

Java SE Feladatsor 1

Ciklusok, Tombok, OOP alapok

Szabo Daniel
daniel.szabo99@outlook.com

2021. május 3.

Kivonat

Tartalomjegyzék

1. Tesztek Futtatasa	3
2. Szamologep	4
2.1. kivon(a,b)	4
2.2. szoroz(a,b)	4
2.3. oszt(a,b)	4
2.4. hatvany(a,b)	4
2.5. negyzetgyok(a)	5
3. Ciklusok	6
3.1. szamOsszeg(n)	6
3.2. legnagyobbPrim(n)	6
3.3. fibonacci(n)	6
3.4. legnagyobbKozosOszt(a,b)	7
4. Tombok	8
4.1. ujTomb10HosszuIntTomb()	8
4.2. ujIntTombMegadottSzamokkal(a, b, c, d)	8
4.3. tombHossz(t)	8
4.4. legnagyobb(t)	9
4.5. legkisebb(t)	9
4.6. atlag(t)	9
4.7. keres(t,k)	9
4.8. szamol(t, k)	9
4.9. tukroz(t)	9
4.10. osszefuz(t)	10
5. Versenyauto	11
5.1. get-set, konstruktor	11
5.2. gyorsitas, lassitas	11
5.3. toString	11
5.4. verseny	11
6. Megoldas beadasa	12

1. Tesztek Futtatása

Ez a feladatsor tesztekkel lett kiegészítve, ami lehetővé teszi számodra, hogy azonnal visszajelzést kapj, hogy megfelelően működik-e egy bizonyos feladatra írt megoldásod, illetve a megoldás értékelőjének is segít egy pillanat alatt ellenőrizni, hogy működőképes megoldást adtal be. A tesztelési keretrendszer működési elve miatt a feladatoknak muszáj az OOP elveit használnia, a metódusoknak paramétereket befogadnia és visszatérési értékeket adnia. Erdemes a tesztet minden változtatás után lefuttatni, hogy folyamatosan lásd, esetleg elrontottál-e valamit egy változtatással, ami korábban működött.

Segítség: Tesztek futtatása IntelliJ IDEA-ban, Tesztek futtatása Eclipse-ben

2. Szamologep

Bevezetes Ebben a feladatban a **Szamologep** osztaly fuggvenyeit fogjuk implementalni. Az egyszeruseg kedveert ebben az osztalyban csak **float** adattipust használunk számításokhoz. A feladat 5 részből áll, minden részben egy metódus hiányzó logikáját kell pótolnod. Egy példa metódus **osszead()** segítségével lett helyezve a forráskódban.

```
1 public float osszead(float a, float b){  
2     return a + b;  
3 }
```

Kodreszlet 1. Pelda Metodus

2.1. kivon(a,b)

A **kivon** függvény a két parameterként kapott szám különbséget kell visszaadja, például:

kivon(4,3) → 1
kivon(-2.5,3) → -5.5
kivon(0,2.25) → -2.25

2.2. szoroz(a,b)

A **kivon** függvény a két parameterként kapott szám szorzatát kell visszaadja, például:

szoroz(4,3) → 12
szoroz(-2.5,3) → -7.5
szoroz(0,2.25) → 0

2.3. oszt(a,b)

Az **oszt** függvény a két parameterként kapott szám hányadosát kell visszaadja, például:

oszt(4, 2) → 2
oszt(-4, 8) → -0.5
oszt(3, 0) → **hiba**(-∞)

2.4. hatvany(a,b)

A **hatvany** függvény a két parameterként kapott szám hatványát kell visszaadja, például:

hatvany(2,3) → 8

```
hatvany(2,-2) → 0.5  
hatvany(0,2.25) → 0
```

2.5. negyzetgyok(a)

A `negyzetgyok` függvény a két parameterkent kapott szám különbséget kell visszaadja, például:

```
negyzetgyok(4) → 2  
negyzetgyok(1) → 1  
negyzetgyok(-2) → hiba (-1)
```

3. Ciklusok

Ebben a feladatban a `Ciklusok` osztály függvényeit fogjuk implementálni, aminek során ciklusos algoritmusokat gyakorlunk. Az egyszerűség kedvéért ebben az osztályban csak `int` adattípust használunk számításokhoz. A feladat 4 részből áll, minden részben egy metódus hiányzó logikáját kell pótolnod. Egy példa metódus `szamOsszeg10ig()` segítségével el lett helyezve a forráskódban. A feladatok megoldásához javasolt a `for(int i = 0; i < n; i++){...}` struktúra használata, de más megoldás is megengedett (`while(condition), for(Object o: objects){...}`)

