- Ha az `operandStr` nagybetűvel kezdődik, akkor az operandus egy cellanév, ekkor a `Sheet` `getFromSheet()` metódusával lekérdezzük a táblázatból a cella tartalmát, ami egy `Evaluable` lesz. Ennek kiértékelésével (`eval()`) megkapjuk az operandus értékét egész számként, ami a visszatérési érték.

- Ha az `operandStr` nem betűvel kezdődik, akkor abban csak egész szám

lehet, így `operandStr`-t egész számmá kell konvertálni és visszatérni vele.

Implementáljuk az `Equation` osztály `eval()` metódusát. A metódus paraméterként kap egy `Sheet` referenciát, hiszen egy képlet hivatkozhat más mezőkre is, így a képlet kiszámolásához el kell érni `Sheet` elemeit.

Az `eval()` metódus külön-külön kiszámolja a két tárolt operandusból (`operand1` és `operand2`) annak értékét (végeredményét) egész számként (azaz kétszer meg kell hívni a `constructIntFromOperandStr()` segédfüggvényt). Ekkor rendelkezésre áll a két operandus egész számként, és az `operator` adattag ismeretében elvégezzük a kért matematikai műveletet, majd annak eredményével visszatérünk. Amennyiben a kivonás negatív eredményt adna, dobjunk `ArithmeticException` kivételt, hiszen negatív számokat nem támogatunk. Amennyiben osztáskor a nevező 0, dobjunk `ArithmeticException` kivételt.

Mejegyzés: Ez a megoldás képes arra, hogy olyan cellát számoljunk, ami hivatkozik olyan cellára, amit szintén ki kell számolni (a bevezető példában pl. a D oszlop). Ez a megoldás a cellák körkörös hivatkozásánál nem terminálló rekurziót eredményez, aminek hatására elfogy a stack `StackOverflowError` kivétel kiváltásával. Ezzel most **nem** foglalkozunk.

Írjunk `toString()` metódust a `Sheet` osztályhoz, amely visszatér a táblázat sztringbeli reprezentációjával. A bevezető példában bemutatott módon minden cella végeredményét adjuk vissza.

A `toString()` metódus a szomszédos cellákat szóközzel válassza el, a sor utolsó cellája után pedig sortörés (újsor) karaktert tegyen (`System.lineSeparator()`). Az utolsó sor után ne legyen sortörés. Például, a bevezető példa így néz ki képernyőre írva:

```
6 5 11 5
2 6 8 4
2 9 11 5
```

A táblázat minden eleme egy `Evaluable`, így a sztringhez fűzéskor annak eredményét kell venni, tehát meg kell hívni annak az `eval()` függvényét az aktuális `Sheet` objektumon (ha valamely cella értéke `null` referencia, akkor a sztringbe a `null` szöveg kerüljön a cella esetében). A `toString()` ne dobjon kivételeket, ezért a fellépő kivételeket kapja el (`catch (Exception exc) {}`).

Írjon unit tesztet a táblázat kinézetének tesztelésére. Hozzuk létre a bevezető példa táblázat A és B oszlopát. Ez egy 3 soros, és 2 oszlopos `Sheet`-et eredményez. Ekkor ez elvárt eredmény:

```
6 5
2 6
```

```
2 9
```

A `toString()` visszatérési értékének tesztelésekor ne felejtsük el, hogy a sortörés (újsor) karaktereket is rögzítenünk kell. Például, az előbbi táblázatot tesztelő sztringet így állíthatjuk elő:

```
String expected = "6 5" + System.lineSeparator() + "2 6" + System.lineSeparator
() + "2 9";
```

Írjon unit teszteket `Equation` tesztelésére a következő szempontok szerint. Kiinduló `Sheet`-ként használja az előbb összeállított táblázatot.

- Egy 3 soros, 3 oszlopos táblázat C oszlopában tárolja a `Ci=Ai+Bi` képletet. Ekkor az elvárt eredmény:

```
6 5 11
2 6 8
2 9 11
```

- Az előbbi unit tesztet duplikálva, bővítsük D oszloppal a táblázatot, ahol számítsuk ki a C oszlopba írt értékek felét. Ekkor az elvárt eredmény a bevezető példa:

```
6 5 11 5
2 6 8 4
2 9 11 5
```

- Egy 3 soros, 3 oszlopos táblázat C oszlopában tárolja a `Ci=10*Bi` képletet. Ekkor az elvárt eredmény:

```
6 5 50
2 6 60
2 9 90
```

Főprogramot **nem** kell készíteni.

1. kérdés

28.5 / 30 pont

A megoldásokat ehhez a kérdéshez, ide kérjük feltölteni .zip archívumba csomagolva.

.class és .jar fájlokat **ne** adjanak be (a tesztelőkönyvtárak kivételével).

A Canvas-be lehetőség van többször feltölteni és beadni megoldást.

Az utoljára beadott megoldást fogjuk értékelni.

zh.zip (https://canvas.elte.hu/files/1182226/download)

Kvízeredmény: **28.5** az összesen elérhető 30 pontból