Asztali alkalmazások fejlesztése – Java – 7. óra

October 13

2021

Absztrakt osztály, Final kucslszó, Statikus osztály, Interface

Feladatlap

Objektumorientált tervezés

A tervezés során a valós világ elemeire vonatkozó feladat objektummodelljét állítjuk elő. Az egyes elemeket objektumokkal modellezzük. Ezt a folyamatot gyakran megszemélyesítésnek vagy antropomorf tervezésnek nevezzük.

Általánosságban az Objektum Orientált Programozás tervezés lépései a következőek:

- A feladat pontos specifikálása
- A feladat elvégzéséhez szükséges objektumok meghatározása
- Az objektumok tulajdonságainak (adattagok), tevékenységeinek (metódusok) felmérése.
- A közös tulajdonságok és tevékenységek kiemelése. (Öröklődés, Polimorfizmus)
- Az osztályhierarchia kialakítása általánosítással és specializációval
- Az osztályok/objektumok kapcsolatának, együttműködésének kiépítése.

A feladat megoldásához szükséges objektumok azonosításában segítenek a specifikációban szereplő főnevek, illetve a tevékenységeket pedig igék segítségével a legegyszerűbb meghatározni.

Absztrakt osztály

- Az Objektum Orientált fejlesztés során olyan osztályokat is kialakíthatunk, melyeket csak továbbfejlesztésre, származtatásra lehet használni, vele objektumpéldány nem készíthető, azonban objektumreferencia igen.
- Az osztály fejlécében az abstract kulcsszóval hozunk létre abstract osztályt.
- Az abstract osztályok további jellegzetessége, hogy bizonyos műveletek, amelyek szükségesek az osztály működéséhez, általában nincsenek kidolgozva – a függvény dekralációt pontosvessző zárja és nincs törzsük. Az ilyen metódusoknál is alkalmazni kell az abstract kulcsszót.
- Ilyenkor az abstract metódusok implementációját a származtatott osztályban kell megtenni. Amennyiben ezt nem tesszük meg, akkor a származtatott osztálynak is abstract-nak kell lennie!
- Ha egy osztályban van abstract metódus, akkor azt az osztály fejrésznél is jelezni kell.
- Abstract osztálynak nem kötelező minden függvényének, hogy abstract legyen. Abstract metódus nem lehet private és final!

A final kulcsszó

A final kulcsszó több vonatkozásban is használatos módosító, s jelentése az, hogy amit módosít az bizonyos értelemben nem változtatható meg.

final osztályok

A fordító nem engedi meg a final deklarációjú osztályokból szubosztályok létrehozását.

final metódusok

Egy metódus is deklarálható mint final. A final-ként delkarált metódusok nem írhatók felül a szubosztályban. A szintaxis egyszerű, a hozzáférési jog megadása után kell írnia a final kulcsszót, de a visszatérési típus előtt. Például: public final String penznemAtvaltas()

final mezők

A mezők is deklarálhatók mint final mezők. Ez azonban nem azonos az osztályok vagy metódusok final deklarációjával. Ha egy mező final, akkor az nem fog és nem tud megváltozni. Egyszer beállítható (például amikor az objektumot létrehozzuk, de ezután már nem változtatható). Változtatási kísérlet vagy fordítási hibát vagy kivételes állapotot generál.

Azok a mezők, melyek egyidejűleg final, static és public deklarációjúak, lényegében konstatnsok. Például egy fizikai program a fény sebességét az alábbiak szerint definiálhatja:

```
public class Fizika {

  public static final double c = 2.998E8;

}
```

final argumentumok

Végül egy metódus argumentumai is deklarálhatók mint final argumentumok. Ez azt jelenti, hogy a metódus nem fogja közvetlenül megváltoztatni azokat. Mivel az argumentumok érték szerint adódnak át, erre nincs is szükség.

Statikus adattagok

A statikus adattag az osztály valamennyi objektuma számára egy példányban létezik, azok osztottan használják.

A statikus adattag definíciója minden tagfüggvények kívül helyezkedhet el. Célszerű helye az osztály implementációs file-ja. Inicializálása nem történhet a konstruktorban, csak a definíciójában.

