

**Minden feladatot kötelező beküldeni!**

**A beküldési határidő: október 11., 23:00 óra**

**I. feladatsor (ezen feladatsor feladatai 1 projektbe kerülnek, külön csomagokba)**

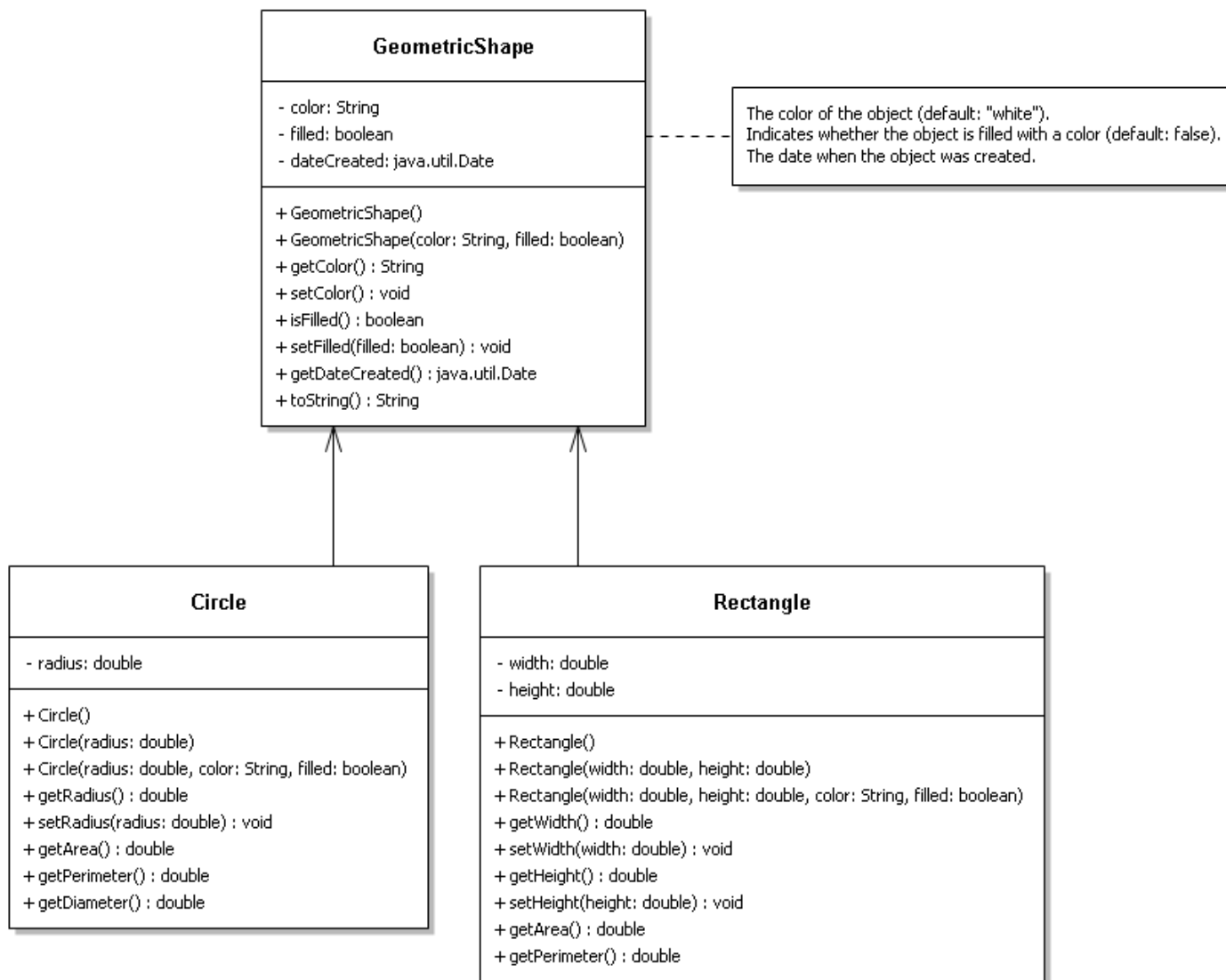
**1. Feladat:**

Tegyük fel, hogy olyan geometriai objektumok modellezésére szeretne osztályokat tervezni, mint például a körök és a téglalapok. A geometriai objektumoknak számos közös tulajdonságuk és viselkedésmódjuk van. Egy bizonyos színnel rajzolja meg őket, kitöltve vagy kitöltetlenül (csak a körvonalukat). Ennélfogva használhat egy általános `GeometricShape` osztályt az összes geometriai objektum modellezésére. Ez az osztály tartalmazza a `color` és `filled` tulajdonságokat, valamint az őket kezelő `get` és `set` metódusokat. Tegyük fel, hogy ez az osztály tartalmazza a `dateCreated` tulajdonságot is, a `getDateCreated()` és `toString()` metódusokkal együtt. A `toString()` metódus visszaadja az objektum sztring reprezentációját. Mivel a kör egy speciális fajta geometriai objektum, megosztja a közös tulajdonságokat és metódusokat más geometriai objektumokkal. Ezért van értelme a `Circle` osztályt a `GeometricShape` osztály kiterjesztéseként definiálni. Hasonlóan, a `Rectangle` osztály is deklarálható a `GeometricShape` osztály alosztályaként. A következő ábrák mutatják a kapcsolatokat ezen osztályok között. Az ábrákon a szuperosztály irányába mutató nyíl jelzi két osztály között az öröklődési kapcsolatot.

