5. Gyakorlat

legendi@inf.elte.hu

2010. március 9.

Assertek

Motiváció: `http://geekandpoke.typepad.com/geekandpoke/2010/02/simply-explained-edge-cases.html`

Általános forma:

```
assert <boolean>;
assert <boolean> : <non-void>;
```

Az első paraméter logikai feltétel, ha hamis, akkor AssertionError, a második paraméter lesz az üzenete. Futtatási idejű ellenőrzések, feltételezések biztosítására, azonban könnyen ki-, és bekapcsolhatók (esetenként erőforrásigényes a kiértékelés, pl. minimum elem meghatározása egy komplex, rendezetlen adatszerkezetben). Fejlesztés során rendkívül hasznosak.

Mikor **ne** használjuk?

- publikus függvények argumentumellenőrzésére (NullPointerException, IllegalArgumentException, etc.), ezeknek akkor is teljesülniük kell, mikor az asserteket kikapcsolják (specification, contract)
- Mellékhatással ne járjon! Pl.:

  ```
  boolean b = false;
  assert b = true; // Broken!
  ```

Mikor használjuk?

- Bekső invariánsok: ha commentbe állítunk invariáns tulajdonságot, pl.:

  ```
  if (i % 3 == 0) {
      ...
  } else if (i % 3 == 1) {
      ...
  } else { // i % 3 == 2
      ...
  }
  ```

Ebből:

```
if (i % 3 == 0) {
    ...
} else if (i % 3 == 1) {
    ...
} else {
    assert i % 3 == 2 : "Hiba: " + i;
    ...
}
```

- Control flow invariant: feltételezhetően elérhetetlen kódrészletekhez, pl.:

  ```
  try {
      ...
  } catch (Exception e) {
      // Never happens
  }
  ```

  Helyett:

  ```
  try {
      ...
  } catch (Exception e) {
      assert false : "Never happens";
  }
  ```

- Elő-, utófeltételek, invariánsok. Nem egy teljes design-by-contract eszköz, de segít informatívan, és ehhez hasonló módszerrel kódolni. **Megjegyzés:** `public` függvények paraméterének ellenőrzésére **ne**. Előfeltétel: paraméterek ellenőrzése, utófeltétel: return előtt. Invariáns: minden publikus metódus előtt-után ellenőrizhető (private esetben az objektum épp lehet köztes állapotban). Bonyolultabb feltétel kiemelhető private függvénybe.

Alapból kikapcsolt, bekapcsolás:

```
> java -ea ...
```

Részletesen: `http://java.sun.com/j2se/1.4.2/docs/guide/lang/assert.html`

Generic - Bevezetés

(Csak a collectionökhöz, minimális szinten) Típussal paraméterehzetőség. Type erasure: csak fordítási időben ismert a típusinformáció, utána automatikusan törli a fordító, bájtkódból nem szerezhető vissza - nem C++ template-ek, nem generálódik fordítási időben új típus, nincs template metaprogramozás.

Nem kötelező velük foglalkozni (@SupressWarnings("unchecked")), de rendkívül hasznosak, fordítási időben tudunk potenciális hibalehetőségeket kiszúrni - persze ez is a programozón múlik. Kényelmes, típusbiztos.

Collectionöknél aktívan használjuk őket:

```
Vector<String> s = new Vector<String>();
```

Az előnyük:

```
// Pre-1.5 era:
Vector v = new Vector();

v.add( new Integer(1) );
v.add( new Integer(2) );

for (int i=0, n=v.size(); i<n; ++i) {
    Integer act = (Integer) v.get(i);
    System.out.println(act);
}

// Uj:
Vector<Integer> v = new Vector<Integer>();
v.add(1);
v.add(2);

for (int i=0, n=v.size(); i<n; ++i) {
    Integer act = v.get(i);
    System.out.println(act);
}
```

Autoboxing-unboxing: csak objektum referenciákat tárolhatnak, ezért primitív típusok helyett wrapper osztályokat kell használnunk - azonban ezek ilyen esetekben automatikusan konvertálódnak (ld. fenti példa). **Amire figyelni kell:** teljesítmény, == operátor, null unboxing NPE-vel jár.