Segítség: docs.oracle.com - The for statement.

```
1 public int szamOsszeg10ig(){
2     int eredmeny = 0;
3
4     for (int i = 1; i <= 10; i++) {
5         eredmeny = eredmeny + i;
6     }
7
8     return eredmeny;
9 }
```

Kodreszlet 2. Példa Metódus

3.1. szamOsszeg(n)

A `szamOsszeg` 0 és `n` között (inkluzív) lévő összes egész szám összeget adja vissza eredményül, például:

`szamOsszeg(-3) → -6`

`szamOsszeg(0) → 0`

`szamOsszeg(4) → 10`

3.2. legnagyobbPrim(n)

A `legnagyobbPrim` 0 és `n` között (inkluzív) a legnagyobb prímszámot adja eredményül, például:

`legnagyobbPrim(0) → hiba(-1)`

`legnagyobbPrim(2) → 2`

`legnagyobbPrim(10) → 7`

3.3. fibonacci(n)

Az `fibonacci` a fibonacci sorozat `n`-edik elemét adja vissza

(`[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...]`), ahol `fibonacci(0) = 0`, például:

`fibonacci(-1) → hiba (-1)`

```
fibonacci(1) → 1  
fibonacci(7) → 13
```

3.4. legnagyobbKozosOszto(a,b)

A legnagyobbKozosOszto függvény a két parameterként kapott szám legnagyobb közös osztóját kell visszaadja, például:

```
legnagyobbKozosOszto(0, 4) → hiba(-1)  
legnagyobbKozosOszto(3, 7) → 1  
legnagyobbKozosOszto(54, 24) → 6
```

Segítség: Wikipedia - Legnagyobb Közös Osztó.

4. Tombok

Ebben a feladatban a `Tombok` osztály függvényeit fogjuk implementálni, aminek során tombok használatát gyakoroljuk. Az egyszerűség kedvéért ebben az osztályban csak `int` típusú tombot használunk számításokhoz, egyedül a 10. feladat használ `String` típust. A feladat 10 részből áll, minden részben egy módszer hiányzó logikáját kell pótolnod. Egy példa módszer `szum()` segítségével el lett helyezve a forráskódban. A feladatok megoldásához javasolt a `for(int i = 0; i < n; i++){...}` struktúra használata, de más megoldás is megengedett (`while(condition)`, `for(Object o: objects){...}`)

Segítség: docs.oracle.com - Arrays.

```
1 public int szum(int[] tomb) {  
2     int eredmeny = 0;  
3  
4     for (int i = 1; i <= 10; i++) {  
5         eredmeny = eredmeny + tomb[i];  
6     }  
7  
8     return eredmeny;  
9 }
```

Kódreszlet 3. Példa Metodus

4.1. uj10HosszuIntTomb()

A `uj10HosszuIntTomb` egy 10 hosszúságú `int` tombot ad vissza. A tomb minden eleme 0 legyen.

4.2. ujIntTombMegadottSzamokkal(a, b, c, d)

A `ujIntTombMegadottSzamokkal` 4 megadott számmal tölt fel egy 4 hosszúságú tombot, például:

`ujIntTombMegadottSzamokkal(2, 3, 8, 3) → [2, 3, 8, 3]`

4.3. tombHossz(t)

Az `tombHossz` a parameterként kapott tomb hosszát adja vissza eredményül, például:

`tombHossz([]) → 0`

`tombHossz([9, 23]) → 2`

4.4. legkisebb(t)

A `legnagyobb` függvény a két parameterként kapott tomb legkisebb értékét adja vissza, például:

```
legnagyobb([]) → hiba(-1)
legnagyobb([3, 4, 1, 6]) → 1
```

subsectionlegnagyobb(t)

A `legnagyobb` függvény a két parameterként kapott tomb legnagyobb értékét adja vissza, például:

```
legnagyobb([]) → hiba(-1)
legnagyobb([3, 4, 1, 6]) → 6
```

4.5. atlag(t)

A `legnagyobb` függvény a két parameterként kapott tomb értékeinek átlagát adja vissza `int`-ként, például:

```
legnagyobb([]) → 0
legnagyobb([2, 0, 5, 7]) → 3
```

4.6. keres(t,k)

