A programozás alapjai 3.

Java segédosztályok

Goldschmidt Balázs
balage@iit.bme.hu

1

Másolás és azonosság

2

Objektumok azonossága

- `==` operátor
  - referencia-alapú összehasonlítás
- **`boolean equals(Object o)`**
  - tartalom alapú összehasonlítás
  - rekurzió javasolt
  - *a default megvalósítás referencia alapú!*

`a == b` ⟺ `a.equals(b)`

Azonosság példa

```
public class Car {
       private String owner;
       private String plate;
       ...
       public boolean equals(Object o) {
              Car other = (Car)o;
              return plate.equals(other.plate());
       }
}
```

rekurzió

```
Car car1 = new Car("Yogi Bear",     "ABC123");
Car car2 = new Car("Roger Rabbit",  "RR4321");
Car car3 = new Car("Yogi Bear Jr.", "ABC123");

boolean b1 = car1 == car2; boolean b2 = car1.equals(car2);
boolean b3 = car1 == car3; boolean b4 = car1.equals(car3);
```

Összehasonlítás

- Természetes (natural)
  - megvalósítja a `Comparable<T>` interfészt
  - `int compareTo(T o)`

this < o ↔ this.cmp(o) < 0
this = o ↔ this.cmp(o) = 0
this > o ↔ this.cmp(o) > 0

  - osztályonként egy implementáció
  - fordítási időben rögzül
    - persze ki lehet játszani ☺

Összehasonlítás

- Comparator segítségével
  - *Strategy* minta: felelősség egy külön osztályban
  - `interface Comparator<T>`
  - `int compare(T o1, T o2)`
    - összehasonlítja *T*-ket

o1 < o2 ↔ cmp(o1,o2) < 0
o1 = o2 ↔ cmp(o1,o2) = 0
o1 > o2 ↔ cmp(o1,o2) > 0

  - `boolean equals(Object obj)`
    - más *Comparator*-okhoz hasonlítja ezt

Összehasonlítás példa: rendezés

```
public class X
implements Comparable<X> {
  int q;
  ...
  public int compareTo(X x){
    return q-x.q;
  }
}
```

```
public class XC1
implements Comparator<X> {
  public int compare(X x1, X x2){
    return x2.q-x1.q;
  }
}
```

```
List<X> l =
    new ArrayList<>();
...

// természetes
Collections.sort(l);

// comparator-os
Collections.sort(l,
                 new XC1());
```

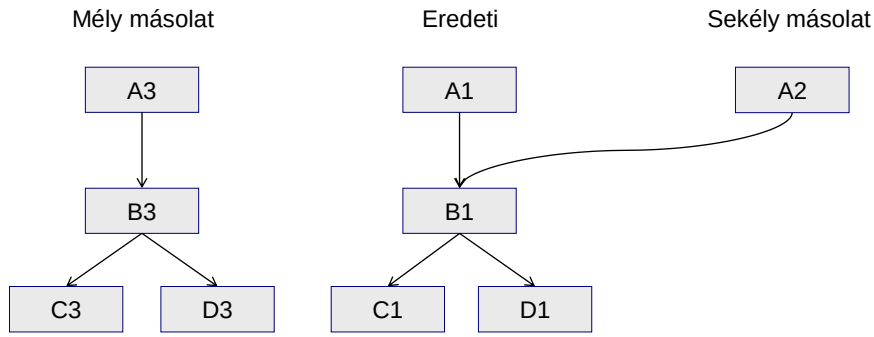

7

Objektumok másolása

- Megvalósítandó a **`java.util.Cloneable`**
- Felüldefiniálandó az **`Object.clone()`**
  - mindig hívjuk meg a **`super.clone()`** metódust
  - *Object.clone()* trükkös: leszármazottat példányosít
    - az összes attribútum a másoltból kap értéket
- *Sekély másolás (shallow copy)*
  - csak a referenciák másolódnak
    - pl. egy List a tárolt objektumok referenciáit tárolja
- *Mély másolás (deep copy)*
  - rekurzív másolás
    - pl. egy List esetén ekkor az egyes elemek másolata készül el