A `Circle` osztály örökli az összes hozzáférhető adatmezőt és metódust a `GeometricShape` osztálytól. Van neki továbbá egy új adatmezője, a `radius`, valamint a hozzá tartozó `get` és `set` metódusok. Tartalmazza ugyanakkor a `getArea()`, a `getPerimeter()` és a `getDiameter()` metódusokat is, amelyek a kör területének, kerületének és átmérőjének az értékét adják vissza.

A `Rectangle` osztály örökli az összes hozzáférhető adatmezőt és metódust a `GeometricShape` osztálytól. Van neki továbbá egy `height` és egy `width` adatmezője, valamint a hozzájuk tartozó `get` és `set` metódusok. Tartalmazza ugyanakkor a `getArea()` és a `getPerimeter()` metódusokat is, amelyek a téglalap területének és kerületének értékével térnek vissza.

Készítse el a feladat leírásának és a lenti osztálydiagramoknak megfelelő osztályokat!



Az elkészült osztályok teszteléséhez használhatja a következő Java osztályt.

Java:

```

public class TestCircleRectangle {
    public static void main(String[] args) {
        Circle circle = new Circle(1.0);
        System.out.println("A circle " + circle.toString());
        System.out.println("The radius is " + circle.getRadius());
        System.out.println("The area is " + circle.getArea());
        System.out.println("The diameter is " + circle.getDiameter());

        Rectangle rectangle = new Rectangle(2.0, 4.0);
    }
}
  
```

```

        System.out.println(System.getProperty("line.separator") + "A rectangle " +
rectangle.toString());
        System.out.println("The area is " + rectangle.getArea());
        System.out.println("The perimeter is " + rectangle.getPerimeter());
    }
}

```

A fenti tesztprogram futtatásának egy lehetséges kimenete:

A circle created on Sat Oct 22 20:03:10 CEST 2011

color: white and filled: false

The radius is 1.0

The area is 3.141592653589793

The diameter is 2.0

A rectangle created on Sat Oct 22 20:03:10 CEST 2011

color: white and filled: false

The area is 8.0

The perimeter is 12.0

## 2. Feladat:

(A ***Triangle*** osztály.) Tervezzon egy Triangle nevű osztályt, amely kiterjeszti a GeometricShape osztályt! Az osztály tartalmazzon

- három double típusú, side1, side2 és side3 nevű adatmezőt, alapértelmezés szerint 1.0 értékekkel a háromszög három oldalának a tárolására;
- egy konstruktort, amely a megadott side1, side2 és side3 oldalakkal hoz létre egy háromszöget;
- hozzáférő metódusokat mindhárom adatmezőhöz (getterek és setterek);
- egy getArea() nevű metódust, amely visszaadja a háromszög területét;
- egy getPerimeter nevű metódust, amely visszaadja a háromszög kerületét;
- egy toString() metódust, amely visszaadja a háromszög sztring reprezentációját.

