2.3 Absztrakt, Interfész

A példánk egy állatokkal foglalkozó alkalmazás lesz, melyben szárazföldi, vízi; ragadozó és növényev állatok is lesznek.

> *Ehhez szükségünk lesz néhány osztályra, melyek írásáról, példányosításáról már korábban beszéltük, így ezeket az ismereteket **is mertnek** tekintjük. Akinek esetleg mégis problémája lenne az egyszer osztályok létrehozásával, az a 4. gyakorlat és az 5. gyakorlat anyagát ismételje át.*

Mivel állatokkal fogunk foglalkozni, így szükségünk lesz egy `Állat` osztályra. Egy állatnak legyen neve, jóllakottsága, illetve ereje.

```
public class Allat {
    private String nev;
    private int jollakottsag;
    private int ero;

    public Allat(String nev) {
        this.nev = nev;
        this.jollakottsag = 100;
        this.ero = 0;
    }

    //.. getterek és szetterek

    public String hangotAd(){
        return "";
    }
}
```

Absztrakt

Fent láthatjuk, hogy a `hangotAd()` metódusnak nincs gyakorlati haszna, csupán azért hoztuk létre, hogy a gyerekosztályokban majd felüldefiniálhassuk ket, így például egy `Állat` tömbben meg tudjuk az elemekre hívni a `hangotAd()` metódust.

Absztrakt metódusok

Ezzel már korábban is találkoztunk, és igazából elkerülhetjük az ilyen helyzeteket, hiszen a `hangotAd()` metódusnak egyedül annyi a lényege, hogy a gyerekosztályokban felül lehessen definiálni. Azonban ezzel jelenleg két probléma is van: az `Állat` osztályban van mködése, holott egy általánosságba véve vett állatról nem tudjuk, milyen hangot ad; illetve jelen pillanatban nem kötelez a gyerekosztályban felüldefiniálni, hiszen az örökölt metódusokat vagy felüldefiniáljuk, vagy nem, semmi sem kötelez rá, st, akár el is felejthetjük, ha nem vagyunk figyelmesek.

A probléma megoldására az `abstract` kulcsszó szolgál. Ezt odabiggyeszthetjük a metódusaink elé, cserébe nem kell ket implementálni az adott osztályban. Az `abstract` kulcsszóval azt mondjuk meg, hogy az adott osztályban egy (vagy több) adott metódust nem szeretnénk implementálni, csak azt szeretnénk megmondani, hogy a gyerekosztályban majd ezt az adott metódus(oka)t felül kell definiálnunk.

Absztrakt osztályok

Viszont, ha egy metódusunk elé odabiggyesztettük az `abstract` kulcsszót, akkor az osztálynak is **kötelez** absztraktnak lennie. Ennek a kulcsszava szintén az `abstract`, amelyet az osztály deklarációjában is ki kell tenni.

Erre azért van szükség, mert ha egy osztályunk absztrakt, akkor az nem példányosítható, és ugye ha van egy olyan metódusunk, aminek nincs törzse, akkor azt nem nagyon kellene példányosítani. Éppen ezért, ha az osztályban van **legalább egy absztrakt metódus, akkor az egész osztálynak absztraktnak kell lennie**. Persze ez csak annyit jelent, hogy jelen formájában nem példányosítható, és majd a gyerekosztályban kell az absztrakt metódusokat megvalósítani (ellenkez esetben a gyerekosztálynak is absztraktnak kell lennie).

Természetesen ezen kívül az absztrakt osztályokban is lehetnek adattagok, st olyan metódusok is, melyek meg vannak valósítva. Írjuk át az `Állat` osztályt!

```java
public abstract class Allat {
    private String nev;
    private int jollakottsag;
    private int ero;

    public Allat(String nev) {
        this.nev = nev;
        this.jollakottsag = 100;
        this.ero = 0;
    }

    // getterek, szetterek

    public abstract String hangotAd();
}
```

Ezáltal elértük, hogy az osztályunkban ne legyen egy olyan metódus, aminek nem tudunk normális mködést biztosítani, helyette csak megmondjuk, hogy a gyerekosztályokban a `hangotAd()` metódust felül **kell** definiálni, ha olyan osztályt szeretnénk, amit tudunk példányosítani is. Ha absztrakt osztályt próbálunk meg példányosítani, fordítási hibát kapunk.