8

Mély és sekély másolás



9

Másolás: nincs ős, naívan

```
// ez egy naív implementáció
public class A implements Cloneable {
      B b;
      public Object clone() { // sekély
            A a2 = new A();
            a2.b = b;
            return a2;
      }
      public Object clone() { // mély
            A a3 = new A();
            a3.b = (B)b.clone();
            return a3;
      }
      ...
}
```


10

Másolás: van ős

```
public class A extends E {
      B b;
      public Object clone() { // sekély
            A a2 = (A)super.clone(); // !!!
            a2.b = b;
            return a2;
      }

      public Object clone() { // mély
            A a3 = (A)super.clone();
            a3.b = (B)b.clone();
            return a3;
      }
      ...
}
```

Ez már *clone()*-ban is lefut!

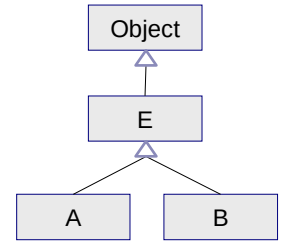
Ős *clone()*-ját hívja, de *A* példány jön létre!

11

Clone megvalósítása

```
public class E implements Cloneable {
  public Object clone() {
    try { return super.clone(); // Object.clone()-t hívjuk!
    } catch (CloneNotSupportedException e) {}
    return null;
  }
}
```

```
public class A extends E {
  B b;
  public Object clone() {
    A a3 = (A)super.clone();
    a3.b = (B)b.clone(); // mély
    return a3;
  }
  ...
}
public class B extends E { ... }
```



12

Másoló konstruktor?

```
public class A {
      B b;
      public A(A a) {
        this = (A)a.clone(); // NE!!!
      }
      public A(A a) {
        this.b = (B)a.b.clone(); // ctr vs clone
      }
      public A(A a) {
        this.b = new B(a.b); // öröklés?
      }
      ...
}
```


Mély másolás v. másoló ktr.

- (Java vs. C++)

	Előny	Hátrány
Mély másolás	absztrakt osztályoknál is jó	**new** nincs -> ki allokál (C++)?
Másoló ktr.	egységes használat **new, delete** rendben (C++)	absztrakt osztályokkal gond lesz

Gyors azonosítás: *hash*

- *public int hashCode()*
  - visszaad egy objektum-specifikus int-et
    - *a.equals(b)==true → a.hashCode() == b.hashCode()*
    - cél: objektumok gyors megtalálása és tárolása
      - pl. HashMap, HashSet
  - gyári megvalósítás az *Object*-ben: memóriacím
- Jó hash függvényt nehéz csinálni ☹
  - nem csomósodhat, gyorsan számítható kell legyen
  - tengernyi irodalom

Egy elfogadható megvalósítás

```
class Test {
       Object o1; // bármilyen típusra jó
       Object o2;
       ...
       Object on;
       public int hashCode() {
              int h = 0;

              h = 31*h+o1.hashCode(); // rekurzió!
              h = 31*h+o2.hashCode();
              ...
              h = 31*h+on.hashCode();

              return h;
       }
}
```

Enum: felsorolás OO módon

- Enum típus saját osztályt kap
  - attribútumok
  - metódusok
- Enum előnyei
  - sorosítható
  - stringgé alakítható
  - for-each ciklusban használható
  - switch-case szerkezetben is működik

```
public enum Planet {
    Mercury, Venus, Earth, Mars,
    Jupiter, Saturn, Uranus, Neptune
}
```


17

Enum példa

```
public enum Planet { // összetett enum
    Mercury(3.3e23, 2.44e6), Venus(4.868e24, 6.052e6),
      Earth(5.972e24, 6.371e6), Mars(6.417e23, 3.39e6),
      Jupiter(1.899e27, 6.991e7), Saturn(5.684e26, 5.823e7),
      Uranus(8.681e25, 2.536e7), Neptune(1.024e26, 2.462e7);

    private final double mass, radius; // kg, m

    Planet(double m, double r) { mass = m; radius = r;}

    public double mass() { return mass; }
    public double radius() { return radius; }
    public double sGrav() { return 6.674e-11*mass/radius/radius; }
}
```

```
for (Planet p : Planet.values()) {
       System.out.println(p+": "+p.sGrav());
}
```


