Programozási nyelvek Java

Kozsik Tamás előadása alapján
Készítette: Nagy Krisztián

5. előadás

```
class Point{
      int x,y; // int x = 0; int y = 0;
      Point(int x, int y){
            this.x = x;
            this.y = y;
      }

      Point(){}
}
```

new Point(1,2); és new Point(); -al történő létrehozás is helyes.

Túlterhelés (Overloading)

- ugyan azon információ többféleképpeni átadása
- általános – speciálisabb paraméterezési változat
- (Java-ban nem lehet a formálisan definiált paramétereknek default értéket adni)
- A compiler eldönti, hogy a túlterhelt változatok közül melyiket kell végrehajtani. Ez egy fordításiidejű döntés

Egy vagy több azonos elnevezésű, szignatúrájában csak a formális paraméterlista típusaiban eltérő metódusok esetén túlterhelésről beszélünk.

```
void set(int x, int y){
      this.x = x;
      this.y = y;
}

void set(Point other){
      this.x = other.x;
      this.y = other.y;
}
```

Point p = new Point();
p.set(1,2);
p.set(new Point(1,2));

```
class Point{
      int x,y;
}
```

Ebben az esetben is tudunk new Point()-al létrehozni új objektumot, ekkor a default konstruktor hajtódik végre.
Default konstruktor: Olyan kostruktor, melynek nincsen paramétere és nincs leírva az osztályban.
(Így nézne ki: Point(){})

Ebben az esetben, amennyiben set(new Point(x,y));-t hívnánk, úgy végtelen kölcsönös rekurzió jönne létre, mely következményeként túlcsordul a stack.

Egy konstruktorban meghívhatunk egy másik konstruktort is a this([PARAMÉTER_LISTA]);
segítségével, viszont ez a speciális metódus csak az adott konstruktor első utasítása lehet.
(Csak a legelső tevékenység lehet egy másik konstruktor meghívása)

Ezek a változatok sokszor hívják egymást azért, hogy a redundanciát (kódismétlést) elkerüljük.

Default konstruktor csak akkor jön létre, ha egyáltalán nincs konstruktor írva az adott osztályhoz.

<u>Láthatóság</u>

A hatókörből kivonjuk az ugyan olyan néven deklarált változók hatókörét.
(Eddig csak egymásba ágyazott blokkoknál beszéltünk láthatóságról.
Most osztály szinten beszélünk róla)

public - bárhol hozzáférhetünk, a csomagon kívül is
private - csak az osztály területén belül férünk hozzá
nem írunk semmit - bárhol hozzáférhetünk, de csak a csomagon belül
protected - az öröklődés kapcsán lesz róla szó

Ökölszabály, hogy az osztályban található mezők private láthatóságúak legyenek.
Ebben az esetben szükségünk van olyan speciális függvényekre, melyekkel ezen private láthatóságú értékeket manipulálni tudjuk. Le tudjuk kérdezni és be tudjuk állítani.
A lekérdező függvényeket getter, míg a beállító eljárásokat setter metódusoknak nevezzük.

```
class Point{
      private int x,y;

      int X(){ return this.x; }
      void X(int x) { this.x = x; }
      int Y(){ return this.y; }
      void Y(int y) { this.y = y; }
}
```

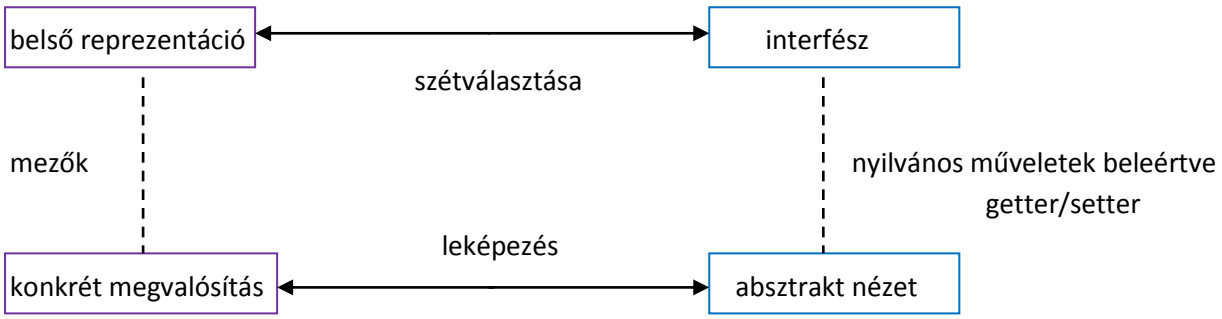
Point p = new Point();
p.x = 2 - HIBÁS! - Osztálydefiníción kívül nem lehet állítani így.
p.X(2); - Helyes
int x = p.X(); - Helyes

Hogy ne kavarodjunk bele a sok X és Y-ba, ezért gyakrabban az alábbit használjuk:

```
class Point{
      private int x,y;

      int getX(){ return this.x; }
      void setX(int x) { this.x = x; }
      int getY(){ return this.y; }
      void setY(int y) { this.y = y; }
}
```

Miért használjuk ezt?
Segít a belső reprezentációt elválasztani az interfésztől, azaz a konkrét megvalósítás és az absztrakt nézet között létrehoz egy leképezést.



Szűk interfészeket definiáljunk, mert négyzetesen hat a programunk komplexitására.
Legyen funkcionálisan teljes.

```
class Point{
      private int[] coordinates = {0,0};

      int getX(){ return coordinates[0]; }
      void setX(int x) { coordinates[0] = x; }
      int getY(){ return coordinates[1]; }
      void setY(int y) { coordinates[1] = y; }
}
```

A reprezentáció megváltoztatása nem hat a Point osztály klienseire.
Kevésbé költséges továbbfejlesztés és karbantartást biztosít.
Kisebb változások hatása lokalizált.

Végezetül az N dimenziós Point egy helyes megvalósítása:

```
class PointN{
      private int[] coordinates;

      PointN(int dimension){
            coordinates = new int[dimension];
      }

      void set(int dim, int val){
            coordinates[dim] = val;
      }
      // …

      void set(int[] coords){
            for(int i = 0; i < coords.length; ++i){
                  coordinates[i] = coords[i];
            }
      }
}
```

A void set(int[] coords){ coordinates = coords; } továbbra is HIBÁS megoldás!

Az implementáció – interfész szétválasztásánál fontos a típus invariáns megörzése:

```
class Line {
      private Point p,q;
…
}
```

Invariánsok: p != 0 & q != 0 & (p.x != q.x | p.y != q.y)

A konstruktor beállítja, az interfészben található műveletek megörzik ezeket az állításokat.