Java és web programozás

Kovács Kristóf, Rimay Zoé

Budapesti Műszaki Egyetem

2013. szeptember 18.

2. Előadás

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;
    }
}
```

Objektum

- Objektum a konkrét adat és a rajta végezhető műveletek
- Osztály egy típus, melyből létrehozhatunk konkrét példányokat, objektumokat.
- Az objektumoknak jogköre van
- Képesek kommunikálni más objektumokkal
- A belső adatszerkezete és a működését megvalósító algoritmus rejtve marad
- Könnyen módosítható, újrafelhasználható, általánosítható

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 öröklléssel foglalkozunk.

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.

```
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
    }
}
```

- Maga a *comp* objektum elérhető, így kiírható
- Viszont az adott *private* adattagja nem, mert rejtett

- 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 (másrészt mi se fogunk neki örülni)
- Egy hasznos praktika, ha alsóvonással jelölitek a *private* adattagokat

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 osztályok 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

```
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

```
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 következő heten lesz csak