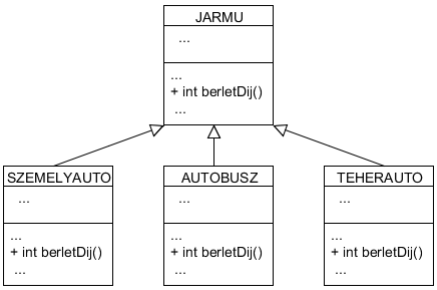
Programozás III

KOLLEKCIÓK

ISMÉTLÉS

A HF-EK APROPÓJÁN

Feladat:



Van egy számolt alapdíj, + egy típustól függő egyéb díj.

a/ hogyan számoljuk a bérletdíjakat?

b/ milyen legyen a toString(), ha ki akarjuk írni a bérletdíjat?

A HF-EK APROPÓJÁN

1. Az utódokba csak a módosítás kerül:
   return super.berletDij() + saját számítás.

NE a toString()-ben számoljon ki fontos adatokat, hiszen ez csak kiíratásra szolgál.

A berletDij() kiíratása kizárólag az ős toString() metódusában, a többiben nem, hiszen futáskor a futtató környezet tudja, hogy most épp melyik utód példányról van szó, és ezeknek megfelelően számolja a bérletdíjat.

A HF-EK APROPÓJÁN

Osztálynév kiíratása:
vagy this.getClass().getSimpleName().toLowerCase(),
vagy kell egy String getOsztalyNev() metódus.

Lehet-e két toString()?

Nem, de

a/ lehet mellette más String eredmenyString() metódus;

b/ lehet egy feltételtől függő elágazás a toString()-en belül.

A HF-EK APROPÓJÁN

Alaposztályból SOHA nem hivatkozunk a Global osztályra.
Miért?

Alaposztályból SOHA nincs input/output.
Miért?

Hogyan lehet egyedi sorszámot rendelni a példányokhoz?

EGYEDI SORSZÁM

Pl. Ember példányok egyedi sorszámozása

Ötletek: Az Ember osztályban vagy a konstruktorban, vagy set() metódussal.

Probléma: nem egyedi, hiszen a sorszám értéke a példányosítótól vagy a példány használójától függ.

(Ez persze nem jelenti azt, hogy ne adhatnánk kívülről generált, egymástól eltérő sorszámokat, csak ilyenkor lehetne egyformákat is adni.)

EGYEDI SORSZÁM

Megoldás: Az Ember osztályban bevezetünk egy minden emberre jellemző statikus változót – ez lesz az utolsó ember sorszáma. Az új ember a rákövetkező sorszámot kapja:

```
private int sorszam;
private static int utolsoSorszam;

public Ember( ...  ) {
        ...
    utolsoSorszam ++;
    this.sorszam = utolsoSorszam;
}
```

A HF-EK APROPÓJÁN

Tesztelés:

1. Lehetnek teszt célú kiíratások. Lehet, de helyette jobb a debug, pl:

```
27     public void rendel(Ital ital){
           int index = italok.indexOf(ital);
```

Berakhatunk break pointo(ka)t, ezeket figyelhetjük.

http://www.cs.uga.edu/~shoulami/sp2009/cs1301/tutorial/NetBeansDebuggerTutorial/NetBeansDebuggerTutorial.htm

A HF-EK APROPÓJÁN

Dátumkezelés:
A Java dátumkezelése elég nehézkes (volt a JDK_7-ig).
Ezért egyik megoldás:
Joda Time (dátumkezelő külső Java csomag)
http://www.joda.org/joda-time/
http://www.joda.org/joda-time/quickstart.html

De a JDK_8-ba átkerült a Joda Time, így már standard.
Csomag: java.time
http://javarevisited.blogspot.hu/2015/03/20-examples-of-date-and-time-api-from-Java8.html

KONTÉNEREK – GYŰJTEMÉNYEK (ismétlés)

A konténer olyan objektum, amely objektumokat tárol, és alkalmas különböző karbantartási, keresési és bejárási funkciók megvalósítására.

