Java és web programozás

Kovács Kristóf

Budapesti Műszaki Egyetem

2015. 02. 11.

2. Előadás

Néhány programozási módszer:

- Idők kezdetén való programozás
- Struktúrált
- Moduláris
- Funkcionális
- Objektum-orientált
- ...

Néhány programozási módszer:

- Idők kezdetén való programozás
- Struktúrált
- Moduláris
- Funkcionális
- Objektum-orientált
- ...

Idők kezdetén:

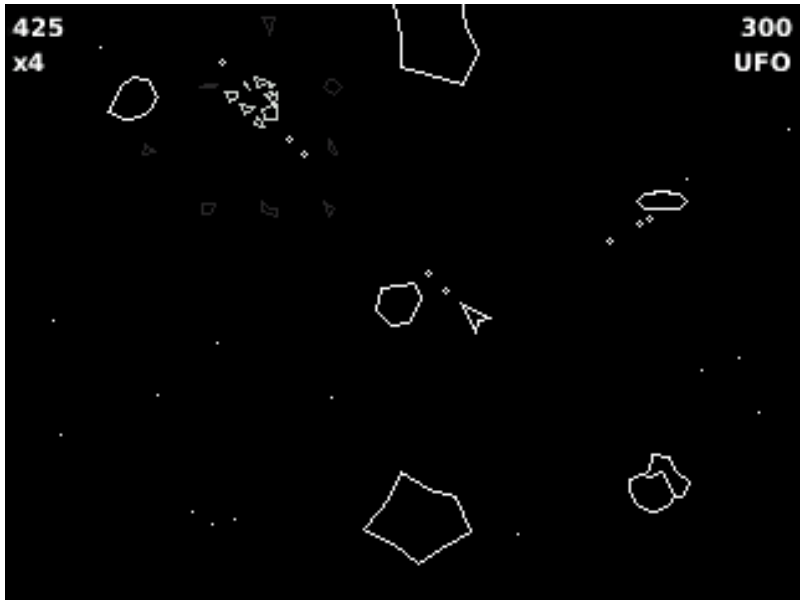
- Ránézésre semmi szerkezet
- Csak az érti aki írta
- Jóformán lecserélhetetlenné teszi a programozót

Struktúrált programozás

- Struktúrált adatok, műveletek
- Áttekinthetőbb, hordozhatóbb
- Nem túl hatékon, nehezen cserélhető

Struktúrált programozás

- Struktúrált adatok, műveletek
- Áttekinthetőbb, hordozhatóbb
- Nem túl hatékon, nehezen cserélhető




```
AsteroidVector asteroidDestroyed(Asteroid ast) {
  int n = ast.size; // jobb név: numberOfNewAsteroids
  AsteroidVector newAsteroids = createAsteroidVector(n);
  int i;
  for(i = 0; i < n; i++) {
    newAsteroids.push(instantiateFragmentAsteroid(ast.pos, n));
    ...
  }
  return newAsteroids;
}
```

Moduláris programozás

- Egy modul egy önálló egység
- Interfészekkel kapcsolódik más modulokhoz
- Önállóan fordítható, tesztelhető
- A modulok nem tudnak egymás működéséről
- Az adattípusok megjelennek az interfészeken
- Nehezen cserélhető egy adattípus

Moduláris programozás

- Egy modul egy önálló egység
- Interfészekkel kapcsolódik más modulokhoz
- Önállóan fordítható, tesztelhető
- A modulok nem tudnak egymás működéséről
- Az adattípusok megjelennek az interfészeken
- Nehezen cserélhető egy adattípus

```
AsteroidHandler.c
GameHandler.c
PlayerController.c
Physics.c
Renderer.c
...
```

Objektum-orientált megoldás

```
class Asteroid implements Renderable, Destroyable... {
   private int size;
   private Position pos;
   ...
   public AsteroidContainer destroy() {
      AsteroidContainer newAsteroids = new AsteroidVector(size);
      for(int i = 0; i < size; i++) {
          Asteroid a = AsteroidFactory.createFragmentAsteroid(
                                pos, size);
          ...
      }
      return newAsteroids;
   }
   public void render(Renderer rend) {
      ...
   }
}
```

Objektum-orientált programozás

- Egységbe zárás
- Felbontás egyszerűbb feladatokra
- Megtervezhető a program anélkül, hogy végig kellene gondolni a megírásához szükséges algoritmusokat
- Interfészek kötik össze az absztrakt részeket
- Objektum:

Objektum-orientált programozás

- Egységbe zárás
- Felbontás egyszerűbb feladatokra
- Megtervezhető a program anélkül, hogy végig kellene gondolni a megírásához szükséges algoritmusokat
- Interfészek kötik össze az absztrakt részeket
- Objektum:
  - Könnyen módosítható
  - Általánosítható
  - Tárolja a saját állapotát
  - Elrejti a belső adatszerkezetét, a műveleteinek algoritmusait
  - Minél absztraktabb annál jobb, a használathoz ne kelljen ismerni a működését
  - Pl: Komplex szám

Komplex szám

```
public class Complex {
    private float rePart_;
    private float imPart_;

    public Complex() {
        rePart_ = 0;
        imPart_ = 0;
    }

    public Complex(float rePart) {
        rePart_ = rePart;
        imPart_ = 0;
    }
```

```
    public Complex(float rePart, float imPart) {
        rePart_ = rePart;
        imPart_ = imPart;
    }

    public Complex add(Complex other) {
        float rePart = this.rePart_ + other.rePart_;
        float imPart = this.imPart_ + other.imPart_;
        Complex retval = new Complex(rePart, imPart);
        return retval;
    }
}
```

```
    public Complex(float rePart, float imPart) {
        rePart_ = rePart;
        imPart_ = imPart;
    }

    public Complex add(Complex other) {
        float rePart = this.rePart_ + other.rePart_;
        float imPart = this.imPart_ + other.imPart_;
        Complex retval = new Complex(rePart, imPart);
        return retval;
    }
}
```

- Objektum a konkrét adat és a rajta végezhető műveletek
- Osztály egy típus, melyből létrehozhatunk konkrét példányokat, objektumokat.

Adattakarás javaban

```
private float rePart_;
private float imPart_;
```

- Ami *private* csak az osztályon belülről érhető el, kívülről nem.
- *public* dolgok mindenhonnan elérhetőek.
- Lesz még egy *protected* jelző is, ez majd ha már örökléssel foglalkozunk.
- Valamint Java-ban van az elég speciális *default*, ha nem írjuk ki egyiket se. Ekkor a saját package-én belül elérhetõ, de mindenki másnak *private*.

Adattakarás javaban

```
private float rePart_;
private float imPart_;
```

- Ami *private* csak az osztályon belülről érhető el, kívülről nem.
- *public* dolgok mindenhonnan elérhetőek.
- Lesz még egy *protected* jelző is, ez majd ha már örökléssel foglalkozunk.
- Valamint Java-ban van az elég speciális *default*, ha nem írjuk ki egyiket se. Ekkor a saját package-én belül elérhetõ, de mindenki másnak *private*.

```
public class Main {
    public static void main(String[] args) {
        Complex comp = new Complex(5, 6);
        System.out.println(comp.rePart_); // Hiba
        System.out.println(comp); // Nem hiba
    }
}
```

- Mindez vonatkozik változókra és függvényekre is
- Létre lehet hozni egy *private* függvényt, és *public* változót is
- Ezek viszont ritkább esetek
- A cél mindig az legyen, hogy bárki tudja használni az általatok írt osztályokat, anélkül hogy a forráskódba kellene néznie

- Mindez vonatkozik változókra és függvényekre is
- Létre lehet hozni egy *private* függvényt, és *public* változót is
- Ezek viszont ritkább esetek
- A cél mindig az legyen, hogy bárki tudja használni az általatok írt osztályokat, anélkül hogy a forráskódba kellene néznie
- Ezeket nyelvi szinten nem kötelező betartani, lehet minden adattagja egy osztálynak *public*, de ezzel maga alatt ássa az ember a fát
- Egy hasznos praktika, ha alsóvonással jelölitek a *private* adattagokat, nem kötelező, van aki szereti, van aki nem, a lényeg hogy ha egy jelölést már választottatok akkor használjátok azt, a kódotokban mindig legyenek egységesek a jelölések

Dinamikus memória foglalás javaban

```
public class Main {
    public static void main(String[] args) {
        Complex comp = new Complex(5, 6);
        System.out.println(comp.rePart_); // Hiba
        System.out.println(comp); // Nem hiba
    }
}
```