18

Enum példa 2

```
enum Letter {A, B, C}
```

```
Letter e = Letter.A;
switch (e) {
      case A: System.out.println("A!"); break;
      case B: System.out.println("B!"); break;
      case C: System.out.println("C!"); break;
}
```

tagok importálása

```
import static mypackage.Letter.*;
```

```
if (e == B) { ... }
```


Enum metódusai

- **`String name()`**
  - a konstans neve Stringként
- **`int ordinal()`**
  - a konstans sorszáma
- **`static <T extends Enum<T>> T valueOf([Class<T> enumType,] String name)`**
  - String reprezentációból egyedi enum példány
- **`static <T extends Enum<T>> T[] values()`**
  - a típus összes értéke tömbként

Segédosztályok

Dátum- és időkezelő osztályok

- Date
  - egy időpillanatot reprezentál
    - millisec pontosság
  - használata nem javasolt
- Calendar
  - absztrakt osztály
  - különféle naptár- és időrendszerek kezelésére
- GregorianCalendar *extends Calendar*
- LocalTime, LocalDate stb. (Java 8 óta)
- DateFormat
  - dátum stringgé alakítása és stringből való beolvasása

Date

- Date() és Date(long d)
  - ktr-ok *most* vagy *d* alapján (*ms* 1970-01-01UTC00:00:00 óta)
- boolean after/before(Date d)
- int compareTo(Date d)
- int equals(Object o)
  - értelemszerű összehasonlítások
- long getTime()
  - eltelt *ms* a fenti dátum (epoch) óta
- String toString()
  - "*dow mon dd hh:mm:ss zzz yyyy*" formátummá alakít

Calendar

- Egy időpillanat leírására
  - absztrakt osztály
  - a statikus *getInstance()* visszaad egy példányt
- Az idő kezelése egységes metódusokkal
  - *add(f, delta)*
  - *set(f, delta), get(f), clear(f)*
  - *roll(f, delta)*
    - *delta*-val növeli, állítja, forgatja, lekérdezi, törli az *f* mezőt
    - szükség szerint más mezőket is állít
    - *f* értéke konstansokban definiálva

Calendar

- Mezők konstansai
  - DATE, DAY_OF_MONTH, DAY_OF_WEEK, DAY_OF_YEAR
  - WEEK_OF_MONTH, WEEK_OF_YEAR
  - ERA, YEAR, MONTH, MINUTE, SECOND, MILLISECOND
  - AM_PM, HOUR_OF_DAY (0-24), HOUR (0-12)
  - ZONE_OFFSET
- Értékek konstansai
  - JANUARY, FEBRUARY, MARCH stb.
    - vigyázat! JANUARY==0 **!!!**
  - MONDAY, TUESDAY stb.
    - SUNDAY==0, MONDAY==1 stb.
  - AM, PM
  - (GregorianCalendar: AD, BC)

Calendar

- Mindenre van metódus
  - boolean after/before(Object when)
  - int clear()
  - int get[Actual]Maximum(f)
  - int get[Actual]Minimum(f)
  - int getGreatestMinimum(f)
  - int getLeastMaximum(f)
  - get/setFirstDayOfWeek
  - get/setMinimalDaysInFirstWeek
  - …

GregorianCalendar

- *Calendar* bővítése Gergely-naptárrá
  - Konstruktorok
    - év, hó, nap *[,óra, perc [, mp]]*
  - Konstansok
    - *AD, BC*
  - Metódusok
    - setGregorianChange(Date date)
    - Date getGregorianChange()
    - boolean isLeapYear()
    - …

Calendar példa

```
Calendar c = new GregorianCalendar(1996,0,23);//96-01-23
c.set(Calendar.MONTH, Calendar.MAY); // 1996-05-23
c.set(Calendar.DATE, 31);          // 1996-05-31

c.add(Calendar.MONTH, 15);         // 1997-08-31
c.roll(Calendar.DATE, 10);         // 1997-08-10

DateFormat df = DateFormat.getDateTimeInstance();
System.out.println(df.format(c.getTime()));
   // Aug 10, 1997 12:00:00 AM
```

LocalTime, LocalDate stb.