Osztály esetén azt jelenti a static, hogy egyrészt csak statikus tagjai vannak, másrészt statikus osztályból nem hozható létre objektumpéldány, hanem az osztályon keresztül lehet elérni az egyes adattagokat, metódusokat.

Tehát anélkül lehet hozzáférni, hogy egy objektumpéldányt kellene létrehozni egy osztályból.

pl. Az Autók osztály statikus, a Motorok osztály pedig nem.

```
class Program
{
    static void Main(string[] args)
    {
        Autok.Tipus = "Opel";

        Motorok motor = new Motorok();
        motor.Tipus = "Yamaha";
    }
}

static class Autok
{
    public static string Tipus;
}

class Motorok
{
```

```
        public string Tipus;
    }
```

Interface

Az interfészek jelentik az absztrakció következő szintjét. Egy interfész olyan osztálynak fogható fel, amelynek csak abstract metódusai és final, static mezői vannak. Az interfész összes metódusa és mezője public.

Az osztályokkal azonban ellentétben, egy interfészt hozzá lehet adni egy olyan osztályhoz is, amely már egy másik szubosztálya. Továbbá egy interfész sok különböző típusú osztályhoz is hozzáadható. Például definiálhatunk egy Import interfészt egyetlen metódussal, melynek neve calculateTariff().

```
public interface Import {

  public double calculateTariff();

}
```

Valószínűleg használni fogja tudni ezt az interfészt az autó osztályok esetén, de nem csak ott, hanem a ruhák, élelmiszerek, elektronika, stb. esetén is. Ésszerűtlen lenne mindezeket az osztályokat egyetlen közös osztályból származtatni. Továbbá az egyes tételeknél valószínűleg eltérő módon kell számolni a vámot. Ezért definiálunk egy Import interfészt, s deklaráljuk, hogy minden osztály implementálja az Import-ot.

A szintaxis egyszerű. Mivel az Import public deklarációjú, ezért minden osztályból hozzáférhető. Protected deklaráció is használható, akkor csak az azonos csomag osztályai férhetnek hozzá. Ez azonban ritka, csaknem minden interfész public. Nyilvánvalóan a private deklarációnak nincs értelme.

Az interface kulcsszó a class kulcsszó szerepét veszi át. A metódusok deklarálva vannak, de nincs törzsük. Ezeket az implementáló osztályoknak kell megadni.

Sok különböző metódust deklarálhat az interfészben. Ezek a metódusok overloaded típusúak is lehetnek.

Egy interfésznek mezője is lehet, de annak final és static típusúnak kell lenni (azaz konstansnak).

Absztrakt Osztály

1. Feladat – SíkidomokOOP

1. Feladat

Készíts egy SikidomokOOP projektet!

Hozz létre egy absztrakt Sikidom osztályt.

```
public abstract class Sikidom {
```

```
    protected abstract double keruletSzamit();
    protected abstract double teruletSzamit();
}
```

Az alábbi osztálydefiníció az alábbi tervezési meggondolásokat fejezi ki:

- minden síkidomnak van területe és kerülete,
- nincs algoritmus a terület és kerület számítására a síkidom tulajdonságainak ismerete nélkül.

Absztrakt metódus :

- Nincs törzse, és a végét „ ; ” zárja
- Kifejteni, majd csak származtatás után fogjuk
- Nem lehet módosítója a private, final, static mivel későbbiekben felül akarjuk őket definiálni

Absztrakt osztály :

- Ha van legalább egy absztrakt metódusa
- Nem lehet példányosítani
- Gyereke is absztrakt (!) ha nem valósítjuk meg az összes definiált metódust

Hozz létre egy Téglalap, Kör, és Háromszög osztályt a síkidomok osztályból, ahol kell hozz létre a megfelelő adattagokat, ha szükséges akkor a getter és setter metódusokat, és implementáld a kerulet es terulet örökölt metódusokat.

```
public class Teglalap extends Sikidom {
    private double a, b;

    public Teglalap (double a, double b) {
        this.a = a;
        this.b = b;
    }
    public double a () {
        return a;
    }
    public double b () {
        return b;
    }
    protected double keruletSzamit () {
        return 2*(a+b);
    }
    protected double teruletSzamit () {
        return a*b;
    }
}
```

Bővítsd a Haromszog osztályt egy paraméter nélküli konstruktorral, amely véletlenszerűen inicializálja az objektumpéldányt!