Gyűjtemény keretrendszer

Collections Framework, java.util.* csomag, objektumok memóriában tárolására, lekérdezése, manipulálása (c++ STL). Általános célú adatszerke-

zetek:

- Collection
  - List
  - Deque
  - Set
    - SortedSet
- Map
  - SortedMap

Nem megvalósított művelet UnsupportedOperationException-t dob. Copy konstruktorok vannak (egyik a másikra konvertálható). Műveletek 3 csoportja:

1. Alapvető műveletek: size(), isEmpty(), contains(), add(), remove()
2. Elemek együttes kezelése: addAll(), containsAll(), removeAll(), clear(), retainAll()
3. Tömbbé konvertálás - gány:

```
A[] arr = (A[]) sitcom.toArray(new A[sitcom.size()]);

// Kicsit egyszerubb, bar kevesbe hatekony, biztonsagos:
A[] arr = (A[]) sitcom.toArray();
```

Iterátorokkal rendelkeznek, használhatók for-each-ben. Példa:

```
package gyak5;

import java.util.Vector;

public class VectorTest {
    public static void main(String[] args) {
        Vector<Double> vector = new Vector<Double>();

        for (int i=0; i<5; ++i) {
            vector.add( Math.random() );
        }

        System.out.println(vector);
    }
}
```

Halmaz

Duplikált elemeket nem tartalmazhat, kell hozzá az objektumon az equals() és hashCode() (hashelő implementációk, nem számít a sorrend, 2 halmaz egyenlő, ha ugyanazokat az elemeket tartalmazzák). HashSet, TreeSet: előbbi hatékonyabb, utóbbi rendezett.

Feladat

Készíts egy programot, amely megszámolja a parancssori argumentumokra, hogy azokban hány különböző betű van! A megvalósításhoz használj halmazt (Set<Character>, String#toCharArray())! Példa output:

```
> java gyak5.CharCounter asdfasd jkl
asdfasd -> 4
jkl -> 3
```

Lista

Elemek pozíció szerinti elérése, iteráció, részlista kezelés. A remove() az elem 1. előfordulását távolítja el, az add(), addAll() a lista végéhez fűz hozzá. Két lista egyenlő, ha ugyanazokat az elemeket tartalmazzák, ugyanabban a sorrendben. A lista iterátora a ListIterator, 2 irányban is bejárható: hasNext(), next(), ill. hasPrevious(), previous(). Részlista: balról zárt, jobbról nyílt intervallumot kell megadni. Két implementáció: ArrayList, LinkedList, előbbi a pozicionális műveleteknek kedvez, utóbbi akkor, ha a lista elejére kell sokat beszúrni, és iteráció közben törölni (általában az ArrayList használata a célravezetőbb).

Feladat

Készíts egy programot, amely a parancssori argumentumaként megkapott szavakat lexikografikusan lerendezi, majd kiírja a képernyőre. A megvalósításhoz használj egy tetszőleges lista adatszerkezetet (ArrayList, LinkedList, Stack, Vector), valamint a java.util.Collections#sort() függvényt! Példa output:

```
> java gyak5.StringSorter ad df vc y a
[a, ad, df, vc, y]
```

Leképezés

Kulcs-érték párokhoz: HashMap, Hashtable (minimális különbség: utóbbi szinkronizált, megengedi a null értékeket is). Minden kulcshoz egy érték tartozhat. Nem iterálható, azonban lekérdezhető a keySet(), entrySet(), ami már igen.

Feladat

Készíts egy programot, amely megszámolja egy fájlból az egyes szavak előfordulásainak számát! A program a fájl elérési útját argumentumként kapja. A megvalósításhoz használj egy String → Integer leképezést (HashMap<String, Integer>)! Példa output:

```
> cat test.txt
a a a b b c d
> java gyak5.WordCounter test.txt
{d=1, b=2, c=1, a=3}
```

Rendezés

Beépített típusoknak értelemszerű a relációja - felhasználói típusokat a programozó dönti el. Comparable interfész → compareTo() metódusa, melynek eredménye int típusú:

- 0, ha a két objektum egyenlő
- < 0, ha az adott objektum kisebb a paraméternél
- > 0, ha fordítva

Implementálás:

```
class Foo implements java.util.Comparable<Foo> {
    ...
    public int compareTo(final Foo foo) {
        return ...;
    }
}
```

Ha ennek használatára nincs lehetőség, marad egy saját Comarator készítése (pl. egyazon objektumot több szempont szerint kell rendezni).

