November 11

Szoftverfejlesztés gyakorlat – 11. óra

2019

OOP – Interface

Feladatlap

Interface

Egyes nyelvekben megállapodás, hogy „I"-vel kezdődnek az interface osztályok nevei. Javaban nem, ugyanúgy mint az osztályokat, nagy betűvel kell kezdeni.

Az interfacek olyan speciális absztrakt osztályok, amiknek nincs adattagja és nincs egyetlen implementált metódusa sem.

Az egyes metódusoknak nincs törzse az interface-ben, csak a szignatúrájukat kell megadni, mint az absztrakt osztályok esetén.

- Nem kell kiírni, de minden metódus public és abstract minősítésű!
- A szükséges bemenő paramétereket meg kell adni!

Az implements kulcsszó fog arra utalni, hogy az adott osztály megvalósítja az adott interface-eket.

- A felhasznált interface-ek minden metódusát meg kell valósítani!

Továbbá https://www.tutorialspoint.com/java/java_interfaces.htm oldalon láthatunk néhány összetettebb példát is. Például kiemelnél sportok interface-t, amiből származtat Foci és Hoki interfaceket is.

1. Feladat – Geometria alakzatok

Készíts négyzeteket leíró **geometry.Square** osztályt

- Legyen egy rejtett valós adattagja, melyben eltároljuk a négyzet oldalainak hosszát.
- Legyen egy publikus konstruktora, mely megkapja az oldalhosszt, és letárolja.
- Legyen egy publikus getA() metódusa, mely visszaadja az oldalhosszt.
- Legyen egy toString() metódusa, mely a következőt adja vissza:
  Square[a=oldalhossz]

Készíts általános téglalapokat leíró **geometry.GeneralRectangle** osztályt

- Legyen két rejtett valós adattagja, melyekben eltároljuk a téglalap oldalainak hosszát.
- Legyen egy publikus konstruktora, mely megkapja az oldalhosszakat, és letárolja.
- Legyen egy publikus getA() és egy publikus getB() metódusa, mely visszaadja az oldalhosszakat.
- Legyen egy toString() metódusa, mely a következőt adja vissza:
  Rectangle[a=oldalhossz,b=oldalhossz]

Készíts téglalapokat leíró **geometry.Rectangle** absztrakt osztályt

- Legyen két absztrakt publikus metódusa, getA() és getB() metódusa, melyek visszaadják az oldalhosszakat.

```
public abstract class Rectangle {
    public abstract double getA();
    public abstract double getB();
}
```

- A **Square** és a **GeneralRectangle** származzon a **Rectangle** típusból. A négyzet esetében nyilván a getB() metódus ugyanazzal fog visszatérni, mint a getA().

Készíts általános rombuszt leíró **geometry.GeneralRhombus** osztályt

- Legyen két rejtett valós adattagja, melyekben eltároljuk a rombusz oldalhosszát, valamint a (kisebbik) szögét.
- Legyen egy publikus konstruktora, mely megkapja az oldalhosszt és a szöget, és letárolja. (A szög ellenőrzésével most ne foglalkozzunk.)
- Legyen egy publikus getA() és egy publikus getAlpha() metódusa, melyek visszaadják a megfelelő adatokat.
- Legyen egy toString() metódusa, mely a következőt adja vissza: Rhombus[a=oldalhossz,alpha=szög]

Készíts rombuszt leíró **geometry.Rhombus** absztrakt osztályt

- Legyen két absztrakt publikus metódusa, getA() és getAlpha() metódusa, melyek visszaadják az oldalhosszt és a szöget.

```
public abstract class Rhombus {
    public abstract double getA();
    public abstract double getAplha();
}
```

- A **Square** és a **GeneralRhombus** származzon a **Rhombus** típusból. A négyzet esetében nyilván a getAlpha() metódus mindig 90-nel tér vissza.
- Grrrr! A Java nem enged osztályok között többszörös öröklődést, a Square nem származhat egyszerre a Rectangle és a Rhombus osztályokból. Mit tehetünk? Legyen mindkettő interfész!

