Java tutorial

Beágyazott osztályok

- A blokkstrukturáltság támogatása
  - Eddig: egymásba ágyazható blokk utasítások
- Osztálydefiníciók is egymásba ágyazhatók
- Sokszor kényelmes: frappáns, kompakt kód
- Vigyázat: könnyű nehezen olvashatóvá, bonyolulttá tenni a programunkat vele

Osztályok fajtái

- Kutya közönséges osztályok
- Beágyazott osztályok
  - statikus tagosztályok

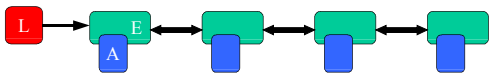
jelentős minőségi különbség

  - (nem statikus) tagosztályok
  - lokális osztályok
  - névtelen osztályok

} **belső osztályok**

Alapprobléma

- Saját lista adattípus megvalósítása
- Lista, Elem
  - Az Elem osztály egy segédosztály
  - Láncolt ábrázolás



Kapcsolat a Lista és az Elem között

- Nagyon erős kapcsolat közöttük
- Kifelé csak a Lista fontos, az Elem nem
- Megoldás: tegyük őket egy csomagba, de csak a Lista lesz publikus
- Mi van, ha kell egy Fa adattípus is?
  - Külön csomagba
    - túl sok csomag lesz, nehéz használni
  - Ugyanabba a csomagba
    - elveszik az elkülönültség előnye
    - mégis ez lez a megoldás... :)

Cél

- Egy csomagon belül van Lista, Elem, Fa, stb.
- A Lista és az Elem erősebb kapcsolatban legyen, mint Lista és Fa, vagy Elem és Fa
- A Lista és az Elem lássák egymás privát dolgait is, és a Fa egyáltalán ne lássa az Elem osztályt
- Rejtsük el az Elem osztályt a Lista osztályon **belülre**

Statikus tagosztály

- Tagok: adattagok és metódusok; plussz lehetnek osztályok is!
- Statikus tag: osztályszintű tag
  - static kulcsszó

```
public class Lista {
  static private class Elem { ... }
  private Elem első = null;
  ...
}
```

```
public class Lista {
  private Elem első;
  private static class Elem {
    Object adat;
    Elem előző, következő;
    Elem(Object adat, Elem előző, Elem következő){
      this.adat = adat; this.előző = előző;
      this.következő = következő;
    }
  }
  public void beszúr (Object adat) {
    első = new Elem(adat,null,első);
    if (első.következő != null)
      első.következő.előző = első;
  }
  ...
}
```

A statikus tagosztályok által kínált előnyök

- Logikai összetartozás kifejezése
  - kompaktság, olvashatóság
- Tokbazárás, adatelrejtés támogatása
  - Az Elem segédosztályt elrejtettük a külvilág elől
- A beágyazott osztály hozzáfér a befoglaló osztály privát adataihoz
  - A Felsoroló osztályból használhatjuk a Lista osztály privát adatait
  - Természetesen a befoglaló is a beágyazottéihoz

Java tutorial


```
public class Lista {
    private Elem első = null;
    ...
    private static class Felsoroló
                    implements Iterator {
        final Lista lista;
        Elem aktuális;
        Felsoroló (Lista lista) {
            this.lista = lista;
            aktuális = lista.első;
        }
        ...
    }
    public Iterator felsoroló(){
        return new Felsoroló(this);
    }
}
```

Használat

- A megfelelő könyvtár importálása
- A Lista osztályból példány(ok) létrehozása
- Feltöltés adatokkal
  - létrejönnek az Elem objektumok a háttérben
- Egy iterátor lekérdezése
  - egy Felsoroló osztályú objektumot kapunk
  - hozzákapcsolódik a listához, amit be kell járnia
- Lista bejárása az iterátor segítségével
  - az iterátor tudja, melyik listát kell bejárnia

