Programozás I. - 1. ZH, 2015.03.23	**Név: .....**
A. csoport	**EHA: .....**
25 pont	**Gyakorlatvezető: .....**

1. Írj futtatható programot, ami a parancssorban kapott számokból minden egymást követő három elemből kiszámolja a számok (mint oldalhosszak) által leírt háromszög területét a Hérón-képlet segítségével (tehát először az első három szám által meghatározott háromszöget, aztán a második-harmadik-negyedik szám által meghatározott háromszöget), stb... Tárold el melyik három számból készíthető háromszög területe volt a legnagyobb, és ezt írd ki az alapértelmezett kimenetre. *(Tipp: gyökvonáshoz használd a* `Math.sqrt(szam)` *metódust.)* ***(6 pont)***

$T = \sqrt{s(s-a)(s-b)(s-c)}$
$s = \frac{a+b+c}{2}$
	`>java Elso 4 3 6 8 5 4 2`
	`>A legnagyobb háromszög a 6, 8, 5,`
	`számokból képezhető, területe: 14.9812`

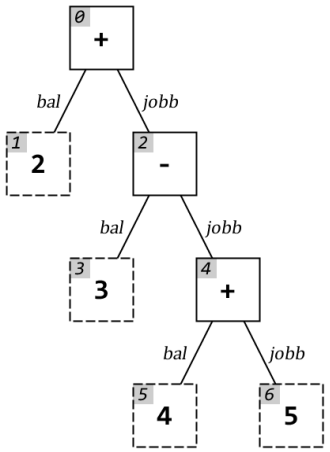
2. Írj egy `Kifejezes` nevű osztályt, amely konstans kifejezéseket reprezentál. Helyezd az osztályt a `kifejezesek` csomagba. Egy kifejezésnek csak az értékét (egész szám) tároljuk el. Ez az adattag csak az osztály és leszármazottai számára legyen látható, de legyen lekérdezhető és beállítható publikus metódusokon keresztül. Az osztály rendelkezzen egy olyan konstruktorral, amely a kifejezés értékét az alapértelmezett nullára állítja, továbbá rendelkezzen egy olyan konstruktorral is, amely a kapott paraméter alapján állítja be ezt. Definiáld felül az osztályban a `toString()` metódust úgy, hogy a kifejezés értékét írja ki. *(Tipp: szöveggé alakításhoz használd a* `String.valueOf(ertek)` *metódust.)****(5 pont)***

3. A `kifejezesek` csomagban hozz létre egy `Muvelet` osztályt, amely kétoperandusú műveleteket reprezentál és a `Kifejezes` osztályból származik. Egy művelet tárolja az operátorát (összeadás, vagy kivonás), illetve a bal és jobboldali operandusát (azaz két `Kifejezes`-t) és rendelkezik egy olyan adattaggal, amely eltárolja, hogy a kifejezés kiértékelése megtörtént-e már. Az operátor és az operandusok legyenek beállíthatóak publikus metódusokon keresztül és a beállításukkor a kifejezés állítódjon kiértékeletlenre. Írj kétféle paraméteres konstruktort. Az egyik csak a művelet operátorát, a másik a művelet operandusait is állítsa be. Használd az ősosztály konstruktorát. A művelet kezdetben legyen mindig kiértékeletlen. Készíts egy csak az osztályból látható `kiertekel()` metódust, amely lekéri az operandusok értékeit, ezután ezeken az értékeken elvégzi az operátornak megfelelő műveletet, végül pedig visszatér az így kapott eredménnyel. Abban az esetben, ha a művelet operátora nem összeadás, vagy kivonás, akkor a kiértékelés eredménye legyen nulla és írasd ki az alapértelmezett kimenetre, hogy `"Ismeretlen operator: [operator]"`. Definiáld felül az érték lekérő és beállító metódusokat. A lekérő metódus a művelet értékét adja vissza. A kiértékeletlen művelet értékét először ki kell számolni a `kiertekel()` metódus használatával. A beállító metódus pedig írja ki az alapértelmezett kimenetre, hogy `"Nem ertelmezheto muvelet!"`. Készíts `toString()` metódust is, amely a művelet kifejezést írja ki az alapértelmezett kimenetre `"([operandus1][operator][operandus2])"` formában. ***(9 pont)***