Teszteld az osztályokat.

2. Feladat

Származtass a Sikidom absztrakt osztályból egy paralelogramma osztályt!

Az osztály tartalmazzon (az örökölt elemeken túl)

- Egy privát, valós típusú adatmezőt (alfa) a két oldal által bezárt szög tárolására, a szöget fokokban tároljuk
- Egy paraméter nélküli konstruktort, amely véletlenszerűen inicializálja az objektumpéldányt.
- Egy paraméteres konstruktort.
- get, illetve set metódust az alfa privát adatmezőhöz
- A kerulet(), terulet() metódusok implementációját paralelogramma esetén.
    - A területet a sinusos területképlettel számold!
    - Ne feledkezz meg arról, hogy a szöget radiánba át kell számolnod!
  - Teszteld az osztályt!

3. Feladat

Saját ötlet alapján származtass egy újabb osztályt a Sikidom absztrakt osztályból!
Valósítsd meg az első feladatban leírt részleteket ebben az osztályban is!

4. Feladat

Definiálj egy Sikidomok osztályt! Az osztály tartalmazzon:

- Egy privát adatmezőt, amely a síkidomok tárolására alkalmas lista legyen.
- Egy konstruktort, amely
- véletlenszerű síkidomokkal töltse fel tömböt, vagy listát.
- A következő metódusokat:
    - . toString(): stringben adja vissza az objektum teljes tartalmát
    - . osszKerulet(): adja vissza az objektumban tárolt síkidomok összkerületét
    - . osszTerulet(): adja vissza az objektumban tárolt síkidomok összterületét
    - . maxTerulet(): adja vissza a legnagyobb területű síkidom sorszámát

2. Feladat – Házi állatok

1. Készítsen egy `Pet` nevű absztrakt osztályt!
    - az osztálynak legyen egy `name` nevű, szöveg típusú, valamint egy `numberofLegs` nevű, egész típusú adattagja! A láthatóságot a tanultaknak megfelelően kezelje!
    - az osztálynak legyen egy konstruktora, ami beállítja az adattagokat!
    - hozza létre a `Description` nevű, paraméterek nélküli, felülírható metódust! A metódus írja ki, hogy „`nem sokat tudunk erről az állatról`”!
    - hozza létre a `HowManyLegs` nevű, paraméterek nélküli metódust, ami vissza adja az állat lábát!
    - hozza létre a `Voice` nevű, paraméterek nélküli, absztrakt metódust!
2. Készítsen egy `Dog` nevű osztályt, ami a `Pet` osztály leszármazottja legyen!
    - az osztálynak legyen egy `breed` nevű, szöveg típusú adattagja, a kutya fajtájának tárolására!
    - az osztály konstruktora az ősosztályban elvégzett műveleteken felül állítsa be a `breed` adattagot is!
    - írja felül a `Description` metódust, és írja ki, hogy „`Ez egy {breed} fajtájú kutya, a neve {name}`”!
    - implementálja a `Voice` metódust, és írja ki szövegesen, hogy milyen hangot ad ki a kutya!
3. Készítsen egy `Fish` nevű osztályt, a `Dog` osztály mintájára!
4. Készítsen egy tetszőleges osztályt, amit a `Pet` osztályból származtat! Értelemszerűen vegyen fel új adattagokat, implementálja a metódusokat!

5. A `Main.java` fájlban hozzon létre egy listát, ami különféle háziállatokat képes tárolni! Töltse fel pár állattal, mindenféle fajtából! Írja ki a lista elemeit, leírásukkal, lábaik számával együtt!