Interfészek

- Tegyük a két absztrakt osztályunkat interfésszé! Egy interfész lényegében egy olyan absztrakt osztály, melynek csak publikus absztrakt metódusai vannak (konstruktora és adattagja sem lehet). Interfészek között lehet többszörösen öröklődni!
- A class kulcsszó helyett használjuk a két érintett típus elején az interface kulcsszót. Az osztályok pedig innentől nem leszármazni fognak ezekből, hanem úgymond megvalósítják ezeket az interfészeket, ezt az implements kulcsszóval jelöljük:

```
public class Square implements Rectangle, Rhombus {
    // ...
}
```

- Mivel interfészben minden metódus publikus és absztrakt, ezt jelölnünk sem kell:

```
public interface Rectangle {
    double getA();
    double getB();
}
```

- De nem hiba, ahogy az interfész előtt is kint hagyhatjuk az abstract kulcsszót, jelentése interfészek esetén nincs.
- Interfészeket is lehet örököltetni egymásból. Interfészek közti öröklődésre ismét az extends kulcsszót használjuk (egy interfész is természetesen több másik interfészből származhat).

Valósítsuk meg a **geometry.Parallelogram** interfészt

- A következő metódusokkal rendelkezzen az interface:
  - double getA();
  - double getB();
  - double getAlpha();

- Mind a Rectangle, mind a Rhombus interfészünk származzon a paralelogrammából.

Öröklődés használata

Valósítsunk meg egy egyszerű láncolt lista típust! Az osztály legyen a **lists.GeometryList**, és definiáljuk a következőképp:

- Legyen egy value rejtett adattagja, mely az aktuális értéket tárolja, ennek típusa legyen Object, hogy tetszőleges típusú adatot eltárolhassunk benne.

- Legyen egy next rejtett adattagja, melynek típusa GeometryList. Ez lesz a láncolt listánk következő eleme.
- Legyen egy Object és egy GeometryList típusú paramétereket fogadó publikus konstruktora, mely beállítja az adattagokat.
- Legyen egy Object paramétert fogadó publikus konstruktora, mely beállítja az adattagokat, a next-et null-ra.
- Legyen egy getNext és get publikus metódusa, előbbi a következő elem, utóbbi az érték (value) lekérdezésére.
- Legyen egy set(Object) metódusa az érték módosítására.
- Legyen egy toString metódusa, mely visszaadja az aktuális érték String reprezentációját, majd ha a next nem null, hozzákonkatenálja egy vesszővel elválasztva a next String reprezentációját.

Legyen egy **Main főosztályunk**, melyben létrehozunk egy listát a következő módon:

```
GeometryList list = new GeometryList(new Square(5), new GeometryList(new
GeneralRectangle(10, 2), new GeometryList(new Square(20)))
);
```

Ezt ugye bejárhatjuk a következő módon:

```
for (GeometryList t = list; t != null; t = t.getNext()) {
    System.out.println(t.get());
}
```

Mi van, ha a lista elemeinek oldalhosszait is ki akarjuk írni? Hiszen csak négyzeteket pakoltunk bele! Akkor tudunk castolni!

```
Object o = t.get();
Parallelogram p = (Parallelogram) o;
System.out.println(p.getA() + " " + p.getB());
```

A listánkat most módosítsuk a kiíratás előtt:

```
list = new GeometryList(new Object(), list);
```

Ekkor a lista első eleme egy Object lesz. Mi van, ha most akarunk castolni paralelogrammára? Hogyan lehetne ezt biztonságosan?

```
Object o = t.get();
if (o instanceof Parallelogram) {
    Parallelogram p = (Parallelogram) o;
    System.out.println(p.getA() + " " + p.getB());
}
```

2. Feladat

Írj egy **Arlap** interfészt, ami egy mennyibeKerul() metódust tartalmaz, mely egy lebegőpontos értékkel (az áru árával) tér vissza. Legyen továbbá egy CSESZEKAVE konstansa, ami egy kávé árát adja meg, ennek értéke 180.

Írj egy **Peksutemeny** absztrakt osztályt, ami implementálja az **Arlap** interfészt.

- Az osztály a következő lebegőpontos adattagokkal rendelkezik: mennyiseg, alapar. Az alapár csak ebből az osztályból legyen látható, míg a mennyiség legyen látható a leszármazott osztályokban is (használd a lehető legszűkebb láthatóságot).
- Az osztály rendelkezzen paraméteres konstruktorral, ami beállítja az adattagok értékeit. Legyen ezen felül egy megkostol public láthatóságú absztrakt függvénye, ami nem tér vissza értékkel. Az osztály valósítsa meg az implementált interfész metódusát. Egy péksütemény értéke az alapárának és a mennyiségének szorzatából számolható ki.
- Az osztály legyen továbbá szöveges formára alakítható, kiírva az adattagok értékét. (pl. "X db - Y Ft", ahol Y azt jelenti, mennyibe kerül összesen).