- Egyszerű és kényelmes dátum- és időkezelés
  - package *java.time (Java 8 óta)*
  - sokkal inkább OO, mint *Date* vagy *Calendar*
    - getter/setter-ek, isAfter, isBefore, isEqual, until stb.
    - statikus példányosító metódusok (*now*, *of* stb.)
  - nem kezeli az időzónát
  - immutábilis
- *LocalTime*: csak idő (óra, perc, másodperc)
- *LocalDate*: csak dátum (év, hónap, nap)
- *LocalDateTime*: dátum és idő egyben

LocalDate, LocalTime példa

```
LocalDate d1 = LocalDate.of(1996,1,23); //96-01-23
LocalDate d2 = d1.plusDays(100);        //96-05-02
// van még plusWeeks, plusMonths, plusYears stb.
// és mindezek minusXXX alakban is...
```

```
LocalTime t1 = LocalTime.of(14,32,56); // 14h 32m 56s
LocalTime t2 = LocalTime.of(15,10);    // 15h 10m 00s
boolean b = t1.isBefore(t2);           // true
// minusSeconds, minusMinutes, minusHours, minusNanos,
// plusSeconds, plusMinutes, plusHours, plusNanos stb.
```

```
LocalDateTime dt1 = t1.atDate(d1);  // 96-01-23 14:32:56
LocalDateTime dt2 = d2.atTime(t2);  // 96-05-02 15:10:00
long hours = dt1.until(dt2, HOURS); // 2400 óra van köztük
```

StringBuffer és StringBuilder

- Mutábilis (változtatható) string reprezentációk
- String-intenzív műveletekhez
  - összefűzés (append, concat, insert stb.)
  - karaktermódosítás, törlések stb.
- StringBuffer
  - több szál esetén, lassabb működés
- StringBuilder
  - egy szál esetén, gyorsabb működés

StringTokenizer

- Megoldandó feladat
  - Sorok feldolgozása
    - konfigurációs állományok
    - szkriptek
    - stb.
- Megoldás
  - Sorok tokenekre (szavakra) törése
    - *String.split*: String-ből stringtömb
    - *StringTokenizer*: String-ből iterálható tokenek
  - Szóválasztót meg kell adni (alapból whitespace)

StringTokenizer

- Konstruktorai
  - a tokenizert először inicializálni kell
  - StringTokenizer(String str)
    - elválasztó: whitespace (szóköz, tab, újsorjel stb.)
  - StringTokenizer(String str, String delim)
    - *delim* karakterei lesznek az elválasztók
  - StringTokenizer(String s, String d, boolean retDels)
    - mint előző, de a tokenek között az elválasztókat is megkapjuk

StringTokenizer

- int countTokens()
  - tokenek száma
- boolean hasMoreElements
- boolean hasMoreTokens
  - megadja, hogy van-e még token
- Object nextElement
- String nextToken
  - visszaadja a következő tokent
- String nextToken(String delim)
  - módosítja az elválasztót és visszaadja a következő tokent

StringTokenizer példa

```
StringTokenizer st =
      new StringTokenizer("alpha beta gamma");
while (st.hasMoreTokens()) {
      System.out.println(st.nextToken());
}
```

```
alpha
beta
gamma
```


Scanner osztály

- Karakter alapú inputhoz
  - BufferedReader
    - macerás
    - összetett
    - olvashatatlan lesz a kód
  - Scanner
    - közvetlenül a forráshoz nyúl
    - iterátorszerű használat
    - különféle konverziókat is támogat
      - pl. következő *double* beolvasása

Scanner osztály

- Konstruktorai
  - paraméter: *File*, *InputStream*, *Path*, *Readable*, *String*
- Metódusok
  - *hasNext[XXX]*
  - *next[XXX]*
    - BigDecimal, BigInteger, boolean, byte, double, …
  - *useDelimiter(Pattern|String), delimiter()*
  - *useRadix(int radix), radix()*
    - elválasztó és számrendszer (radix) beállítása
  - *findInLine, skip, match* stb.