3. Feladat – Mesterember

Készíts **Mesterember** absztrakt osztályt, mely tartalmaz:

- egy *nev* nevű string adatmezőt
- egy *napiDij* nevű adatmezőt
- egy *foglaltNapok* nevű 31 bool elemből álló tömb adatmezőt, mely tömb adott eleme azt mutatja meg, hogy az adott mesterember a hónap adott napján foglalt-e (true=foglalt, false=szabad)
- egy konstruktort, mely a fenti két adatot (név, napidíj) paraméterként kapja meg. (Kezdetben minden nap szabad a mesterember)
- egy *toString()* nevű metódust, mely az objektum valamennyi adatát adja vissza egy karaktersorozatként
- valamint egy *MunkatVallal ()* nevű bool visszatérési értékkel, és egész típusú paraméterrel rendelkező metódust. A metódus nem tartalmaz megvalósítást, ezért az legyen abstrakt!

Készíts **Burkoló** osztályt, mely a Mesterember osztály leszármazottja és tartalmaz:

- egy *szakterulet* nevű adatmezőt, amelynek a típusa egy *Helyszin* felsorolás (enum), melynek értéke csak „Belső”, vagy „Külső” lehet.
- egy konstruktort, mely az összes szükséges adatot paraméterként kapja meg
- egy o*sszesSzabadnap()* nevű metódust, mely az adott Burkoló szabadnapjainak számát adja vissza
- egy felüldefiniált *toString()* nevű metódust, mely az adott objektum valamennyi adatát adja vissza egy karaktersorozatként!
- egy *MunkatVallal()* nevű metódust. A metódus paramétereként kapott értéke a hónap adott számú napját tartalmazza. Ellenőrizze le, hogy a Burkoló szabad még azon a napon, ha igen foglalja le és a visszatérési érték legyen igaz. Ha nem szabad, akkor a visszatérési érték legyen hamis.

Készíts **VízvezetékSzerelő** osztályt, mely az Mesterember osztály leszármazottja és tartalmaz:

- egy egész típusú adatmezőt, mely azt tartja nyilván, hogy hány év *tapasztalat*tal rendelkezik.
- egy konstruktort, mely az összes szükséges adatot paraméterként kapja meg
- egy felüldefiniált *toString()* nevű metódust, mely az adott objektum valamennyi adatát adja vissza egy karaktersorozatként!
- egy *MunkátVállal()* nevű metódust. A metódus paramétereként kapott értéke a hónap adott számú napját tartalmazza. A VízvezetékSzerelőnek 3 napra van szüksége, hogy a feladatát elvégezze. Egy napra a megadott érték előtt és egy napra a megadott érték után. Ellenőrizze le, hogy a VízvezetékSzerelő szabad még a szükséges napokon, ha igen foglalja le, s a visszatérési érték legyen igaz. Ha nem szabad, akkor a visszatérési érték legyen hamis. Ügyeljen a hónap elejére és végére.

Készíts **Szakember** osztályt, amely a fenti három osztály működését mutatja be a következő módon:

- készítsen egy 6 elemből álló listát, mely Mesterember típusú objektumok tárolására képes.
- Töltsd fel a következő elemekkel:

objektum típusa	név	napidíj	szakterület	tapasztalat

Burkoló	Tapéta Lajos	60.000	Belső	-
VízvezetékSzerelő	Megszer Elek	12.000	-	3
Burkoló	Vakolat Péter	50.000	Külső	-
VízvezetékSzerelő	Víz Elek	15.000	-	5
Burkoló	Eresz János	30.000	Külső	-

- A 6. elemet a felhasználó adja meg.
- Az alkalmazás kérdezze meg a felhasználót, hogy mely típusú mesteremberek adataira kíváncsi, azok minden adatát írassa ki.
- Szimuláljon pár megrendelést a burkolók számára!
- Majd írassa ki a legtöbb szabadnappal rendelkező Burkoló minden adatát.

4. Feladat – Árverés szimulátor

A feladatmegoldás során fontos betartani az elnevezésekre és típusokra vonatkozó megszorításokat, illetve a szövegek formázási szabályait. Segédfüggvények is létrehozhatók, a feladatban nem megkötött adattagok és elnevezéseik is a feladat megoldójára vannak bízva. Törekedni kell arra, hogy az osztályok belső reprezentációja a lehető legjobban legyen védve, tehát csak akkor és csak olyan hozzáférés megengedett, amelyre a feladat felszólít, vagy amit az osztályt használó kódrészlet megkíván!