4. Írj egy csomagon kívüli futtatható osztályt `Szamologep` néven, amely a paraméterként kapott kifejezés értékét számolja ki. A parancssori paraméterek alapján hozd létre a megfelelő kifejezéseket és tárold el őket egy tömbben. A bejárást a parancssori argumentum tömb utolsó elemével kezd, és haladj visszafele:

amennyiben az aktuális (`i`-edik) paraméter egy operátor (“`+`”, vagy “`-`”), akkor művelet kifejezést hozz létre, melynek az operandusai legyenek a tömb két rákövetkező (azaz az `i+1` és `i+2` pozíción lévő) eleme. Különben hozz létre konstans kifejezést. Végül írasd ki az alapértelmezett kimenetre a tömb első kifejezését és annak értékét “`[kifejezes]=[ertek]`” formában. *(Tegyük fel, hogy a felhasználó mindig helyesen paraméterezi fel a programot, így hibakezeléssel nem kell foglalkoznod.)* ***(5 pont)***

```
>java Szamologep + 2 - 3 + 4 5
>(2+(3-(4+5)))=-4
```



Programozás I. - 1. ZH, 2015.03.23 **Név: .....**

B. csoport **EHA: .....**

25 pont **Gyakorlatvezető: .....**

1. Írjunk futtatható programot, ami parancssorban kapott egész típusú paraméterek sorozatából kiválasztja a két legkisebb, és két legnagyobb elemet, és ha páratlan számú paraméter van, akkor az eredeti sorozat középen elhelyezkedő elemét (ha ezt már egyszer belevettük, akkor megint vesszük). Ezekből számoljunk számtani átlagot, terjedelmet (legnagyobb és legkisebb elem közti különbség), ezeket írjuk ki. (**Tipp**: az `Arrays.sort(tomb)` statikus metódussal sorba rendezhető a paraméterül adott tömb.) ***(6 pont)***

```
java valami 4 10 4 23 5 10
>Atlag = 10.25 Terjedelem = 19     //(4+4+23+10)/4

java valami 4 10 4 23 5 10 4
>Atlag = 12.8  Terjedelem = 27     //(4+4+23+10+23)/5
```

2. Készíts egy `Harcos` nevű osztályt. Egy harcosnak tároljuk el az *ütőerejét*, és az *egészségét*. Ezek legyenek privát láthatóságú adattagok, egész értékű számok. Ezekhez írjunk publikus lekérdező és módosító metódusokat. A Harcosnak legyen egy olyan konstruktora, ami paraméterül várja ezt a két értéket. Default konstruktor ne legyen. Definiáljuk felül az osztályban a `toString()` metódust, és írjuk ki az adattagjai értékét, pl.: "`Harcos vagyok, ütőerőm: [ütőerő], egészségem: [egészség].`". A harcosnak legyen egy `megut()` publikus metódusa, ami nem ad vissza értéket, egy másik Harcos-t vár paraméterül, és a paraméterként kapott Harcosnak az egészségéből levonja az aktuális harcos ütőerejét. Írja ki a metódus, hogy a harcos megütötte a másikat, ehhez felhasználva a megírt `toString()` metódust. Helyezd a Harcos osztályt `harcosokklubja` csomagba! ***(6 pont)***

3. Legyen szintén a `harcosokklubja` csomagban egy `Nagydarab` nevű osztály is, ami származzon a Harcosból. A Harcos adattagjain kívül legyen egy szintén privát *állóképesség* adattagja, ez is legyen lekérdezhető és beállítható publikus metódusokon keresztül. A `Nagydarab` osztály egészségének lekérdezésekor mindig adódjon hozzá az állóképessége is a visszaadott értékhez. A `Nagydarab` osztálynak is legyen paraméteres konstruktora, amiben bekéri az adattagjai értékét. Használjuk fel a konstruktorban a ősosztály konstruktorát is! Írjuk felül a `toString()` metódust, ami hívja meg a ősosztály ugyanilyen metódusát is, és ezen kívül még írja ki, hogy egy Nagydarab harcosról van szó, valamint hogy mennyi az állóképessége. ***(5 pont)***

4. Írjunk egy csomagon kívüli futtatható osztályt! Az osztály rendelkezzen egy statikus `meccs(Harcos h1, Harcos h2)` metódussal, melynek nincs visszatérési értéke. Ezen belül először írjuk ki, melyik két harcos küzd, aztán a harcosok felváltva megütik egymást háromszor a saját `megut()` metódusukkal. A meccs végén ha mindkettőnek elfogyott az egészsége, vagy egyiknek sem, akkor döntetlen. Ha csak az egyiknek a kettejük közül, akkor a másik harcos győzött. A meccs eredményét írjuk ki a képernyőre (azt is, hogy melyik harcos győzött, ha győzött valaki).