Megtehetjük azt is, hogy egy osztály örökldik egy absztrakt osztályból, de nem feltétlen implementálja az összes örökölt absztrakt metódust, ilyenkor az az osztály is absztrakt lesz. Csináljunk két gyerekosztályt: `SzárazföldiÁllat` és `VíziÁllat`, amely szárazföldi és vízben él állatok sosztályai lesznek:

```java
public abstract class SzarazfoldiAllat extends Allat{
    public SzarazfoldiAllat(String nev) {
        super(nev);
    }

    private int labakSzama;

    public int getLabakSzama() {
        return labakSzama;
    }

    public void setLabakSzama(int labakSzama) {
        this.labakSzama = labakSzama;
    }

}

public abstract class ViziAllat extends Allat {
    public ViziAllat(String nev) {
        super(nev);
    }

    @Override
    public String hangotAd() {
        return "nem hallható a víz alatt";
    }
}
```

Ezek után már létrehozhatunk konkrét állatokat, attól függen hogy vízben vagy szárazföldön élnek. Csináljunk még két sosztályt: `Növényev` és `Ragadozó`, melyek növényev és ragadozó állatok sei lesznek.

```java
public class Ragadozo {
    public void eszik(Allat kit){}
    public void pihenes(int mennyit){}
}

public class Novenyevo {
    public void eszik(){}
}
```

Ez így megint nem néz ki túlságosan jól. Ehelyett alkalmazhatjuk az elbb megismert trükköt, azaz absztrakttá tehetjük az osztály metódusait, és magát az osztályt is. Ez így rendben is lehet. Azonban ezáltal beleütközünk a Java egyik korlátába, a többszörös öröklds hiányába, hiszen egy `Kutya` nem lehet egyszerre `Állat` és `Ragadozó`. Az absztrakttá tétel mködképes konstrukció ugyan, de ebben az esetben nem biztos, hogy a legjobb.

Interfész

Ugyanis, ha egy osztályban nincsenek sem adattagok, sem pedig megvalósított metódusok, akkor gyakorlatilag csak egy interfészrl beszélünk, ami megmondja, hogy az t kiterjeszt osztály milyen metódusokat valósítson meg, ha szeretnénk tudni példányosítani. Viszont mi absztrakttá tettük. Javaban létezik az interfésznek is saját kulcsszava, amely az `interface`kulcsszó lesz. Ezáltal jelezzük a világ számára, hogy ebben az "osztályban" csak és kizárólag metódusdeklarációkat lehet találni, tehát biztosan nincs megvalósított metódus, sem pedig adattag. Erre szolgál tehát az `interface` kulcsszó.

```
public interface Novenyevo {
    public abstract void eszik();
}

public interface Ragadozo {
    public abstract void eszik(Allat kit);
    public abstract void pihenes(int mennyit);
}
```

Korábban szó volt róla, hogy többszörös örökldés nincs Javaban, azonban bármennyi interfészt implementálhat egy osztály, ezáltal egy medve egyszerre lehet `Állat` és `Ragadozó` is. St, azok az állatok akik mindenevk, egyszerre implementálhatják a `Ragadozó` és `Növényev`interfészt is.

Egy interfészben minden metódus implicit módon abstract, így ezt kiírni sem kell.

```
public interface Novenyevo {
    public void eszik();
}

public interface Ragadozo {
    public void eszik(Allat kit);
    public void pihenes(int mennyit);
}
```

Interfészeket nem örököltetünk (kivéve, ha egy interfész örökldik egy másik interfészbl), hanem implementálunk, azaz egy interfészben deklarált metódusokat megvalósítunk abban az osztályban, amely implementálja az interfészt.

```
public class Csirke implements Novenyevo {

    @Override
    public void eszik() {
        System.out.println(this.getNev() + " jóllakott magvakkal.");
        this.setJollakottsag(100);
    }
}
```

Ugyanakkor, ha egy interfészt nem szeretnénk teljesen implementálni, azt is megtehetjük. Viszont így lesznek olyan metódusaink, amelyeket nem implementálunk. Így az ilyen osztályokat szintén el kell látnunk `abstract` kulcsszóval, hiszen absztrakt osztályok lesznek (vagyis olyan osztályok, amelyben vannak olyan metódusok, amelyek absztraktak).