A megoldásnak működnie kell a mellékelt tester package-n belüli tesztprogrammal. A megírt forráskód kellően általános és újrafelhasználható legyen!

auction.Lot osztály: Az osztály egy árverési tételt (műalkotást) reprezentál.

- Az osztálynak három rejtett adattagja van: egy szöveg típusú alkotó (artist), egy szöveg típusú cím (title) és egy egész típusú leütési ár (hammerPrice).
- Az osztálynak legyen egy rejtett konstruktora (private), amely paraméterként megkapja az alkotó nevét, a műalkotás címét, valamint a kikiáltási árat, és beállítja a megfelelő adattagokat (a leütési ár legyen a kikiáltási ár (upSetPrice)).
- Definiáljunk egy osztályszintű (static) make nevű metódust is. A make metódus szintén az alkotó nevét, a műalkotás címét és a kikiáltási árát kapja meg paraméterként. A metódus először ellenőrzi, hogy a paraméterek megfelelőek. Amennyiben igen, akkor létrehozza és visszaadja a paramétereknek megfelelő Lot típusú objektumot. Ha a paraméterek nem megfelelőek, akkor a metódus null-t adjon vissza.
    - Az alkotó neve akkor megfelelő, ha nem egy null referencia.
    - A műalkotás címe akkor megfelelő, ha szintén nem egy null referencia, és legalább 2 karakter hosszú, csak nagybetűkből és szóközökből áll. (írásjelekre is ügyelj!)
    - A kikiáltási ár akkor megfelelő, ha pozitív szám.

  Segítség: a metódusban használható a Character osztály isLetter() és isUpperCase() metódusa.
- Definiáljuk az osztályban az alábbi, paraméter nélküli lekérdező metódusokat: getArtist(), getTitle() és getHammerPrice(), amelyek rendre visszaadják a műalkotás címét és a leütési árat.
- Az osztálynak legyen egy bid nevű metódusa, mely visszatérési érték nélküli, és egy pozitív egész paramétert vár, és amelynek segítségével licitálni lehet az aktuális műalkotásra. A licit a következőképpen történik: ha a

paraméter nagyobb, mint műalkotás leütési ára, akkor a leütési árat a paraméterrel tesszük egyenlővé. Különben nem történik semmi.

- Definiáljunk egy paraméter nélküli toString nevű metódust is, amely visszaadja az objektum szöveges reprezentációját. A formátum legyen a következő: alkotó: műalkotás címe (leütési ár GBP). Pl. Henri Matisse: JACQUY (350000 GBP), Salvador Dali: PORTRAIT DE MADAME DUCAS (500000 GBP).
- Definiáljunk egy moreExpensiveThan() metódust, mely egy műtárgyat vár paraméterül, és logikai igazat ad vissza, ha az aktuális műtárgy, melyen a metódust meghívták, drágább, mint a paraméterül kapott, továbbá a paraméter nem null.

auction.Auction osztály: Az osztály egy árverést reprezentál.

- Az osztály egy rejtett műtárgy-sorozat adattagban tartsa nyilván, hogy milyen műtárgyakra (Lot típusú objektumok) lehet licitálni. A típus tetszőleges, lehet rögzített méretű sorozat típus is.
- Az osztálynak legyen egy publikus konstruktora, amely műtárgyak tömbjét kapja paraméterként. A konstruktor inicializálja a sorozat adattagot a tömböt használva, ügyelve arra, hogy a belső állapot ne szivárogjon ki. Feltesszük, hogy egyik elem sem null.
- Definiáljunk egy numberOfLots nevű metódust, amely visszaadja az árverésen szereplő műtárgyak számát.
- Definiáljunk egy paraméter nélküli toString nevű metódust is, amely visszaadja az árverés szöveges reprezentációját. Az egyes alkotásokat sortörés vagy szóköz karakter is elválaszthatja. A szöveg összeállításakor a műtárgyak olyan formában szerepeljenek, ahogyan a Lot toString nevű metódusa előállítja őket. Az utolsó műtárgy után opcionálisan lehet sortörés vagy szóköz.