- *new* kulcsszóval
- Egyszerűbb mint C-ben (malloc, realloc, calloc)
- Cserébe mindennek dinamikusan kell memóriát foglalni
- Kivételek ez alól a primitív adatszerkezetek, mint az *int*, *float*
- Általában a primitívek kis betűvel kezdődnek, tehát pl a *String* nem az

```
VáltozoTípusa változóNév =
        new ObjektumTípusa(konstruktor bemenetei)
```

Konstruktor javaban

```
public Complex() {
    rePart_ = 0;
    imPart_ = 0;
}
public Complex(float rePart) {
    rePart_ = rePart;
    imPart_ = 0;
}
```

- Konstruktor a definíció és inicializálás egybevonása
- Nincs visszatérési értéke
- A paraméter nélküli konstruktor a *default konstruktor*
- Amikor *new*-val létrehozunk egy objektumot a megfelelő konstruktora hívódik meg
- Lehet *private* egy konstruktor, de ritkán van értelme

Javaban nincs destruktor

- Furcsa lehet ez azoknak akik tudnak C++-ban programozni, de javaban nincs destruktor
- De akkor mi van helyette?

Javaban nincs destruktor

- Furcsa lehet ez azoknak akik tudnak C++-ban programozni, de javaban nincs destruktor
- De akkor mi van helyette?
- Garbage collector
- Ez annyit tesz, hogy nem akkor szűnik meg egy objektum amikor bezárul a blokk ahol létre lett hozva, hanem amikor ezt a java jónak látja
- Természetesen olyan objektumokat nem szabadít fel amik használatban lehetnek még

Most egyben

```
public class Complex {
    private float rePart_;
    private float imPart_;

    public Complex() {
        rePart_ = 0;
        imPart_ = 0;
    }

    public Complex(float rePart) {
        rePart_ = rePart;
        imPart_ = 0;
    }
```

```
    public Complex(float rePart, float imPart) {
        rePart_ = rePart;
        imPart_ = imPart;
    }
    public Complex add(Complex other) {
        float rePart = this.rePart_ + other.rePart_;
        float imPart = this.imPart_ + other.imPart_;
        Complex retval = new Complex(rePart, imPart);
        return retval;
    }
} // Itt új osztály és ezzel új fájl kezdődik
public class Main {
    public static void main(String[] args) {
        Complex comp = new Complex(5, 6);
        System.out.println(comp.rePart_); // Hiba
        System.out.println(comp); // Nem hiba
    }
}
```

Bonyolultabb példa

```
public class ComplexVector {
    private Complex[] coords_;
    private int dimension_;

    public ComplexVector(int dimension) {
        dimension_ = dimension;
        coords_ = new Complex[dimension];
    }

    public ComplexVector(ComplexVector other) {
        this.coords_ = other.coords_.clone();
        this.dimension_ = other.dimension_;
    }
```

```
    public ComplexVector add(ComplexVector other) {
        ComplexVector retval =
                new ComplexVector(this.dimension_);
        for (int i = 0; i < retval.coords_.length; i++) {
            retval.coords_[i] =
                    this.coords_[i].add(other.coords_[i]);
        }
        return retval;
    }
}
```

- Itt egy *Complex* tömb az egyik adattag
- Ez nagyon gyakori lesz, az általatok írt osztályokban nagy részében szintén általatok írt osztályok lesznek az adattagok
- Ezért is nagyon fontos, hogy rejtett maradjon minden osztály működése
- Ebben az esetben, ha meg is változik az adatszerkezete vagy a számítási algoritmusa az egyiknek attól még a többivel való kapocs nem romlik el

Amiket kiemelnék az előbbi példából

```
private Complex[] coords_;
```

- Így lehet tömböt deklarálni (gyakon már volt)

```
coords_ = new Complex[dimension];
```

- Tömb létrehozása, ilyenkor a *default konstruktor* hívódik meg sokszor

```
retval.coords_.length;
```

- A tömböknek van egy *length* (hossz) adattagja, így nem is kellene a dimenzióját tárolni

Amiket kiemelnék az előbbi példából

```
private Complex[] coords_;
```

- Így lehet tömböt deklarálni (gyakon már volt)

```
coords_ = new Complex[dimension];
```

- Tömb létrehozása, ilyenkor a *default konstruktor* hívódik meg sokszor

```
retval.coords_.length;
```

- A tömböknek van egy *length* (hossz) adattagja, így nem is kellene a dimenzióját tárolni
- Ha precíz lettem volna, az összeadásnál vizsgálom, hogy egyáltalán össze lehet-e őket adni
- Ezt majd hibakezeléssel oldjuk meg, ami a később lesz csak