A Csirke osztály `SzárazföldiÁllat` is és `Növényev` is, így származhat a `SzárazföldiÁllat`osztályból, ugyanakkor implementálhatja a `Növ ényev` interfészt is.

```
public class Csirke extends SzarazfoldiAllat implements Novenyevo {

    public Csirke(String nev) {
        super(nev);
        setLabakSzama(2);
    }

    @Override
    public void eszik() {
        System.out.println(this.getNev() + " jóllakott magvakkal.");
        this.setJollakottsag(100);
    }

    @Override
    public String hangotAd() {
        return "kott kott";
    }
}
```

instanceof, getClass()

instanceof

Hozzunk létre egy `Csorda` osztályt, amelyben állatok vannak. Minden állat objektumnak van egy konkrét típusa, hiszen az `Állat` osztály nem példányosítható, viszont a memóriában ott vannak a tényleges, létrehozott állatok. Készítsünk továbbá egy `csordábaFogad()` metódust, ami egy állatot vár paraméterül, és abban az esetben, ha van még hely a csordában, kerüljön be az adott állat, ellenkez esetben pedig ne.

```
public class Csorda {
    private Allat[] allatok;
    private int csordaLetszam;
    private int jelenlegiLetszam;

    public Csorda(int csordaLetszam) {
        this.csordaLetszam = csordaLetszam;
        this.jelenlegiLetszam = 0;
        allatok = new Allat[csordaLetszam];
    }

    public boolean csordabaFogad(Allat ki) {
        if (jelenlegiLetszam < csordaLetszam) {
            allatok[jelenlegiLetszam] = ki;
            jelenlegiLetszam++;
            return true;
        }
        return false;
    }

    @Override
    public String toString() {
        String str = "A csordaban vannak:";
        for (int i = 0; i < jelenlegiLetszam; i++) {
            str += allatok[i].toString();
        }
        return str;
    }
```

Elfordulhat, hogy szükségünk van egy s típusú tömb (vagy bármi, később látni fogjuk) elemeinek konkrét típusára. Ez nem probléma, hiszen futásidben tudni fogjuk minden egyes tömbelemrl, hogy konkrétan milyen típusú, hiszen a memóriában ott vana tényleges objektum.

Ennek lekérdezésére az `instanceof` kulcsszó használható, melynek szintaxisa elsre különös lehet: `objektum instanceof Osztály`.

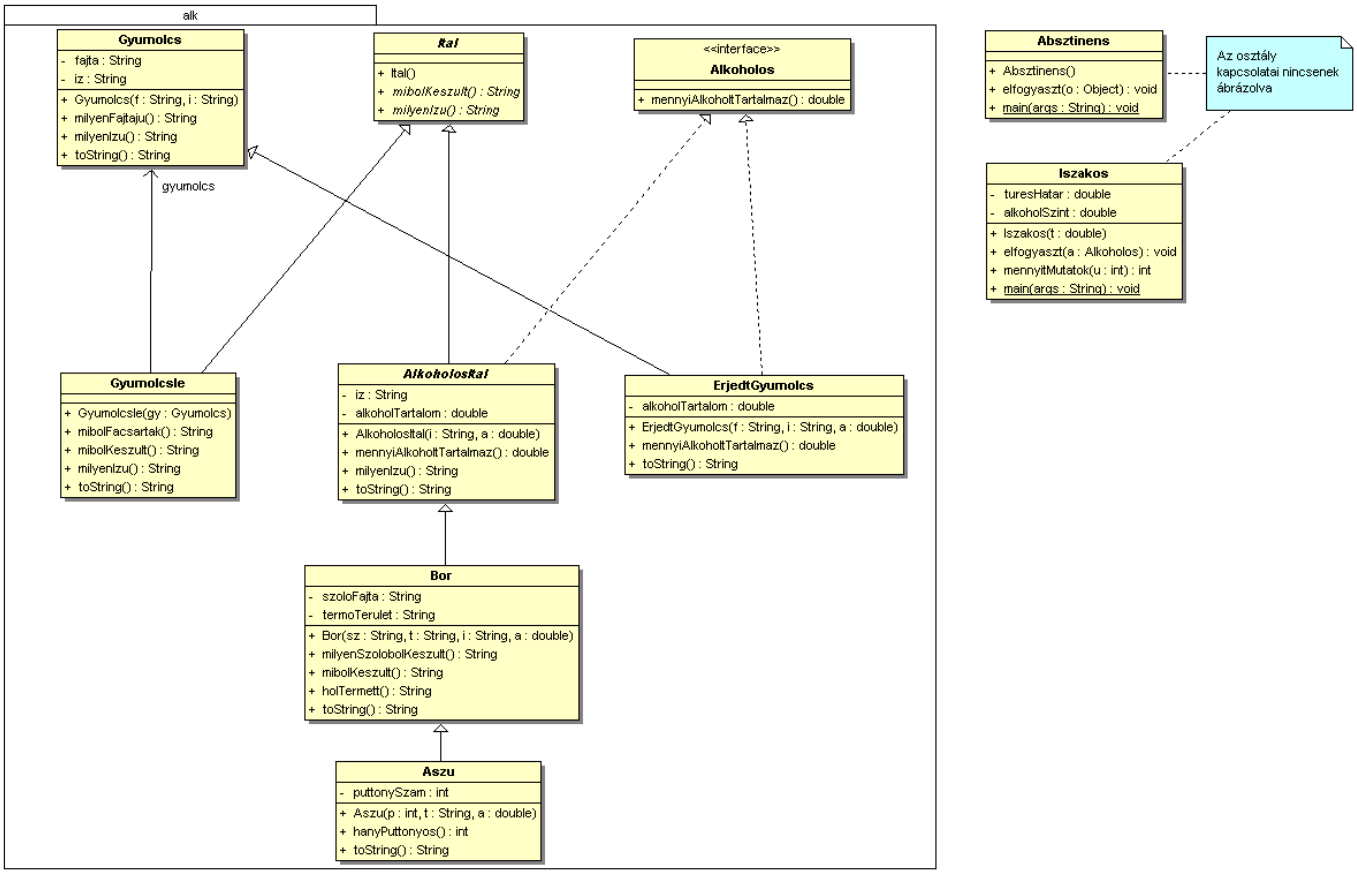
```
Allat pipi = new Csirke("Pipi");
boolean igazHogyCsirke = pipi instanceof Csirke; // true
```