Properties osztály

- Konfiguráció fájl feldolgozása
  - kulcs-érték párok szövegfájlban
  - hogyan kezeljük kényelmesen?
    - saját parsert írunk: overkill
    - meglevő megoldás használunk: *Properties*
- *Properties* osztály
  - *Map* interfészt valósítja meg
  - kényelmes a használata
    - többfajta formátumot támogat (egyenlőségjel, kettőspont, xml stb.)

Properties osztály

- *java.util.Properties* osztály
  - *Map* interfész *(String key, String value)*
  - *getProperty*, *setProperty*
    - beállítások lekérdezése, módosítása
  - *Set<String> stringPropertyNames()*
    - összes beállítás nevének lekérése
  - *load(InputStream vagy Reader)*
    - kulcs-érték párok beolvasása
    - sokfajta formátumból
  - *store(OutputStream vagy Writer, String comments)*
    - tartalom kiírása bárhova (fájl, hálózat stb.)
  - XML formátumot is támogatja

Random osztály

- Véletlen számok generálásához
  - konstruktorok
    - *default*: a seed automatikusan
    - *seeded* (*long* paraméterű ktr.), determinisztikus *(setSeed)*
  - *nextXXX()*
    - egyenletes eloszlás
    - *boolean*, *bytes*, *int*, *long*: eredmény az értékkészlet egészén
    - *double*, *float*: eredmény a [0.0, 1.0) halmazon
    - *nextInt(int n)*: eredmény 0 and *n* között (jobbról nyílt, *n* nélkül)
  - *nextGaussian()*
    - normális eloszlás (várható érték: 0, szórás: 1)

Math/StrictMath

- Utility osztályok (csak statikus metódusok)
  - java.lang.Math, java.lang.StrictMath
- Konstansok
  - E, PI
- Függvények
  - abs, signum, sqrt, *cbrt, ceil, floor, round, rint,*
  - sin, cos, tan, *sinh, cosh, tanh*
  - asin, acos, atan, atan2
  - pow, exp, *expm1,* log, log10, *log1p, scalb,*
  - max, min, *nextAfter, nextUp,* toDegrees, toRadians

Nagy számok kezelése

- BigInteger
  - tetszőleges méretű egész számok pontos kezelése
  - konstansok
    - *0 (ZERO)*, *1 (ONE)*, *10 (TEN)*
  - konstruktor *byte[]*, *random* vagy *String* paraméterrel
  - inicializálás *long*-ból (*valueOf*)
  - műveletek metódusként
    - *add*, *abs*, *and*, *pow* stb.
  - módosítók
    - *setBit*, *testBit*, etc
  - lekérdezések
    - *toByteArray*, *toString*, *longValue* stb.

Nagy számok kezelése

- BigDecimal
  - tetszőleges méretű és pontosságú, decimális, lebegőpontos
  - konstansok
    - *0 (ZERO)*, *1 (ONE)*, *10 (TEN)*
    - *rounding (ROUND_DOWN, ROUND_UP, ROUND_HALF_UP, etc)*
  - konstruktorok *char[]*, *String*, *BigInt.*, *double*, *long*, *int* param-rel
  - műveletek metódusként
    - *add*, *abs*, *round*, *pow* stb.
  - módosítók
    - *setScale*, *testBit* stb.
  - lekérdezések
    - *scale, precision, toBigInteger*, *toString*, *doubleValue* stb.

Köszönöm a figyelmet!