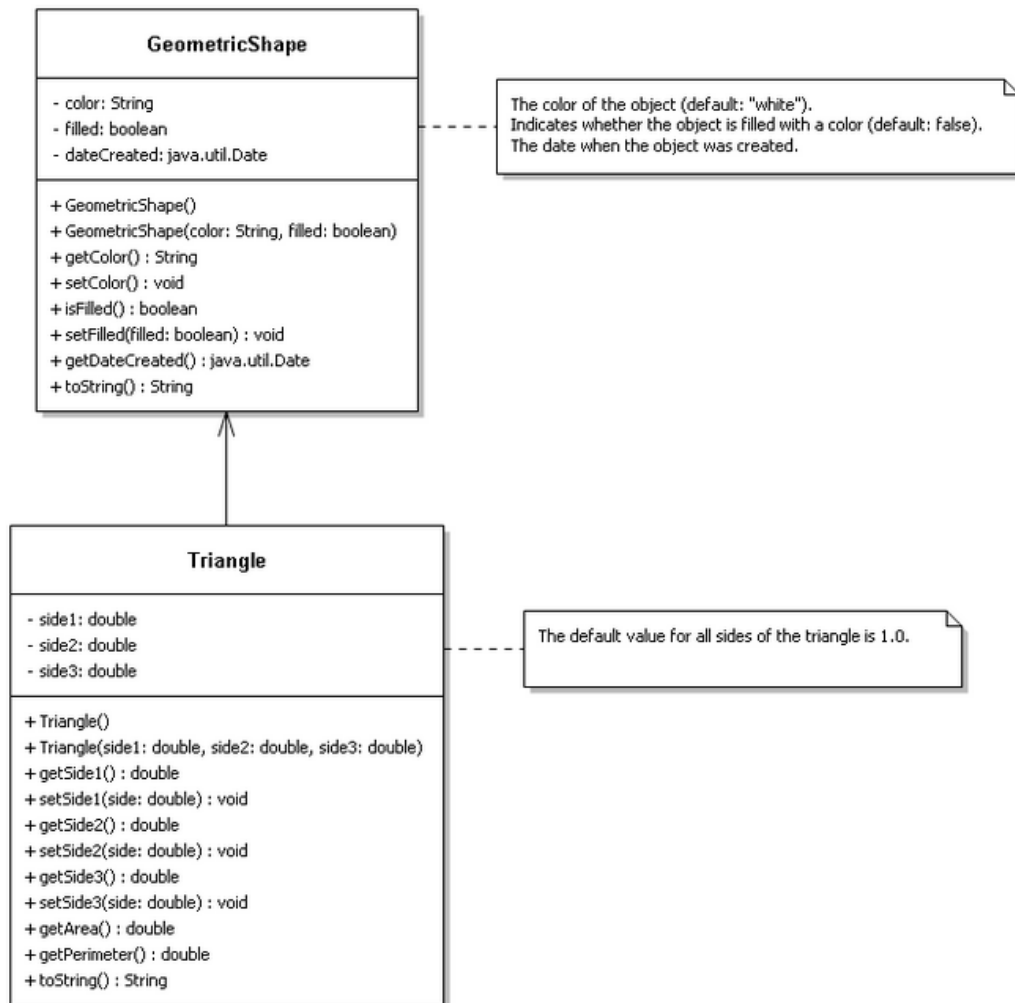
A háromszög területének a kiszámítására használja a Hérón-képletet! A toString() metódust implementálja a következőképpen:

```

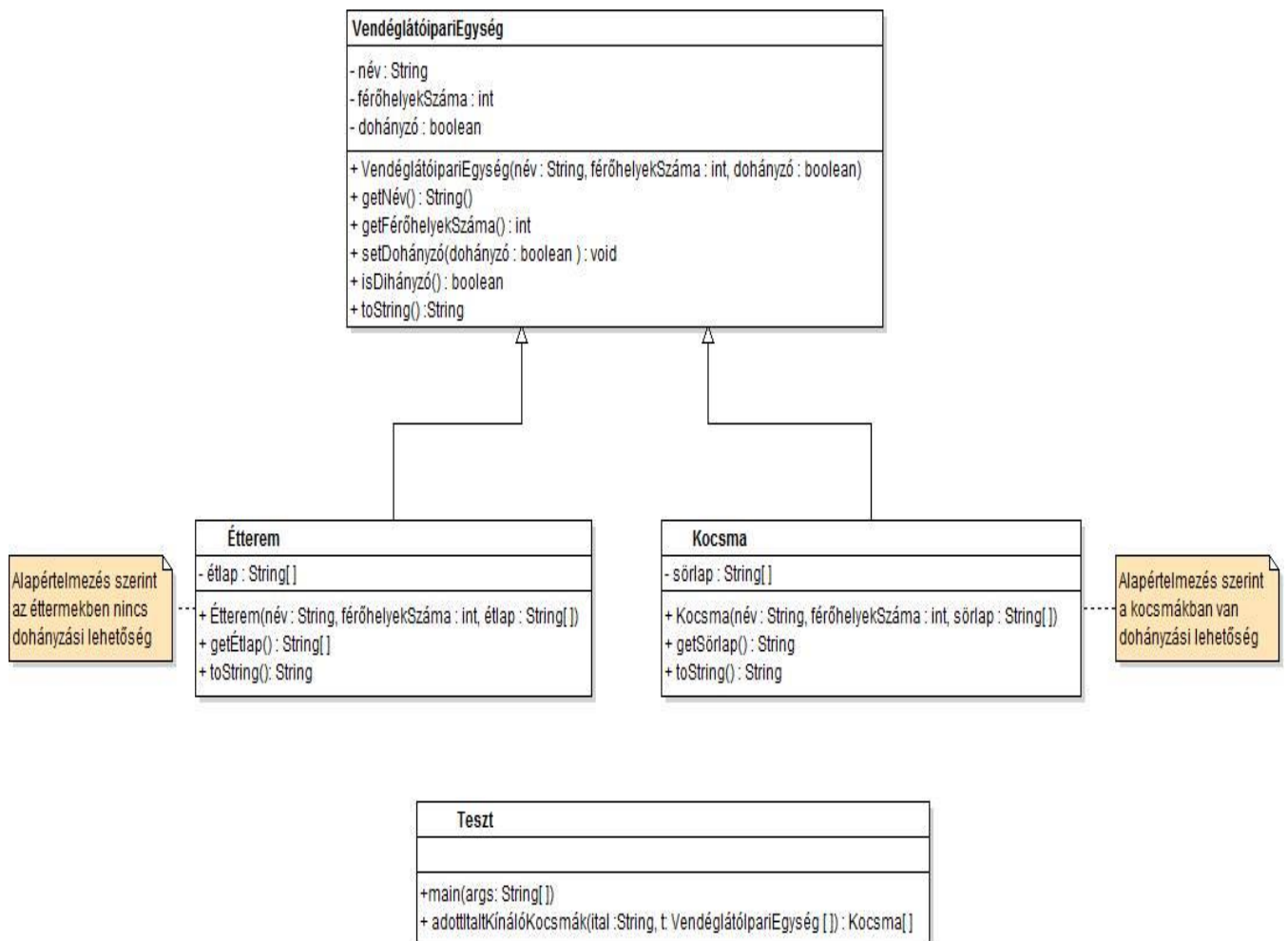
return "Triangle: side1 = " + side1 + " side2 = " + side2 + " side3 = " + side3;

```

Implementálja az osztályt! Írjon egy teszt programot, amely létrehoz egy Triangle objektumot 1, 1.5, 1 hosszúságú oldalakkal, yellow színnel kitöltöttre állítja be, majd a standard kimenetre írja a területét, a kerületét, a színét és azt, hogy ki van-e töltve vagy sem!