A futtatható osztályban a parancssori argumentumokat bejárva hozzuk létre a harcosainkat: ha az aktuális argumentum `"Harcos"`, akkor a következő két paraméter alapján hozzunk létre Harcos objektumot, ha az aktuális argumentum `"Nagydarab"`, akkor a következő három paraméter alapján hozzunk létre egy

Nagydarab objektumot! Tároljuk ezeket egy tömbben. Tartsuk nyilván, hogy hány objektumot hoztunk létre, és ha ez legalább kettő, akkor indítsunk egy meccset: kezdetben az első két Harcost adjuk át neki, majd a másodikat, harmadikat, majd a harmadikat, negyediket, stb… ***(8 pont)***

```
Példa: java valami Nagydarab 2 4 2 Harcos 4 6 Nagydarab 3 3 3
```

Programozás I. - 1. ZH, 2015.03.23 **Név: .....**

C. csoport **EHA: .....**

25 pont **Gyakorlatvezető: .....**

1. Írj futtatható programot, ami parancssorban kapott `String` típusú paraméterek sorozatából kiválogatja a férges gyümölcsöket! Amennyiben a sorozatban szerepel egy `fereg`, az azt követő gyümölcs férges lesz. Számoljuk meg a férges gyümölcsöket, és a számukat írassuk ki a standard hibakimenetre. Amennyiben a sorozatban nincsen férges gyümölcs, írassuk ki a kapott gyümölcsöket fordított sorrendben. Figyelj azonban, mert a sorozatban egymás után tetszőleges számú féreg lehet, az ugyanúgy egy féregnek fog számítani, amennyiben van következő gyümölcs. ***(6 pont)***

```
>java Valami alma narancs fereg mango alma fereg alma banan
 2 darab ferges gyumolcs volt a talban.
>java Valami alma fereg fereg narancs mango alma fereg alma fereg
 2 darab ferges gyumolcs volt a talban.
>java Valami alma narancs mango alma alma banan
 Gyumolcsok visszafele: banan alma alma mango narancs alma
```

2. Írj egy `Macska` nevű osztályt. Egy macskának tároljuk a marmagasságát (lebegőpontos szám) és lábai számát (egész szám). Ezek az adattagok csak az osztályon belül legyenek láthatóak, de legyenek lekérdezhetők és beállíthatók publikus metódusokon keresztül. Az osztály rendelkezzen egy olyan konstruktorral, mely beállítja a `marMagassag` és `labakSzama` adattagokat alapértelmezetten 15.0 és 4 értékre, továbbá rendelkezzen egy olyan konstruktorral, amely kapott paraméterek alapján állítja be ezeket. Definiáld felül az osztályban a `toString()` metódust úgy, hogy kiírja a marmagasságát és lábai számát. pl.: ”`Marmagassag: [marMagassag] cm. Labak szama: [labakSzama] darab.`”. Helyezd az osztályt a `haziallatok` csomagba! ***(5 pont)***

3. Írj egy `Sziami` és egy `Cirmos` osztályt, ezek származzanak a `Macska` osztályból, és szintén a `haziallatok` csomagban legyenek! A `Sziami` osztálynak legyen egy `kenyes` (logikai) adattagja, amely csak az osztályon belül látható, de publikus metódusokon keresztül legyen lekérdezhető és módosítható, a `Cirmos` osztályban legyen egy `pofonErosseg` (egész szám) adattag, amely szintén csak az osztályon belül legyen elérhető, de lehessen lekérdezni egy publikus metóduson keresztül, és az értéke legyen 3! Mindkét osztály használja az ősosztály konstruktorát, valamint mindkét osztály paraméteren keresztül állítsa be az új adattagjait! Definiáld felül ezekben az osztályokban is a `toString()` metódust. pl: ”Cirmos cica [`marmagassag`] cm magas, [`labakSzama`] darab laba van, és [`pofonerosseg`] erosseggel gyepalja a szomszed macskat.”. ***(6 pont)***

4. Írj egy csomagon kívüli futtatható osztályt! Az osztálynak legyen egy `harcol()` metódusa, amely két `Macska` objektumot vár paraméterként és a visszatérési értéke az a macska objektum, amelyik győztes a harcban. A harc győztese a következő szabályok alapján kerül ki:

- ha valamelyik macskának több lába van, ő győzött.
- ha valamelyik macskának nagyobb a marmagassága legalább 2.5 cm-rel, ő győzött.
- ha az egyik macska kényes, akkor a másik nyert.

- egyébként az első macska nyer.

A metódus írja ki melyik macska nyert, és miért. A futtatható osztályban a parancssori argumentumok alapján állíts elő három macska objektumot a következőképpen: ha az aktuális argumentum `Cirmos`, akkor a következő két paraméter alapján hozz létre `Cirmos` objektumot, ha az aktuális argumentum `Sziami`, akkor a következő három paraméter alapján hozz létre egy `Sziami` objektumot! A macska objektumokat tárold el egy tömbben. A létrehozott macskák mérkőzzenek meg egymással a `harcol()` metódus segítségével!