Írj egy **Pogacsa** osztályt, ami a Peksutemeny leszármazottja.

- Az osztálynak egy két paraméteres konstruktora legyen, ami a mennyiséget és az alapárat kéri el, majd állítja be.
- Pogácsa megkóstolásakor mindig csökkenjen felére a mennyisége.
- Az osztály legyen továbbá szöveges formára alakítható. Az objektum tulajdonságain kívül írja ki azt is, hogy Pogacsa osztályról van szó. (pl. "Pogacsa X db - Y Ft", ahol Y azt jelenti, mennyibe kerül összesen). Itt használd fel az ősosztály toString metódusát is!

Írj egy **Kave** osztályt, ami implementálja az Arlap interfészt.

- Az osztály egy habosE private láthatóságú logikai adattaggal rendelkezik.
- Az osztály rendelkezzen egy 1 paraméteres konstruktorral, ami beállítja az adattag értékét. A metódusai az alábbiak szerint legyenek megvalósítva: Egy rendes kávé ára annyi, amennyi az Arlap interfészben adott, habos kávé esetén ennek 1,5-szerese.
- Az osztály legyen szöveges formára alakítható. (pl. "Habos/Nem habos kave - X Ft").

• Írj egy **Pekseg** nevű, futtatható osztályt.

- Az osztálynak legyen egy tárolója (tetszőleges kollekció, pl. lista), ahova Arlap típusú objektumokat tárol.

- Az osztály rendelkezzen egy vasarlok statikus metódussal, ami egy fájl elérési útját várja paraméterül, visszatérése pedig void. A metódus feladata, hogy a fájlból beolvasott sorokat feldolgozza, és létrehozzon belőlük Pogacsa, vagy Kave objektumpéldányokat. A fájl egy sorában az adott objektum tulajdonságai szerepelnek. A létrehozott objektumpéldányokat közös tárolóban tárold le.
- Készíts továbbá egy etelLeltar statikus metódust, ami végigmegy a tárolóban tárolt elemeken, és az összes Pogacsa típus objektum információit kiírja egy "leltar.txt" fájlba.
- Hívd meg a main függvényben sorban a fenti két metódust. A vasarlok metódus paraméterét parancssori argumentumból kérd be.
- Minden esetleges kivételt (főleg: IOException, vagy a bemenet beolvasásakor/konvertálásakor előforduló kivételek) kezelj le vagy kivétel specifikációval, vagy try blokkban!

Egy minta fájl felépítése az alábbi:

Pogacsa 10 150
Kave habos
Kave nem_habos

3. Feladat

Írj **KisGepjarmu** interfészt, ami egy haladhatItt metódust deklarál. A metódus egy logikai értékkel térjen vissza (haladhat -e ezen az úton a gépjármű), és egy egész számot (sebességet) kérjen paraméterként.

Készíts egy absztrakt **Jarmu** osztályt.

- Egy járműnek legyen aktuális sebessége (int) és rendszáma. Az aktuális sebesség látszódjon a leszármazott osztályokban is (használd a lehető legszűkebb láthatóságot), míg a rendszám adattagot csak ebből az osztályból lehessen elérni. Írj konstruktort két paraméterrel, ami beállítja az adattagokat.
- Készíts egy gyorshajtottE absztrakt metódust, ami egy sebességkorlátot (int) kér paraméternek, és logikai értékkel tér vissza attól függően , hogy az adott jármű gyorshajtott -e.
- Készíts toString metódust, ami az alábbi módon alakítja szöveges formára az objektumot: "rendszam - X km/h" (ahol rendszam a jármű rendszáma, X pedig az aktuális sebessége)

Kézsíts egy **Robogo** osztályt, ami a Jarmu osztályból származik és implementálja a KisGepjarmu interfészt.