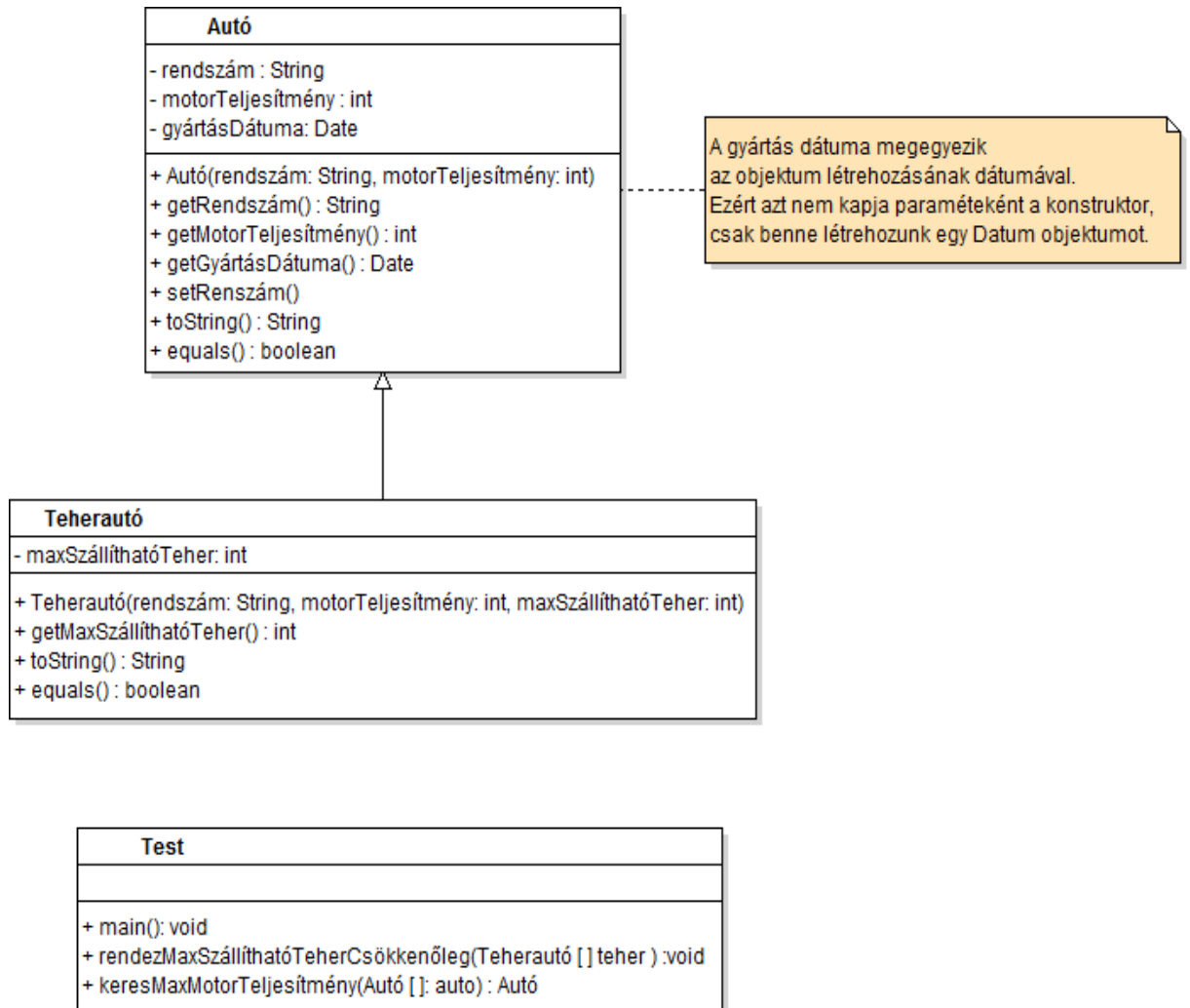
3. Implementálja az alábbi UML diagramon reprezentált osztályokat!



A main-ben hozzunk létre egy 10 elemű VendéglátóipariEgységeket tartalmazó tömböt, amiben 5 étterem és 5 kocsmá van. Az adottItaltKínálóKocsmák metódus függvény ezt kapja meg paraméterül, valamint egy ital nevét.

A keresés eredményül kapott Kocsmá tömb elemeit írassuk ki a standard kimenetre.

4. Implementálja az alábbi UML diagrammon reprezentált osztályokat!



A mainben hozzon létre egy 10 elemű Autó (ami 5 teherautót és 5 autót is tartalmaz!!!) és egy 10 elemű Teherautó tömböt. Ezeket adja át a fenti diagramon lévő függvényeknek. Az első rendezi a (Teherautó) tömböt a maximális szállítható teher szerint csökkenő sorrendben, míg a második megkeresi a legnagyobb motorteljesítménnyel rendelkező autót. Az eredményt minden esetben írassa ki.