Csomagja: java.util

A konténerek általánosak, azokba bármilyen objektumot betehetünk. (De csak objektumot!)

A gyűjtemények „változtatható méretű tömbök”, rendelkeznek karbantartási és keresési funkciókkal.

FELADATMEGOLDÁS (ismétlés)

Gondoljuk végig egy klinika `latogato()` metódusát:

A betegek listája tartalmazza a regisztrált betegek adatait, keresünk egy konkrét beteget.

A programrészlet:

FELADATMEGOLDÁS (ismétlés)

```
private void latogato() {
    Paciens beteg;
    System.out.print("\nKi keresi? 1: orvos - 2: barát ");
    int ki = scan.nextInt();

    System.out.print("A keresett beteg neve: ");
    String nev = scan.next();
    if (ki == 2) {
        beteg = new Paciens(nev, "");
    } else {
        System.out.print("TAJ száma: ");
        String taj = scan.next();
        beteg = new Paciens(nev, taj);

    }
    if (betegek.contains(beteg)) {
        System.out.println("\nMegvan");
    } else {
        System.out.println("\nNincs ilyen beteg");
    }
}
```

FELADATMEGOLDÁS (ismétlés)

Biztos, hogy megtalálja?

Csak akkor, ha a Paciens osztályban van **equals()** és **hashcode()** metódus.

És ezek a metódusok honnan tudják, hogy orvos vagy barát keresi az illetőt?

Egyelőre sehonnan, fel kell rá készíteni őket.

FELADATMEGOLDÁS (ismétlés)

```
public class Paciens {
    private String nev, tajSzam;
    private static int hasonlitasiSzempont = 0;
    public static final int ORVOS = 1;
    public static final int BARAT = 2;

@Override
public int hashCode() {
    int hash = 7;
    hash = 97 * hash + Objects.hashCode(this.nev);
    if (hasonlitasiSzempont == ORVOS) {
        hash = 97 * hash + Objects.hashCode(this.tajSzam);
    }
    return hash;
}
```

FELADATMEGOLDÁS (ismétlés)

```
    public boolean equals(Object obj) {
        if (obj == null) {
            return false;
        }
//        if (getClass() != obj.getClass()) {
//            return false;
//        }
        final Paciens other = (Paciens) obj;
        if (!Objects.equals(this.nev, other.nev)) {
            return false;
        }
        if (hasonlitasiSzempont == ORVOS) {
            if (!Objects.equals(this.tajSzam, other.tajSzam)) {
                return false;
            }
        }
        return true;
    }
```

Mikor nem kell a kikommentezett rész?

```
    private void latogato() {
        Paciens beteg;
        System.out.print("\nKi keresi? 1: orvos - 2: barát ");
        int ki = scan.nextInt();

        System.out.print("A keresett beteg neve: ");
        String nev = scan.next();
        if (ki == 2) {
            beteg = new Paciens(nev, "");
            Paciens.setHasonlitasiSzempont(Paciens.BARAT);
        } else {
            System.out.print("TAJ száma: ");
            String taj = scan.next();
            beteg = new Paciens(nev, taj);
            Paciens.setHasonlitasiSzempont(Paciens.ORVOS);
        }
        if (betegek.contains(beteg)) {
            System.out.println("\nMegvan");
        } else {
            System.out.println("\nNincs ilyen beteg");
        }
    }
```

FELADATMEGOLDÁS (ismétlés)

Mi történik, ha a Paciens osztálynak van egy utód osztálya, és az utódok „ugyanolyanságára” még egy további saját jogú feltétel is van?

A név-re, tajszámra vonatkozó feltétel öröklődik a Paciens osztályból, a saját feltételre vonatkozó megkötést viszont az utód osztályban kell generálni.