- A robogónak legyen egy maximális sebesség adattagja. A robogó konstruktora a rendszámot, az aktuális sebességet és a maximális sebességet kérje el paraméterül, és ez alapján hozza létre az objektumot.
- A gyorshajtottE metódus nézze meg, hogy a jármű aktuális sebessége fölötte van -e a paraméterként kapott korlátnak, és ennek megfelelően térjen vissza logikai értékkel.
- A haladhatItt metódust hamis értékkel térjen vissza, ha a robogó maximális sebessége nagyobb, mint a kapott paraméter, ellenkező esetben igaz legyen a visszatérés.
- Bővítsd ki az örökölt toString metódust, hogy az alábbiakat adja vissza: "Robogo:rendszam - X km/h" (ahol rendszam a jármű rendszáma, X pedig az akutális sebessége) Használd fel az ősosztály toString metódusát is!

Kézsíts egy **AudiS8** osztályt, ami a Jarmu osztályból származik.

- Az Audinak legyen egy lezerblokkolo (boolean) paramétere. Konstruktora a rendszámát, az aktuális sebességét kérje el, és hogy van -e lézerblokkolója, és ezek alapján hozza létre az objektumot.

- A gyorshajtottE metódus nézze meg, hogy a jármű aktuális sebessége fölötte van –e a paraméterként kapott korlátnak, és ennek megfelelően térjen vissza logikai értékkel. Ha a jármű rendelkezik lézerblokkolóval, úgy ehelyett mindig hamissal térjen vissza.
- Bővítsd ki az örökölt toString metódust, hogy az alábbiakat adja vissza: "Audi: rendszam - X km/h" (ahol rendszam a jármű rendszáma, X pedig az akutális sebessége). Használd fel az ősosztály toString metódusát is!

Készíts egy **Orszagut** futtatható osztályt.

- Az osztálynak legyen egy tárolója (tetszőleges kollekció, pl. lista), amiben Jarmu típusú objektumokat tárol.
- Legyen tovább egy statikus jarmuvekJonnek metódusa. Ez egy fájl elérési útját várja paraméterül. A metódus feladata, hogy a fájlból beolvasott sorokat feldolgozza, és létrehozzon belőlük Robogo és AudiS8 objektumpéldányokat, amiket hozzáad a tárolóhoz.
- Legyen egy statikus kiketMertunkBe metódus is, ami végigmegy a tárolón, és kiírja egy "buntetes.txt" fájlba a benne lévő járművek szöveges formában, és az Audi típusúakra pluszban kiírja azt is, hogy gyorshajtottak -e, míg a robogó típusúakra azt, hogy haladhatnak -e ezen az úton. Mindkét esetben 90 legyen a paraméter.
- A main metódusban hívd meg a jarmuvekJonnek metódust egy parancssori argumentumból bekért elérési úttal, majd hívd meg a kiketMertunkBe metódust is.
- Minden esetleges kivételt (főleg az IOException, de figyelj a bemenet feldolgozása közben tömbtúlindexelésekre és a számok átalakítása közben fellépő hibákra is) kezelj le vagy kivétel specifikációval, vagy try blokkban!

Egy minta fájl felépítése az alábbi: (típus;rendszám;aktuális sebesség;blokkoló/max. sebesseg)

robogo;"";40;60
audi;AAA-111;200;true
robogo;"";80;65
audi;AAA-111;130;false

4. Feladat – Film értékelés

Készítsen egy **IKorhatáros** nevű interfészt, ami az alábbiakat írja elő:

- Korhatár nevű, egész számot visszaadó, csak olvasható tulajdonságot
- Büntetés(életkor : szám) nevű, egész számot visszaadó metódust

Készítsen egy **Film** nevű osztályt az alábbiak szerint:

- Kívülről csak olvasható módon tárolja a film címét és árát
- Csak egy konstruktora legyen, ami beállítja ezeket az értékeket
- Rendelkezzen egy Értékel(érték : szám) metódussal, ami egy tömbben eltárolja a paraméterként kapott számot (1 és 5 közötti egész szám)
  Egy Film objektum maximum 10 értékelést tárolhat, ha ennél több érkezik, akkor dobjon egy saját ÚjÉrtékelésHiba kivételt, amely tartalmaz egy referenciát a kivételt dobó Film objektumra, illetve ki lehessen belőle olvasni az elmenteni nem sikerült értéket
- Legyen egy csak olvasható Átlag tulajdonsága, ami visszaadja az értékelések átlagát
  Amennyiben még nincs értékelés, akkor dobjon egy ÁtlagSzámításHiba kivételt, ami tartalmaz egy

referenciát a kivételt dobó Film objektumra
A két kivételnek célszerűen legyen egy közös őse a közös mezők és tulajdonságok kezelésére