5. Egészítse ki az alábbi kódban szereplő `insertionSort` statikus metódust oly módon, hogy az a paraméterként megkapott tömb elemeit beszűrő rendezéssel rendezze növekvő sorrendbe! A kód már megírt többi részét ne módosítsa!

```
import java.util.Random;

public class Test {

    public static void main(String[] args) {
        Random rnd = new Random();
        int[] t = new int[30];

        for (int i = 0; i < t.length; i++) {
            t[i] = rnd.nextInt(10);
        }

        printArray(t);

        long timeStart = System.currentTimeMillis();
        insertionSort(t);
        long timeEnd = System.currentTimeMillis();

        printArray(t);

        System.out.println(timeEnd - timeStart);
    }

    static void printArray(int[] a) {
        for (int i = 0; i < a.length; i++) {
            System.out.print(a[i] + " ");
        }
        System.out.println();
    }

    static void insertionSort(int[] a) {
        // Write your code here.
    }
}
```

6. (*Mintakeresés: egymást követő négy azonos értékű szám keresése.*) Írja meg a következő metódust, amely ellenőrzi, hogy a paraméterként megkapott tömbben van-e egymást követően négy azonos értékű szám! A metódus írásakor gondoljon arra, hogy nem biztos, hogy a tömbben van egyáltalán négy darab elem!

```
public static boolean isConsecutiveFour(int[] values)
```

Próbálja ki a `main` metódusban, a függvény működését.

7.

- a) Írj egy Ora osztályt, és helyezd az iskola csomagba! Egy óráról el kell tárolnunk a kódját (egész szám), a nevét és a kezdetét (ez egy egész szám 8 és 19 között, ugyanis minden újabb tanóra csak egész órakor kezdődhet, és pontosan 1 óra az időtartama). Írj 3 paraméteres konstruktort, amely megkapja az említett adatokat, és megfelelően inicializálja az adattagokat! Amennyiben az óra kezdetének időpontja nincs benne az elfogadható tartományban, akkor a közelebbi végpontra kell beállítani (tehát ha pl. 5-ös jön, akkor 8-ra). Írj getter és setter metódusokat, valamint toString() metódust az osztályhoz!
  
- b) Írj egy Orarend nevű osztályt, ami szintén az iskola csomagban van! Egy órarendről el kell tárolnunk, hogy melyik napnak az órarendje (hétfő/.../péntek), valamint van egy tömb adattagja is, mely az egyes Ora objektumokat tartalmazza, és egy számláló, melyben el van tárolva, hogy éppen hány órát tartalmaz az adott órarend. Írj 1 paraméteres konstruktort, mellyel beállítható, hogy melyik nap órarendjét hozzuk létre, emellett 0-ra állítja az óraszámológót, és lefoglal egy 12 méretű tömböt az óráknak (maximum 12 óra lehet egy napon). Írj getter és setter metódusokat, valamint toString() metódust az osztályhoz!
  
- c) Írj egy oratHozzaad() metódust, ami paraméterként egy Ora objektumot vár, és megpróbálja azt hozzáadni az órarendhez. Ha az órarendben már van 12 óra, akkor nincs hely többnek, illetve ha már kezdődik egy óra abban az időpontban, amikor az új, akkor is sikertelen a hozzáadás. A metódus aszerint térjen vissza igaz vagy hamis értékkel, hogy sikerült-e az új órát hozzáadni az órarendhez.
  
- d) Írj egy futtatható osztályt (Main), mely a standard bemeneten kap adatokat egy órarendről! Először egy nap érkezik, majd az, hogy hány órát szeretnénk hozzáadni az adott napi órarendhez. Ezután megfelelő számú hármast kell beolvasni, melyek minden órához megadják a kódot, a nevet és a kezdő időpontot. Az így beolvasott órákat egy tömbben eltároljuk. Példányosítjuk az Orarend osztályt a beolvasott napnak megfelelően, és megpróbáljuk hozzáadni a tömbből az órákat az előzőleg megírt oratHozzaad() metódus segítségével. Írassuk konzolra, hogy az egyes hozzáadások sikeresek voltak-e vagy sem. Végül írjuk ki a konzolra az órarend adatait!



8. Írjon programot, amely egy kő-papír-olló játékot valósít meg. A program beolvassa a bemenetről egy stringet (használd: `sc.nextLine()`). Ha az nem „kő”, „papír”, „olló” vagy „vége”, bekér egy újabb stringet. Ha a bekért string „vége”, akkor befejezi a programot, ha a „kő”, „papír” vagy „olló”, akkor generál egy véletlen számot 0 és 2 között, amely rendre a számítógép kő, papír, olló jelének felel meg, majd kiírja a képernyőre, hogy ki nyert!

## II. feladatsor (ezen feladatsor feladatai 1 projektbe kerülnek, külön csomagokba)

1.