A `keres` függvény a parameterként kapott tombban a parameterként kapott int értéket keresi meg, és ha megtalálja akkor a szám tombon belüli indexét adja vissza eredményül, például:

```
keres([1, 3, 6], 3) → 1
keres([1, 3, 6], 2) → hiba(-1)
```

4.7. szamol(t, k)

A `szamol` függvény a parameterként kapott tombban a parameterként kapott int értéket keresi meg, és megszámlálja, hányszor fordul elő benne, majd ezt a számot adja vissza eredményül, például:

```
szamol([1, 3, 3], 1) → 1
szamol([1, 3, 3], 3) → 2
szamol([1, 3, 3], 2) → 0
```

4.8. tukroz(t)

A `tukroz` a parameterként kapott tombot átrendezi úgy, hogy az elemek eredeti sorrendjét megfordítja. Fontos, hogy ez a függvény nem tér vissza eredménnyel, hanem magát a parameterként kapott tombot módosítja, például:

```
tukroz([]) → []
```

```
tukroz([1, 2, 3]) → [3, 2, 1]
```

4.9. osszefuz(t)

A `osszefuz` a parameterként kapott `String` tömb elemeit egy `String` objektumra fűzi össze és azt az objektumot adja vissza eredményül, például:

```
osszefuz(["j", "a", "v", "a"]) → "java"
```

```
osszefuz(["hello", " ", "world!"]) → "hello_world!"
```

```
osszefuz([]) → ""
```

5. Versenyauto

Ebben a feladatban a `verseny` csomag osztályainak (`verseny.Verseny`, `verseny.Versenyauto`, `verseny.Versenyzo`) függvényeit fogjuk implementálni, aminek során az objektum orientált programozási szemlelet alapjait gyakoroljuk. A feladat 3 fő részből áll, minden részben egy vagy több módszer hiányzó logikáját kell pótolnod.

Segítség: docs.oracle.com - Classes, docs.oracle.com - Methods.

5.1. get-set, konstruktor

Implementáld a `Versenyauto` és `Versenyzo` osztályok teljes konstruktorait, get-set metódusait.

5.2. gyorsítás, lassítás

Implementáld a `Versenyauto` osztály `gyorsit` és `lassit` metódusait. Mindkét metódus a `versenyauto` sebességet módosítja a paraméterként kapott mennyiséggel, amíg el nem éri az auto a maximális sebességet (200km/h) vagy meg nem áll (0km/h). Ha az auto tömege 1000kg felett van, gyorsítani és lassítani is egyszerre csak 10km/h-al lehet, alatta 15km/h-val. Ha ezeknél az értékeknél nagyobbat kap a függvény, akkor is csak 10 vagy 15km/h-val fogja módosítani a sebességet. Negatív bemenetet egyik függvény sem kezel, a sebesség maradjon változatlan.

5.3. toString

A `Versenyauto` osztály `toString` függvénye a `versenyauto` adatait egy `String` objektumban összefoglalja és azt az objektumot adja vissza eredményül. A `String`-nek a következő formátumot kell követnie egy példa `Versenyauto` eseteiben:

1	Név: Michael Schumacher (1969)
2	Auto: Mercedes (740kg)

Kodreszlet 4. Példa `toString` kimenet

5.4. verseny

A `Verseny` osztály `verseny` függvénye két `versenyauto` példányt kap bemenetként, a két autót megversenyezteti és a verseny nyertesének nevet adja vissza eredményül. A verseny nyertesét a következő képpen döntjük el: ha az egyik auto tömege 100kg-al vagy többel alacsonyabb mint a másik auto tömege, akkor a könnyebbik auto fog nyerni. Ha a különbség a tömegek között kevesebb mint 100kg, akkor véletlenszerűen 50-50% eséllyel nyerhet bármelyik versenyző.

6. Megoldas beadása

Ha minden feladatot megoldottal és az összes teszt sikeresen lefut, akkor győződj meg róla, hogy a kódod tiszta és olvasható, nem tartalmaz felesleges, kommentelt vagy nem működő kódot, majd a teljes projektet (tehát nem csak a src mappát vagy csak a .java fájlokat) csomagold be egy .zip kiterjesztésű fájlba, aminek a neve legyen például `DanielSzabo_JavaSE1.zip`. Ezt a fájlt utána vagy el kell küldened e-mail-ben vagy egy megadott Google Drive mappába kell feltöltened. Eről utasítást szóban fogsz kapni/kaptal.