Készítsen egy **AkcióFilm** nevű osztályt, amely a Film leszármazottja:

- Egészítse ki egy mezővel, ami tárolja a film alsó korhatárát
- Ez a mező a konstruktorban legyen beállítható
- Módosítsa úgy az Értékel metódust, hogy az 5-ös értékeléseket hagyja figyelmen kívül, csak az annál kisebbeket mentse el a Filmével azonos módon
- Valósítsa meg az IKorhatáros interfészt az alábbiak szerint:
  - A visszaadott korhatár legyen: a korhatár mező értéke
  - A büntetés mértéke legyen: (életkor paraméter – korhatár) * ár

Készítsen egy **Puzzle** osztályt az alábbiak szerint:

- Konstruktorban beállítható és kívülről csak olvasható módon tárolja a darabszámot
- Valósítsa meg az IKorhatáros interfészt az alábbiak szerint:
  - A visszaadott korhatár legyen: darabszám / 500
  - A büntetés mértéke legyen: (életkor paraméter – korhatár) * 1000

Készítse el az alábbi **főprogramot**:

- Hozzon létre egy tömböt, és töltse fel néhány fixen megadott Film és AkcióFilm objektummal
- Keresse meg a legolcsóbb filmet és írja ki annak adatait a képernyőre (kezelje az esetlegesen felmerülő kivételeket)
- Hozzon létre egy tömböt, amiben IKorhatáros interfészt megvalósító elemeket tárolhat. Másolja ide a filmekből az AkcióFilmeket, és rakjon bele fixen megadott Puzzle objektumokat
- Ez alapján számolja meg, hogy egy megadott életkor esetén hány Filmet és Puzzle-t lehet büntetlenül megvásárolni, illetve mennyi lenne az összes büntetés, ha mindent megvennénk

5. Feladat

Készítsen egy **IJáték** nevű interfészt, ami az alábbiakat írja elő:

- Név nevű, szöveget visszaadó, csak olvasható tulajdonságot
- JátszhatVele(életkor : szám) nevű metódust

Készítsen egy **JátékLista** nevű, gyakoriság szerint rendezett láncolt lista osztályt az alábbiak szerint:

- Az osztály legyen generikus, a T generikus típus határozza meg, hogy miket lehet tárolni ebben a láncolt listában. Készítse el az ehhez szükséges JátékListaElem osztályt is.
- Készítsen megszorítást, hogy csak IJáték interfészt megvalósító objektumokat lehessen tárolni
- A JátékLista tartalmazzon egy ÉletkorKorlát egész szám típusú tulajdonságot, amely a konstruktorban kapjon kezdeti értéket.
- Az osztály rendelkezzen a megszokott láncolt lista metódusokkal:
  - ÚjJátékFelvétele Ellenőrizze először, hogy a paraméterként átadott objektum felvehető-e a listába (a JátszhatVele metódus meghívása a listában tárolt életkorral igaz választ ad), és ha igen, vegye fel a listába az elemet

    - JátékTörlése Törölje a paraméterként átadott objektumot a listából
    - JátékKeresés Adja vissza a paraméterként átadott nevű objektumot a listából
- A lista legyen gyakoriság szerint rendezett, emiatt minden új elem kerüljön a lista végére, illetve minden keresés után a keresett elem kerüljön a lista elejére.
- Ha megváltozik a lista saját ÉletkorKorlát mezője, akkor törölje az összes olyan elemet a láncból, amelyik nem felel meg az új korlátnak.
- Készítsen egy ÚjJátékFelvéve nevű eseménykezelőt, amelyhez csak olyan metódussal lehessen feliratkozni, aminek egyetlen paramétere egy IJátékot-ot megvalósító objektum (készítse el a szükséges delegáltat is).
- Módosítsa úgy az ÚjJátékFelvétele metódust, hogy (amennyiben a beszúrni kívánt objektum életkora megfelelő), az hívja meg a fenti eseménykezelőt. Az eseményre feliratkozott metódusoknak paraméterként adja át a beszúrandó objektumot.