- a) Írj egy Katona nevű osztályt. Egy katonának tároljuk a támadó és védőerejét (egész számok). Ezek az adattagok legyenek **privát** láthatóságúak és legyenek lekérdezhetők és beállíthatók publikus metódusokon keresztül. Az osztály rendelkezzen egy olyan konstruktorral, mely beállítja a támadó és védőerejét az objektumnak 5,5-re, továbbá rendelkezzen egy olyan konstruktorral, mely a kapott paraméterek alapján állítja be ezeket. Definiáld felül az osztályban a `toString()` metódust úgy, hogy kiírja az egység támadó és védőerejét. pl.: "TE: [támadóerő], VE: [védőerő]". Továbbá definiálja felül az osztályban az `equals` metódust, amely az egyenlőséget a támadóerő és védőerő alapján vizsgálja.
- b) Írj egy Nyilas és egy Landzsas osztályt, amely származik a Katona osztályból! A Nyilas osztálynak legyen egy **privát** láthatóságú lőtáv (egész szám) adattagja, mely publikus metódusokon keresztül legyen lekérdezhető és módosítható! A Nyilas támadóerejének lekérdezésekor a visszaadott érték legyen a támadóerő és a lőtáv összege! Mindkét osztály használja az ősosztály konstruktorát, a nyilas konstruktora várjon egy lőtáv paramétert is és ez alapján állítsa be a lőtáv értéket is! Definiáld felül ezekben az osztályokban is a `toString()` metódust. pl.: "Nyilas: TE: [támadóerő], VE: [védőerő]" Itt a lőtáv már szerepel (a támadóerőhöz hozzáadva). Továbbá definiálja felül a Nyilas osztályban az `equals` metódust, amely az egyenlőséget a támadóerő, védőerő és lőtáv alapján vizsgálja.
- c) Írj egy Teszt nevű futtatható osztályt! Az osztálynak legyen egy statikus `megkuzd()` metódusa, amely két Katona objektumot vár paraméterként és a visszatérési értéke a győztes katona objektum. A `megkuzd()` metódusban kiszámoljuk mely érték nagyobb:
- az első katona támadóereje - a második védőereje, (ekkor az első katona nyer)

- vagy a második katona támadóereje - az első védőereje. (ekkor a második nyer)
- ha a két érték megegyezik, akkor az első katona nyer.

A metódus írja ki mely katonák küzdtek meg, és ki nyert.

A futtatható osztályban, a main metódusban állíts elő három katona objektumot (például két Nyilast és egy Lándzsást)! A létrehozott katonák segítségével valósíts meg egy csatát három katona részvételével: rendezz egy küzdelmet a megkuzd() metódus segítségével az első két katona között, majd a győztes és a harmadik katona között is! Írd ki mely katona maradt életben!

## 2. feladat

# Fejések 2

Az általános iskolai testnevelésórán a tornatanár nagyság szerint növekvő sorrendbe állítja a nebulókat. A sor elején áll a legalacsonyabb gyerek, a sor végén a legmagasabb. A sor elején álló gyerek kivételével a többiek megkér, hogy mindegyikük jegyezze meg az előtte álló társa nevét (a gyerekek nevei szerencsére különböznek egymástól), hogy legközelebb könnyedén be tudjanak állni a sorba. Miután a gyerekek szépen elrendeződtek a sorban, a tanár a sor elejétől kezdve felváltva hol egy piros, hol egy kék színű sapkát húz az egymás után sorakozó gyerekek fejére, az első gyerekekre piros sapkát húzva.

Írjon programot, amely a standard bemenet első sorából beolvasson egy egész számot ( $n$ -et), ami a tornasorban álló gyerekek számát határozza meg, a tornasor legelején álló gyerek nevét, valamint egy újabb nevet (ami akár meg is egyezhet az előzővel, a későbbi hivatkozás kedvéért jelöljük most  $X$ -szel)! Ezt a három adatot egy-egy pontosvessző választja el egymástól. A következő  $n - 1$  sorban a tornasorban álló gyerekek adatai találhatók az alábbi formában:

*név; előtte*

A program írja a standard kimenetre azt, hogy az  $X$  nevű gyerek hány darab piros sapkás és kék sapkás fejet lát maga előtt, ha a sor elején álló legalacsonyabb társa felé tekint, feltételezve, hogy a gyerekek különböző magasságúak, és mindenki számára biztosított a „jó kilátás” a sor eleje felé! A két adatot egyetlen szóközzel válassza el egymástól!

## Példa bemenet

```
1. 5;Péter;Barnabás
2. Gábor;József
3. Samu;Barnabás
4. József;Péter
5. Barnabás;Gábor
```

letöltés szöveges állományként

## A példa bemenethez tartozó kimenet

```
1. 2 1
```

letöltés szöveges állományként

### 3. feladat

## Budapest Kupa 1 (Java)

A Budapest Kupát a Teljesítménytúrázók Társasága írta ki. Elnyerésének feltétele, hogy a 2017-re meghirdetett 144 túrából legalább 10 túrán vegyen részt az ember, és igazoltan teljesítse is azt. Az ön feladata egy olyan program írása, amely az ebben a sorozatban szereplő túrák adatait dolgozza fel.

A standard bemenet első sorában a feldolgozandó túrák darabszáma szerepel ( $N$ ). A következő  $N$  sor egy-egy túra adatait tartalmazza a következő formában:

*év; hónap; nap; túra\_neve; résztáv<sub>1</sub>; résztáv<sub>2</sub>...*

Az *év*, a *hónap* és a *nap* az év egy napjának leírására szolgál: az *év* az évszámot, a *hónap* a hónap éven belüli sorszámát, a *nap* pedig a túra napját írja le az adott hónapon belül. Az *év*, *hónap*, *nap* adatok mindig egy valódi, létező dátumot jelölnek. A *túra\_neve* egy sztring, a *résztáv* pedig egy pozitív egész szám. Az említett adatokat pontosvessző karakterek választják el egymástól egy soron belül. A pontosvessző karakter nem szerepel egyik túra nevében sem.