Rendezd a macska tömböt úgy, hogy a tömb első eleme az abszolút nyertes macska legyen! ***(8 pont)***

```
>java valami Cirmos 15.0 4 Sziami 14.2 4 true Sziami 13.2 4 false
```

Programozás I. - 1. ZH, 2015.03.23 **Név: .....**

D. csoport **EHA: .....**

25 pont **Gyakorlatvezető: .....**

1. Írj futtatható programot, ami parancssori paraméterekként kapott egész típusú számok sorozatáról megállapítja, hogy azok piramis számokat alkotnak-e (az n-edik piramis szám az első n db egész szám négyzetének összege). Amennyiben igen, írasd ki az alapértelmezett kimenetre, hogy "A sorozat piramis számokat alkot.". Ellenkező esetben írasd ki, hogy melyik elemnél tér el a sorozat a piramis számoktól, a következő formában: "`A sorozat nem alkot piramis számokat, az első eltérő elem: x`". ***(6 pont)***

$$P_n = 1^2 + 2^2 + 3^2 + 4^2 + \cdots + n^2$$

```
>java PiramisSzamok 1 5 14 30
A sorozat piramis számokat alkot.

>java PiramisSzamok 1 5 13 40
A sorozat nem alkot piramis számokat, az első eltérő elem: 13
```

2. Készítsünk to-do listát! Írj egy `Teendo` osztályt, melyet helyezz a `todolist` csomagba! Tároljunk benne egy teendőt, amelynek tudjuk a *nevét* (szöveg), *idejét* (szöveg), *prioritását* (1-5 közötti egész szám), valamint azt, hogy már *teljesítettük-e* (logikai). Az adattagok csak osztályon belül legyenek láthatók, de legyenek lekérdezhetők és beállíthatók publikus metódusokon keresztül, kivéve a logikai adattagot. Ennek az adattagnak is legyen lekérhető az értéke publikus metódus segítségével, viszont a beállítására hozz létre egy `atvalt()` metódust, ami a logikai értéket negálja (igazból hamist, hamisból igazat csinál). Készíts egy default konstruktort, amely valahogy inicializálja az adattagokat, valamint egy paraméteres konstruktort, amely a paraméterek alapján inicializálja az adattagokat. Természetesen, a teljesítettük-e adattag értéke mindig hamis, így ezt ne várja a konstruktor. Figyelj, hogy a prioritás csak 1-5 közötti egész lehet, minden más esetben írj ki egy figyelmeztető szöveget az alapértelmezett hibakimenetre, és állítsd be a prioritást 5-re. Definiáld felül a `toString()` metódust, hogy a teendő leírását adja vissza, pl.: "Teendő: [név], melynek ideje: [időpont], prioritása: [prioritás], teljesítve: [igen|nem]." ***(7 pont)***

3. Készítsük el a `Megbeszélés`, `Bevásárlás` osztályokat, amelyek a `Teendo` osztályból származnak, és a `todolist` csomagban találhatók. Minden megbeszélésről tároljuk el azt, hogy *kivel történik* (szöveg), és *hol* (szöveg). A bevásárlásról tároljuk el, hogy *mit* (miket) szeretnénk vásárolni (szöveg), valamint, hogy *mekkora összeget* szeretnénk költeni (egész). Ezeket az adattagokat csak osztályon belül tegyük láthatóvá, viszont legyenek lekérdezhetők és beállíthatók publikus metódusokon keresztül. Készíts mindkét osztálynak paraméteres konstruktort, amelyek az ősosztály konstruktorát is felhasználják. Minden megbeszélés teendő neve "Megbeszélés", prioritása 1, míg minden bevásárlás neve "Bevásárlás", prioritása 3. Definiáld felül a `toString()` metódust mindkét osztályban, az adott osztálynak megfelelően. ***(5 pont)***

4. Írj egy csomagon kívüli futtatható osztályt! Az osztálynak legyen egy `teendok()` metódusa, amely paraméterül egy `Teendo` tömböt vár. A metódus menjen végig a tömbön, először írja ki azokat a teendőket, melyeknek prioritásuk 1, de még nem teljesítettük őket, majd ezekre hívjuk meg az `atvalt()`

metódust. Ezt ismételjük 2, 3, 4, 5 prioritásra is. A futtatható osztály main függvényében a parancssori argumentumokat bejárva hozzunk létre egy `Teendo` tömböt a következők szerint. Ha az i. argumentum :

- "`Teendo`", a következő három argumentumban van a teendő neve, ideje, prioritása.
- "`Megbeszeles`", a következő 3 adattagban van a megbeszélés ideje, a személy, akivel a megbeszélés történik, és a megbeszélés helye.
- "`Bevasarlas`", a következő három argumentum tartalmazza a bevásárlás tervezett idejét, a bevásárlás tárgyát, valamint az összeghatárt.

Hívjuk meg az így elkészült tömbre a `teendok()` metódust. ***(7 pont)***