Készítsen egy SajátKészítésűJáték nevű osztályt, amely megvalósítja az IJáték interfészt

- Egy mezőben tárolja a játék nevét (ezt adja vissza a Név tulajdonság)
- A JátszhatVele metódus mindig igaz értékkel térjen vissza

Készítsen egy VeszélyesJáték nevű osztályt, ami a SajátKészítésűJáték leszármazottja

- A JátszhatVele metódus csak 18 éven felüliek esetén adjon vissza igazat

Készítse el az alábbi főprogramot:

- Hozzon létre láncolt lista objektumot, iratkozzon fel az eseménykezelőkre
- A listákat töltse fel néhány minta objektummal (jó és rossz életkorral)
- Mutasson egy-egy példát törlésre és keresésre, iratkozzon le az eseménykezelőkről

6. Feladat

Készíts egy **IPenzugy** interface-t, melyben a Metódusok:

- getFizetes: visszaadja a fizetést
- setFizetes: beállítja a fizetést

Készítsünk egy **IFeladat** interface-t, melyben a Metódusok:

- getFeladat: visszaadja a feladatot (String típusban írjuk le majd a feladatokat.)
- setFeladat: beállítja a feladatot

Készíts egy **Alkalmazott** osztályt, amely megvalósítja mindkét interface-t!

- Adattagjai nev, beosztas, feladat és fizetes legyenek.
- Készíts egy konstruktort, amely beállítja a nevet és a beosztást.
- Készíts az interface-ekből adódóan getFizetés, setFizetes, getFeladat, setFeladat neveken metódusok.
- Készíts továbbá egy toString metódust is, amely a nevet és a beosztast formázott módon jeleníti meg.

Írj egy **Vallakozas** nevű programosztályt.

- Készíts benne a főnök számára egy-egy metódust, amelyben lekérdezheti vagy előírhatja egy alkalmazott további feladatait, de a fizetéséhez nem férhet hozzá. Ehhez használd fel az interface-t.
  pl.: public void addFonokFeladat(IFeladat alkalmazott, String feladat){}

- A főnök az IFeladat interface-en keresztül fér az alkalmazotthoz, ezért az alkalmazottnak csak a feladat adattagját módosíthatja. Ha megpróbálná főnökként lekérdezni a fizetést, akkor hibaüzenetet kapna a fordítótól. Az interface-ek használatával tehát nemcsak előírhatja bizonyos metódusok kötelező megvalósítását, hanem szabályozhatja egy osztályon belül az egyes metódusok elérhetőségét is!
- Készíts egy-egy metódust a pénzügyes számára is, aki lekérdezheti, és előírhatja egy alkalmazott fizetését, de a feladatához nem férhet hozzá!

7. Feladat

Írj interfészt VanNeme néven, ami két függvényt tartalmaz: boolean him() és boolean no().

A him() függvény adjon vissza igazat, ha az implementáló osztály aktuális példánya hímnemű, a no() akkor, ha nőnemű.

Írj osztályokat Ember, Csiga és Amoba néven. Az Ember és a Csiga osztályok implementálják a VanNeme interfészt, az Amoba ne. A Csiga osztály him() és no() függvényei mindig igazat adjanak vissza, az Ember-é a konstruktor paraméterének függvényében csak az egyik esetben. Mindhárom osztály definiálja felül a toString metódust.

Írj egy főprogramot, ami egy fájlból szóközökkel elválasztott "no", "ferfi", "csiga" és "amoba" sztringeket olvas be és sorban elhelyez nekik megfelelő objektumokat (a fentebb meghatározott osztályok példányait) egy ArrayList-ben.

Három véletlenszerűen kiválasztott elemet vegyen ki a listából és írja ki a konzolra (a String reprezentációját). Végül írja ki, hogy ezek közül mennyinek volt neme.

8. Feladat

A bevezetőben említett sportok interface és annak leszármazott foci és hoki interfaceit valósítsd meg, úgy hogy kitalálsz hozzá egy feladat megvalósítást. A feladathoz kapcsolódóan néhány kritérium: Fájlból kell beolvasni az adatokat és használj benne 3-4 db programozási tételt.

Házi Feladat – Szuperhős

Írj egy **Szuperhos** interfészt, ami egy legyoziE metódust tartalmaz. A metódus paramétere egy Szuperhos, és egy logikai értékkel tér vissza. Legyen egy mekkoraAzEreje metódusa is, ami nem kér paramétert, és a Szuperhos erejét fogja visszaadni.

Írj **Milliardos** interfészt, ami egy visszatérés nélküli kutyutKeszit() metódust tartalmaz

Írj egy **Bosszuallo** absztrakt osztályt, ami implementálja a Szuperhos interfészt.