A **auction.Auction osztály**ban definiáljuk az alábbi publikus metódusokat:

- browseLots(): a metódus lehetővé teszi a műtárgyak közötti böngészést. Egy alkotó nevét kapja paraméterként és egy listában visszaadja azon műtárgyakat, melyek az adott alkotó művei. Ha az árverezőház nem rendelkezik egyetlen olyan műalkotással sem, mely megfelel a követelménynek, akkor a metódus egy üres listát ad vissza. A végeredmény típusa List legyen, megadva az elemek típusát is.
- priceOfCollection(): a metódus megadja, hogy mennyibe kerülne, ha egy adott alkotó összes művét szeretné megvenni egy rajongó. A metódus egy alkotó nevét várja paraméterül és egy long típusú számot ad vissza eredményül (egy gyűjtemény rengeteg pénzbe kerülhet).
- mostExpensive(): a metódus paraméter nélküli, és az árverezőház legdrágább műalkotását adja vissza (egy Lot típusú objektumot). Ha az árverezőháznak egyetlen műalkotása sincsen, akkor null-t adjunk vissza. Ha több egyformán legdrágább alkotás van, akkor az elsővel térjünk vissza.

Hozz létre néhány műtárgyat, majd teszteld a program helyes működését is.

```
Lot.make("Henri Matisse", "JACQUY", 350000)
Lot.make("Salvador Dali", "PORTRAIT DE MADAME DUCAS", 500000)
Lot.make("Batthiany Gyula", "FIATAL LANY VIRAGOS RUHABAN", 10000)
Lot.make("Schonberger Armand", "ABSZINTIVOK", 50000)
Lot.make("Salvador Dali", "THE CUBE", 75000)
Lot.make("Henri Matisse", "BLUE POT AND LEMON", 15000)
Lot.make("Salvador Dali", "SOLEIL", 25000)
Lot.make("Felix W. de Weldon", "US MARINE MEMORIAL", 55000)
```

5. Feladat – Kártyapakli

Készíts egy **Kartya** osztályt, ami játékkártyát valósít meg, és helyezd el a **zsuga** csomagba!

- Egy kártyának van számértéke (2-14 között), és színe (piros, vagy fekete). Az adattagok csak ebből az osztályból legyenek elérhetőek!
- Készíts 2 paraméteres konstruktort, ami beállítja a fenti értékeket. Ha nem megfelelő a számértéknek megadott paraméter, úgy a kártya értéke legyen 2!
- Készíts default konstruktort, ami egy 0 értékű piros kártyalapot hoz létre!
- Írj minden adattaghoz getter és setter függvényeket! Ha a számértéket nem megfelelő értékre akarják állítani, úgy azt hagyd figyelmen kívül.
- Egy kártyalap legyen szöveges formára alakítható az alábbi módon: "szin ertek" (pl. piros 8).

Írj **Pakli** osztályt, és helyezd a **zsuga** csomagba!