RENDEZÉS (ismétlés)

A Collections osztály algoritmusai úgy működnek, hogy páronként összehasonlítják a gyűjtemény elemeit. Ezért ezek a metódusok megkövetelik, hogy **a konténerbe betett objektumok összehasonlíthatóak legyenek**, vagyis hogy

a/ maguk implementálják a Comparable interfészt, vagy

b/ létezzen hozzájuk a Comparator interfészt implementáló hasonlító osztály.

RENDEZÉS (ismétlés)

Mikor használható a Collections.sort(adatok) hívás?

adatok: List<Adat> adatok;

Adat: class Adat implements Comparable

Adat-ban: compareTo() metódus

RENDEZÉS (ismétlés)

Mikor használható a
Collections.sort(adatok, new Hasonlitas()) hívás?

adatok: List<Adat> adatok;

Adat: class Adat

Hasonlitas: class Hasonlitas implements Comparator

Hasonlitas-ban: compare() metódus

RENDEZÉS (ismétlés)

Rendezések:

Mindegy, hogy melyik fajtát használják, de egy projekten belül lehetőleg csak egyfajta legyen.

(Az AlapOsztaly implements Comparable megoldás esetén is lehet többféle szempontú rendezés – hogyan?)

RENDEZÉS (ismétlés)

Vannak italok és alkoholos italok, az alkoholos italoknak nyilván van alkoholfokuk. Az italok listája vegyesen tartalmaz sima és alkoholos italt (Ital ősosztály típusúak) Hogyan lehet rendezni őket alkoholfok szerint?

```
case ALKOHOLFOK:{
    double tfok = (t instanceof AlkoholosItal) ?
                        ((AlkoholosItal)t).getAlkoholFok() : 0 ;
    double t1fok = (t1 instanceof AlkoholosItal) ?
                        ((AlkoholosItal)t1).getAlkoholFok() : 0 ;
    return (int) ((miModon) ? Math.signum(tfok - t1fok) :
            Math.signum(t1fok - tfok));
}
```

MÉG EGY PÉLDA

Egy listában Sportolo típusú példányok vannak, konkrétan Ugro és Futo objektumok. Mindegyiknek van egy

int teljesitmeny(){…} értéke.

Ez futók esetén az időeredmény (sec), ugróknál az ugrott magasság (cm).

Hogyan rendezhetjük teljesítmény szerint a sportolókat?

Együtt sehogy, nem összemérhető adatok.

Maga az öröklődés is hibás: csak formailag hasonló az utódok teljesitmeny() metódusa, logikailag egyáltalán nem, így nem is szabadna örökíteni.

HA MÁR ISMÉTLÉS, AKKOR EZT IS

```
try (InputStream ins = this.getClass().getResourceAsStream(adatUt);
        Scanner fajlScanner = new Scanner(ins, CHAR_SET)) {
    String sor;
    String[] adatok;
    Ital ital;
    while (fajlScanner.hasNextLine()) {
        sor = fajlScanner.nextLine();
        adatok = sor.split(";");
        ital = new Ital(adatok[0], adatok[1],
                Integer.valueOf(adatok[2]));

        italok.add(ital);
    }
} catch (Exception ex) {

}
```