A túrák teljesítéséhez végig kell haladni a szervezők által kijelölt útvonalon, mely mentén ellenőrzőpontokat kell érinteni. Az ellenőrzőpontok egymástól való távolságát írják le az egyes sorokban szereplő résztávok. Ha egy túrán például a rajtból elindulva két ellenőrzőpontot kell érintenie a túrázónak, akkor három darab *résztáv* értéket fog a sor tartalmazni: a rajtból az első ellenőrzőpontba, az elsőből a másodikba, valamint a másodikból a célba vezető út hosszát. Így a túra teljesítendő útvonalát a résztávok összege fogja megadni.

A programjának a standard kimenetre kell írnia a leghosszabb teljesítendő útvonallal rendelkező túrák dátumát és nevét. Az adatokat rendezze dátum szerint növekvő sorrendbe! Ha két ilyen túrát is ugyanazon a napon rendeznének meg, akkor ezeket a túrákat a nevük alapján állítsa lexikografikusan növekvő sorrendbe (ábécérendbe)! Feltételezheti, hogy a bemeneti adatok között nem fog szerepelni két azonos nevű túra. A rendezett adatokat a példa kimenetben látható formában írja a programja a standard kimenetre!

### Példa bemenet

```
1. 5
2. 2017;05;14;Budaorsi dolomitok;5;4;3;5;5
3. 2017;02;04;Kitaibel Pal emlektura;2;3;2;3
4. 2017;04;01;Baba;3;3;3;4;3;3;3
5. 2017;01;21;Toldi Miklos emlektura;6;4;6;4
6. 2017;04;01;Pipi;4;4;4
```

letöltés szöveges állományként

### A példa bemenethez tartozó kimenet

```
1. 2017;04;01;Baba
2. 2017;05;14;Budaorsi dolomitok
```

letöltés szöveges állományként

#### 4. feladat

## Hullámvasutak (Java)

Egy zsúfolt vidámparkban olykor több órát is sorban kell ahhoz állni, hogy felülhessünk egy-egy hullámvasútra. A parkok méretére és a kigyózó sorokra való tekintettel érdemes megtervezni, hogy milyen sorrendben ülünk fel a játékaikra. A PortAventura World Európa egyik legnagyobb vidámparkja, mely a Costa Braván, Salou és Tarragona között található. Klasszikus területe (a kontinens egyetlen Ferrari parkján kívül) több tematikus világot tartalmaz, amelyek mindegyikében található legalább egy hullámvasút.

Írjon programot, amely a standard bemenet első sorából beolvas egy egész számot ( $n$ -et), amely a további feldolgozandó sorok (és hullámvasutak) darabszámát adja meg! A következő  $n$  sor mindegyikének a felépítése a következő:

*hullámvasút\_neve; világ\_neve; legkisebb\_magasság; várakozási\_idő*

A *hullámvasút\_neve* és a *világ\_neve* a hullámvasút, valamint a világ nevét tartalmazó sztringek, a *legkisebb\_magasság* a felüléshez szükséges magasságot (cm-ben kifejezve), a *várakozási\_idő* pedig az aktuális várakozási idő hosszát (percben kifejezve) tartalmazza.

A programja rendezze a beolvasott hullámvasutak adatait a várakozási idő szerint növekvő sorrendbe! Ha több azonos várakozási idővel rendelkező hullámvasút lenne a listában, akkor őket a legkisebb magasság szerint rendezze csökkenő sorrendbe! Ha ezek alapján sem tudna különbséget tenni két hullámvasút között, akkor őket a nevük szerint rendezze lexikografikus sorrendbe! Feltételezheti, hogy nincs két egyforma nevű hullámvasút.

A hullámvasutak adatait (név, világ és várakozási idő) a példa kimenetben látható formában írja a standard kimenetre!

### Példa bemenet

```
1. 6
2. Furius baco;Polynesia;140;120
3. Shambhala;China;140;120
4. Dragon Khan;China;140;80
5. Stampida;Far West;120;20
6. Tami Tami;SesamoAventura;100;20
7. El Diablo;Mexico;140;30
```

letöltés szöveges állományként

### A példa bemenethez tartozó kimenet

```
1. Stampida (Far West): 20
2. Tami Tami (SesamoAventura): 20
3. El Diablo (Mexico): 30
4. Dragon Khan (China): 80
5. Furius baco (Polynesia): 120
6. Shambhala (China): 120
```

letöltés szöveges állományként

#### 5. feladat

## Izzasztó feladat (Java)

Írjon programot, amely a standard bemenet első sorából beolvas egy egész számot ( $n$ -et) és egy településnevet, amely nem tartalmaz szóköz karaktert! A bemenet következő  $n$  sora egy-egy település nevét és az adott településen mért hőmérsékleti adatot tartalmazza, az előbbit sztringként, az utóbbit egész számként. A két értéket minden sorban egy kettőspont karakter választja el egymástól.