- Egy kártyapakli tömbben tárolja a benne lévő Kartya objektumokat. Legyen továbbá egy konstansa, ami a pakli joker színét tárolja. Az adattagok csak ebből az osztályból legyenek elérhetőek!
- Kártyapaklit lehessen létrehozni maximális lapszámának és joker színének megadásával, valamint default konstruktorral is. A konstruktor inicializálja a lapokat tároló tömböt a megfelelő lapszámmal, és állítsa a joker konstanst a megadott értékre. Alapesetben 52 lapos egy pakli, és a fekete a joker. A konstruktor töltse fel a paklit véletlenszerűen generált Kartya objektumokkal (az érték és a szín is legyen véletlenszerű). (6 pakli kártyát szoktak összekeverni a Black Jack játék esetén, amiből véletlenszerűn választanak ki 52 lapot)
- Készíts getter metódust, ami egy indexet vár, és a kártyákat tároló tömbből visszaadja az ott található kártyát! Ha az adott index nem megfelelő, úgy egy default kártyalappal térjen vissza.
- Készíts getter metódust, ami a maximális lapszámot adja vissza.
- Készíts a paklinak osszesLap metódust, ami végigmegy a benne tárolt kártyákon, és egyesével kiírja őket konzolra.
- A pakli legyen szöveges formára alakítható az alábbi módon: "X lapos kartyapakli fekete/piros joker szinnel"
- Készíts egy blackJackLight metódust, ami egy int paramétert vár. A paraméter megadja, hogy hány kártyát szeretnénk megnézni a pakli tetejéről, és a metódus visszaadja a kártyák értékének összegét. Egy kártyalap az értékének felét (felfele kerekítve) érik, ha a színe nem egyezik meg a pakli joker színével! Ha a paraméter nagyobb, mint a lapok aktuális száma, akkor csak annyi kártyát nézzünk meg, amennyit a tömb tárol.
- Overloading segítségével oldd meg, hogy a blackJackLight metódus Kartya típusú paraméterrel is hívható legyen. Ebben az esetben addig nézz meg kártyákat a pakli tetejéről, amíg el nem éred a paraméterként kapott lapot. Ha nincs ilyen lap a pakliban, akkor nézd végig mindegyiket. A megnézett kártyák értékének összegét az előző ponthoz hasonlóan állapítsd meg, és ezzel térj vissza.
- Készíts egy egyszinuPakli metódust, ami végigmegy a pakliban található kártyákon, és mindegyik színét a joker színre állítja.

Készíts egy **futtatható** osztályt a **futtat** csomagba. A main metódusban:

- Olvass be két egész számot parancssori argumentumból. Az első szám a pakli méretét, a második szám pedig a megnézendő lapok számát adja meg. Ellenőrizd, hogy kaptál-e két paramétert.
- Hozz létre egy Pakli objektumot az első paraméter segítségével. Ha a paraméter nem pozitív szám, úgy a default konstruktort használd!
- Írasd ki a pakli összes kártyáját a megfelelő metódus segítségével.

– Játssz egy blackJackLight játékot a paklival (hívd meg a megfelelő metódust) a második paraméterrel, és írasd ki az eredményt.

Házi feladat - Könnyű

Írj osztályt Bor néven. Egy bor objektum attribútumai a következők: fajta, evjarat. Az attribútumok legyenek privat láthatóságuak.

Írj konstruktort a Bor osztálynak, amely a paraméterei alapján inicializálja az attribútumokat. Az osztálynak ne legyen default konstruktora. Írj publikus függvényeket, amelyekkel lekérdezhetők es módosíthatok az attribútumok. Írj toString metódust az osztálynak.

Írj osztályt Aszu néven, amely származik a Bor osztályból. Az örökölteken felül egy aszu objektum a puttony attribútummal rendelkezik.

Írj konstruktort az Aszu osztálynak, amely 2 paramétert var: puttony es evjarat. Ezeknek a paramétereknek megfelelően inicializálja az attribútumokat. A fajta attribútum aszu objektumok esetében mindig "aszu" legyen. Írj publikus függvényeket, amelyekkel lekérdezhető és módosítható a puttony attribútum. Írj toString metódust az osztálynak.

Írj futtatható osztályt. Az osztálynak legyen egy statikus metódusa kiirBor névvel, amely egy Bor tipusú paramétert fogad. A paraméterül kapott objektumot írja ki a konzolra.

Hozzon létre Bor és Aszu objektumokat, majd írja ki őket a konzolra a kiirBor metódus segítségével.

A megírt osztályokat dokumentáld javadoc kommentekkel es készíts belőlük javadoc-kal dokumentációt. Erről részletesen a https://en.wikipedia.org/wiki/Javadoc oldalon találhat információt.

Házi feladat - Nehéz

Tervezd meg a mellékelt ital UML diagram alapján az előző feladat egy összetettebb megvalósítását is. Az interface új fogalom, nézz utána az interneten, próbáld megérteni a működési elvét.