Nem statikus tagosztály

- Amikor létrehozunk egy Felsoroló objektumot, akkor az hozzákapcsolódik egy Lista objektumhoz
- Objektumok közötti kapcsolatot építettünk ki
- Ezt automatikusan megtehetjük nem statikus tagosztály segítségével is
- Tagként definiált osztály
  - példányszintű
  - nem kell a static kulcsszó

```
public class Lista {
    private Elem első = null;
    ...
    private static class Felsoroló
                    implements Iterator {
        final Lista lista;
        Elem aktuális;
        Felsoroló (Lista lista) {
            this.lista = lista;
            aktuális = lista.első;
        }
        ...
    }
    public Iterator felsoroló(){
        return new Felsoroló(this);
    }
}
```

Java tutorial


```
public class Lista {
    private Elem első = null;
    ...
    private class Felsoroló implements Iterator {
        Elem aktuális = első;
        ...
    }
    public Iterator felsoroló(){
        return new Felsoroló();
    }
}
```

```
első      Lista.this.első
```

- Felsoroló objektum létrehozása csak egy Lista objektumhoz képest lehetséges!

```
public class KülsőFelsoroló
extends Lista.Felsoroló {
    public KülsőFelsoroló (Lista lista) {
        lista.super();
        ...
    }
    ...
}
```

- Felsoroló objektum létrehozása csak egy Lista objektumhoz képest lehetséges!

Lokális osztályok

- Lokális változók mintájára
- Csak a befoglaló metóduson belül használható az osztálydefiníció
  - A lokális osztályú objektum viszont túlélheti az osztályát befoglaló metódust
- Pl. a Felsoroló osztályra csak a felsoroló() metódusban van szükség - definiáljuk ott!
  - A kód kompaktsága tovább nő ezzel

```
public class Lista {
    private Elem első = null;
    ...
    public Iterator felsoroló(){
        private class Felsoroló
        implements Iterator {
            Elem aktuális = első;
            ...
        };
        return new Felsoroló();
    }
}
```

Egy másik példa...

- Alprogram típus helyett Java-ban interfészt használhatunk:
  - Valamilyen objektumon végzett művelet absztrakciója
    - type Művelet = procedure (adat: Object);
    - void (*f)(Object);

```
public interface Művelet {
     void végrehajt (Object adat);
}
```

Belső iterátor

```
public class Lista {
    ...
    public void mindre (Művelet művelet) {
        for ( Elem elem = elso;
              elem != null;
              elem = elem.kovetkezo )
            művelet.végrehajt(elem.adat);
    }
}
```

- Valamilyen műveletet végrehajtunk minden listaelemen

```
static int multiplicitás
(Lista lista, final Object minta) {

    class Számláló implements Művelet {
        int n = 0;
        public void végrehajt (Object adat) {
            if (adat.equals(minta))
                n++;
        }
    }

    Számláló számláló = new Számláló();

    lista.mindre(számláló);
    return számláló.n;
}
```

- Használhatjuk a metódus paramétereit és lokális változóit, csak annyi kell, hogy final-ek legyenek
- A metódus paramétere és lokális változója túlélheti a metódust

A final megkerülése

- Ha azt akarjuk, hogy egy paraméter vagy egy változó értéke megváltozhasson, de a final megmaradhasson valahogy, akkor tegyük be egy egy hosszúságú tömbbe

```
static int multiplicitás
(Lista lista, final Object[] minta) {...}
```

Java tutorial

Névtelen osztályok

- Igazándiból a Felsoroló osztály definíciójára csak egy kifejezésben van szükségünk. Sőt, magára a Felsoroló osztályra sincs szükségünk, csak egy példányra belőle.
- **Egy kifejezésben szereplő osztálydefiníció**

```
public class Lista {
    private Elem első = null;
    ...
    public Iterator felsoroló(){
        return new Iterator(){
            Elem aktuális = első;
            ...
        };
    }
}
```

- Névtelen osztály, fordítás után `Lista$1.class`
- Interfész megvalósítása vagy osztály kiterjesztése
- Értelemszerű megkötések, pl. konstruktor sem írható

Beágyazott osztályok használata

- Névtelen osztály a leggyakrabban
- AWT-ben eseménykezelők írása
- Observer

Java tutorial

Feladat

- Listázzuk ki a tmp alkönyvtár összes txt kiterjesztésű fájljának a nevét.
- Ezt a feladatot már korábban megoldottuk. Írjuk át névtelen osztályosra a megoldást!