A programja döntse el, hogy a bemenet első sorából beolvasott településnév szerepel-e a megadott felsorolásban, és ha igen, akkor írja a standard kimenetre mindazoknak a településeknek az adatait (nevét és hőmérsékletét) a példa kimenetben látható formában, amelyeknél a hőmérséklet értéke meghaladja az első sorból beolvasott településen mért hőmérsékletet! A kimeneten a hőmérsékleti értékek szerint csökkenő sorrendben jelenjenek meg a települések adatai! Amennyiben több településen is azonos hőmérsékletet mértek volna, akkor ezeket a településnevek lexikografikusan növekvő sorrendjében (ábécérendben) írja a standard kimenetre!

Amennyiben a bemenet első sorából beolvasott településnév nem szerepelne a felsorolásban, akkor a programja mindössze egy „Missing data” tartalmú sort írjon a standard kimenetre! Ne felejtse el ebben az esetben sem soremelés karakterrel zárni a sort!

### 1. példa bemenet

```
1. 6 Debrecen
2. Budapest:19
3. Debrecen:24
4. Esztergom:25
5. Miskolc:26
6. Szeged:20
7. Szolnok:25
```

letöltés szöveges állományként

### Az 1. példa bemenethez tartozó kimenet

```
1. Miskolc (26)
2. Esztergom (25)
3. Szolnok (25)
```

## 6. feladat

# Másfél millió lépés (Java)

Írjon programot, amely a standard bemenet első sorából beolvasson egy  $n$  pozitív egész számot, amely a további sorok számát adja meg! A következő  $n$  sorban turisták vándorlásainak leírásai szerepelnek a következő formában:

*turista\_neve;útvonalleíró\_sztring*

A fenti sorban pontosvessző karakter választja el egymástól a turista nevét az általa megtett útvonalat leíró sztringtől. Az útvonalleíró sztringben K, P, S és Z karakterek jelzik azokat a szakaszokat, amelyeket a turista az országos kék, illetve a helyi jelentőségű piros, sárga és zöld jelzéseken tett meg. A pont karakterek (.) ebben a sztringben a turistajelzés nélküli útszakaszokat jelölik.

Minden turista esetén számítsa ki, hogy hány szakaszt tett meg a turista az országos kék jelzésen, és hányat tett meg a helyi jelentőségű piros, sárga és zöld jelzéseken összesen! Rendezze a turisták adatait csökkenő sorrendbe aszerint, hogy melyikük tette meg a legtöbb szakaszt az országos kék jelzésen! Ha ez az érték több turista esetén is megegyezne, akkor őket a helyi jelentőségű egyéb színű jelzéseken megtett szakaszok száma szerint állítsa csökkenő sorba! Ha e szerint az érték szerint sem tudna dönteni két turista sorrendjéről, akkor őket állítsa nevük alapján lexikografikusan növekvő sorrendbe! A program írja az ily módon sorba rendezett turisták nevét a standard kimenetre, soronként egyet-egyet!

## Példa bemenet

```
1. 4
2. Teszt Elek;K.K.K.
3. Gipsz Jakab;.Z.PK..S.SKZ...P.Z..P.K.S
4. Vizi Palma;KSKPKZK
5. Bodon Odon;...K....K...K...
```

letöltés szöveges állományként

## A példa bemenethez tartozó kimenet

```
1. Vizi Palma
2. Gipsz Jakab
3. Bodon Odon
4. Teszt Elek
```

letöltés szöveges állományként

## 7.feladat

# Angol–magyar szótár 2

Az angolórákon minden alkalommal új szavakat tanulnak a diákok. Az új szavak darabszámát ( $N$ -et) a standard bemenet első sora tartalmazza. A következő  $N$  sor mindegyike az alábbi formában tartalmazza a megtanulandó angol szavakat és magyar megfelelőiket:

*angol\_szó: magyar\_szó*

Az *angol\_szó* és a *magyar\_szó* minden esetben tetszőleges, betű karaktereket tartalmazó sztring lehet, amely nem tartalmaz szóköz és kettőspont karaktereket. A kettőspont karakter kizárólag az angol és a magyar szó elválasztására szolgál.

Gyakran előfordul, hogy egy angol szónak több magyar megfelelője is van, illetve fordítva: hogy egy magyar szót többféle szóval is ki lehet fejezni angolul. Például a *take* angol szó magyarul *vesz*, *visz* és *fog* értelemben is használható, vagy a magyar *galamb* szót angolul *dove*-nak és *pigeon*-nak is mondhatjuk. Ha ilyen szavak szerepelnének a felsorolásban, akkor természetesen minden egyes jelentés külön-külön, önálló sorban szerepel.

Készítsen egy olyan programot, amely a standard bemenetről beolvassa a megtanulandó szópárokat! A program rendezze a szópárokat az angol szavak szerint lexikografikusan növekvő sorba (ábécérendbe)! Ha több szópárnál is azonos lenne az angol szó, akkor ezeket a szópárokat a magyar jelentésük szerint tegye lexikografikusan növekvő sorrendbe (ábécérendbe)! Az ily módon sorba rendezett szópárokat aztán a bemeneti formátumhoz hasonló alakban írja ki a standard kimenetre!

## Példa bemenet

```
1. 5
2. take:visz
3. dove:galamb
4. take:vesz
5. pigeon:galamb
6. take:fog
```

letöltés szöveges állományként

## A példa bemenethez tartozó kimenet

```
1. dove:galamb
2. pigeon:galamb
3. take:fog
4. take:vesz
5. take:visz
```

letöltés szöveges állományként