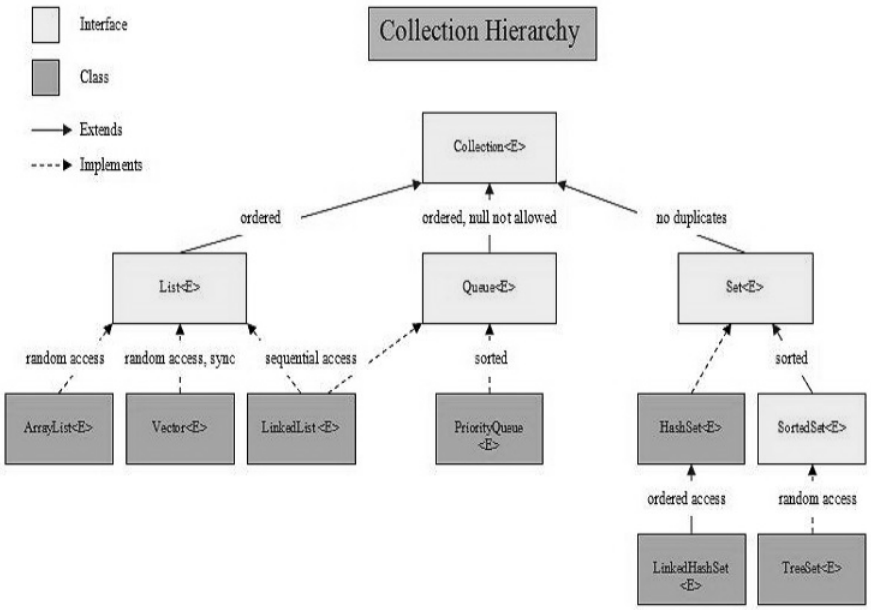
Mi a hiba?

Ezt mindenképpen bele kellene írni:

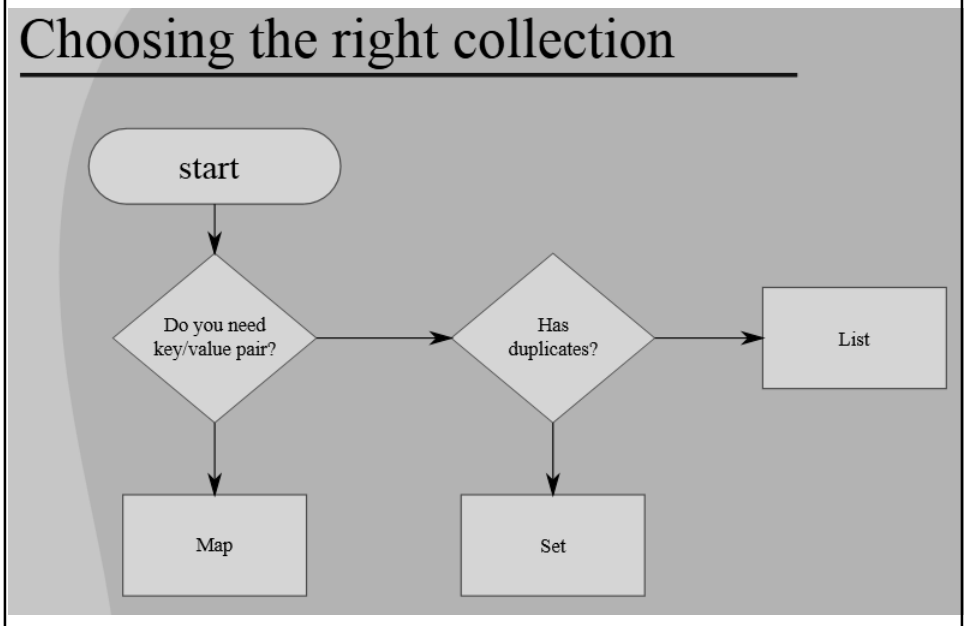
```
catch (Exception ex) {
    ex.printStackTrace();
}
```

TILOS üres catch ágat hagyni!!!

GYŰJTEMÉNYEK ÖSSZEFOGLALÁSA



GYŰJTEMÉNYEK ÖSSZEFOGLALÁSA



NÉHÁNY LINK

http://java-latte.blogspot.hu/2013/11/object-equality-in-java.html

http://java-latte.blogspot.hu/2014/02/comparable-and-comparator-interfaces-in.html

http://java-latte.blogspot.hu/2013/06/dont-know-which-mapcollection-to-use.html

https://www.tutorialspoint.com/java/java_treeset_class.htm

http://java-latte.blogspot.hu/2013/09/java-collection-arraylistvectorlinkedli.html

Kicsit más jellegű ebből a blogból:

http://java-latte.blogspot.hu/2013/08/stack-and-heap-in-java.html