Ez egy logikai kifejezés, tehát használható `if`-ben például, és visszaadja, hogy egy adott objektum konkrétan adott osztályból származik-e, vagy valamelyik sébl, tehát ha van egy `Állat` típusú, `állatok` nev tömb, akkor annak bármely elemére igazat adna vissza az `tömb[i] instanceof Állat` kifejezés.

getClass()

A `getClass()` egyike az `Object` nev sosztályból örökölt metódusainak (itt megtalálhatjuk az Object osztály javadocját), ami egy beépített típussal, `Class` típussal tér vissza, és ezt a metódust a gyerekekben felül sem definiálhatjuk, mivel egy `finaL` metódusról van szó. Ha nem objektumunk van, hanem osztályunk, annak is lekérhetjük a típusát, az `Osztály.class` utasítással. Egy objektumnak tehát lekérhetjük a tényleges osztályát az `objektum.getClass()` metódussal. Ez a metódus csak és kizárólag a tényleges osztállyal történ összehasonlítást engedélyezi, tehát ha van egy `Állat` típusú, `állatok` nev tömb, akkor annak bármely elemére hamisat adna vissza az `tömb[i].getClass() == Állat.class` kifejezés.

Italos példa



A fenti diagram forráskódja megtalálható a `/n/pub/Programozas-I/nappali/gyakorlat/07`útvonalon vagy pedig [ezen][italok_letolt] a linken.

Feladatok

Állatok kiegészítése

1. Hozz létre **három** tetszleges állatot az `allatok` csomagba, ezek között legyen szárazföldi, vízi, növényev, ragadozó. Implementáld az összes szükséges metódust!
2. Jelen állapotában a `Csorda` osztály bármilyen állatot tartalmazhat (például egy csordába tartozhat a cápa, az oroszlán és a csirke is). Ez a való életben nem biztos hogy megállja a helyét, így javítsd ki:
    a. Valósítsd meg, hogy egy csordában csak szárazföldi, vagy csak vízi állatok lehessenek. Mégpedig úgy, hogy az els állatot vizsgáljuk meg aki a csordába szeretne kerülni, és ha az szárazföldi állat, akkor a továbbiakban csak szárazföldi állatok kerülhessenek a csordába, ha vízi volt, csak vízi állatok.
    b. Valósítsd meg, hogy egy csordában csak növényev, vagy csak ragadozó állatok lehessenek. Mégpedig úgy, hogy az els állatot vizsgáljuk meg aki a csordába szeretne kerülni, és ha az növényev állat, akkor a továbbiakban csak növényev állatok kerülhessenek a csordába, ha ragadozó volt, csak ragadozó állatok.
    c. Valósítsd meg az elz két feladatot ömlesztve, azaz egy csordába csak azonos helyen él (szárazföldi, vízi), azonos életmódot folytató (növényev, ragadozó) állatok kerülhessenek.
    d. Valósítsd meg a fenti feladatokat öröklddéssel, melyek legyenek speciális `Csorda`osztályok.