Feladat

Készítsetek egy Date osztályt, amely tartalmazza az év, hónap, nap adatokat (mind számok). Implementáljátok vele a Comparable<Date> interfészt, és ennek megfelelően valósítsátok meg a compareTo() függvényt! Hozzatok létre kódból 3 objektumot, és tároljátok el ezeket egy tetszőleges lista adatszerkezetbe. Ezt aztán rendezzétek le kronológiai sorrend szerint a Collections#sort() függvénnyel, és írjátok ki az eredményt!

Kényelmi lehetőségek

1. java.util.Arrays#asList(): tömbből listát csinál
2. java.util.Collections
   - nCopies(int n, Object o): két paraméter, amely n-szer tartalmazza o-t.
   - Egyelemű, üres, módosíthatatlan, szinkronizált listák
   - Algoritmusok:
     - Rendezés, összefésüléses módszerrel ($nlog(n)$, rendezett listát már nem rendez, szemben a quick sorttal): sort()
     - Összekeveerés: shuffle()
     - Megfordítás, feltöltés, másolás: reverse(), fill(), copy()
     - Bináris keresés: $(-i-1)$-et ad vissza, ahol i az első olyan elem indexe, amely nagyobb az elemnél.
     - Minimum, maximum elem: min(), max()

Megjegyzések:

- $capacity() != size()$
- Az interfész műveleteken kívül rengeteg egyéb hasznos funkcionalitás, érdemes a javadocot olvasgatni
- Saját implementációk: hajrá! A Collections Framework absztrakt osztályokat biztosít (AbstractList, AbstractSet, etc.), lehet származtatni.
- További adatszerkezetek: Dequeue, Stack, BitSet, Vector, etc.
- Felhasználás: paraméterként, változódeklarációként célszerű minél általánosabb interfészt megadni (a collections framework előnye a rugalmassága):

```
Vector v1 = new Vector();  // vektorkent kezeles
List   v2 = new Vector();  // listakent kezeles
```

Részletesen: http://java.sun.com/javase/6/docs/api/java/util/package-summary.html

+/- Feladat

A feladatokat a `legendi@inf.elte.hu` címre küldjétek, **szombat éjfélig**! A subject a következőképp nézzen ki:

`csop<csoportszám>_gyak<gyakorlat száma>_<EHA-kód>_<a megoldott feladatok száma>`

Például:

`csop1_gyak5_LERIAAT_1`

Mellékelni csak a Java forrásfájlokat mellékeljétek (semmiképp ne teljes Eclipse/NetBeans projecteket), esetleg rarolva, zipelve, és egy levelet küldjetek (több levél esetén az utolsó mellékleteit értékelem)! További fontos kritériumok:

- A konvenciókra figyeljetek plz!
- Ékezetes karaktereket **ne** használjatok! Főleg azonosítók esetében ne! A Java ugyan ezt megengedi, ugyanakkor a különböző környezetekbe való konvertáláskor ($latin2 \leftrightarrow UTF-8 \leftrightarrow Cp1250$) összetörnek a karakterek! Az ilyen forrásokat fordítani, következésképp értékelni sem tudom.

1. Feladat

Készítsünk egy sorozat rendező alkalmazást! A program inputja a következő formátumú fájl legyen:

```
# evad : epizod : cim
12:7:Super Fun Time
12:4:Canada on Strike
8:13:Cartman's Incredible Gift
10:8:Make Love, Not Warcraft
```

A # karakterrel kezdődő sorokat hagyjuk figyelmen kívül (String#trim())! Készítsünk egy Sitcom osztályt a megfelelő adattagokkal (season, episode, title), és implementáljuk vele a Comparable<Sitcom> interfészt! A compareTo() működjön úgy, hogy elsődleges szempont szerint az évad, azon belül pedig az epizódszám alapján rendezzen! Az adatokat tároljuk egy Vector<Sitcom> adatszerkezetben. A saját osztály helyes működéséhez implementáljuk az equals(), toString(), hashCode() függvényeket is!

2. Feladat

Egészítsük ki az előző programot úgy, hogy dobjon fel egy ablakot, amelyben egyedül egy JList objektum legyen. Ez a konstruktorában kapja meg az elemek rendezett listáját, hogy megjeleníthesse azokat! A program képernyőképe az 1 ábrán látható.

1. ábra. Képernyőkép