- Az osztály a következő private láthatóságú adattagokkal rendelkezik: egy lebegőpontos szuperero, és egy logikai vanEGyengesege.
- Az osztály rendelkezzen paraméteres konstruktorral, ami beállítja az adattagokat.

- Legyen egy public megmentiAVilagot absztrakt metódusa, ami egy logikai értékkel tér vissza. Valósítsd meg továbbá az interfész metódusait. Az erő lekérdezésekor add vissza a Bosszuallo szupererejét. Egy Bosszuallo egy Bosszuallo szuperhőst akkor tud legyőzni, ha annak van gyengesége, és ereje kisebb, mint az övé. Batman-t csak akkor tudja legyőzni, ha ereje kétszer nagyobb.
- Az osztálynak legyen továbbá getter és setter metódusa az adattagjaihoz, és legyen szöveges formára alakítható, kiírva az adattagok értékét.

Írj egy **Vasember** osztályt, ami a Bosszuallo leszármazottja, és megvalósítja a Milliardos interfészt.

- Az osztálynak egy default konstruktora legyen, ami beállítja a Vasember tulajdonságait. A Vasember szuperereje 150, és van gyengesége.
- Ha a Vasember kütyüt készít, akkor szuperereje nőjön ereje egy 0-10 közötti véletlenszerű lebegőpontos számmal.
- A Vasember akkor menti meg a világot, ha a szuperereje nagyobb, mint 1000.
- Az osztály legyen továbbá szöveges formára alakítható. Az adattagok értékein kívül írja ki azt is, hogy a Vasemberről van szó.

Írj egy **Batman** osztályt, ami implementálja a Szuperhos és Milliardos interfészeket.

- Az osztálynak legyen egy lebegőpontos lelemenyesseg adattagja.
- Az osztály rendelkezzen egy default konstruktorral, ami 100-ra állítja az adattag értékét. A metódusai az alábbiak szerint legyenek megvalósítva: Batman ereje a leleményességének kétszeresével egyezik meg, és bármilyen Szuperhőst képes legyőzni, akinek ereje kisebb, mint Batman leleményessége. Ha Batman kütyüt készít, akkor a leleményessége 50-el nő.
- Az osztály legyen szöveges formára alakítható, ami kiírja, hogy Batmanről van szó, és megadja a leleményességét.

Írj egy **Kepregeny** nevű futtatható osztályt. Az osztály rendelkezzen egy szereplok statikus függvénnyel, ami egy fájl elérési útját várja paraméterül, visszatérése pedig void. A metódus feladata, hogy a fájlból beolvasott sorokat feldolgozza, és létrehozzon belőlük Batman, vagy Vasember objektumpéldányokat, majd ezekre meghívja a kutyutKeszit metódust annyiszor, ahányszor az aktuális sor írja. Ezeket egy közös kollekcióban tárold le.

Készíts továbbá egy szuperhosok statikus metódust, ami végigmegy a tárolóban tárolt szuperhősökön, és kiírja őket.

Hívd meg a main függvényben sorban a fenti két metódust. Minden esetleges kivételt (főleg: IOException) kezelj le vagy kivétel specifikációval, vagy try blokkban!

Egy minta fájl felépítése az alábbi:

Vasember 5
Batman 8

Házi feladat - Repülőgépek

Készíts két interface-t: IPassanger, IFreighter nevekkel. Az első az utasszállítókra jellemző, ami azt tudja, hogy visszaadja az utasok számát, a második a csomagszállító repülőkre jellemző és visszaadja a csomagok súlyát.

Hozz létre egy Repulogep ősosztály, amelynek van egy azonosito és tankMeret adattagja. Származtass a Repulogep ősosztályból különböző típusú repülőgép osztályokat. UtasSzallitoRepulo, CsomagSzallitoRepulo és egy KatonaiRepulogep osztályokat, amelyek csak IPassanger vagy IFreighter-t implementálják illetve az utolsó mindkettőt. Írd meg értelmesen ezeket az osztályokat (utasszállítóban adattag az utasok száma, csomagszállítóban a csomagok súlya).

Hozz létre egy Repulogepek osztályt ami eltárolja egy reptéren éppen várakozó repülőket egy Listába. Töltsd fel konkrét repülőgép típusokkal a listát, úgy hogy minden típusú repülő szerepeljen legalább egy-egy példánnyal.

Keresd meg a legnagyobb tankkal rendelkező repülőgépet!