Java

OOP alapok

Dr. Kiss Gábor

Csomagok, package

- Minden osztályt önálló fájlba teszünk.
- Egy összetettebb program több tucat osztályból épül fel.
- Az osztályok, illetve forrásfile-ok kategóriákba sorolását teszik lehetővé a csomagok.
- Az osztályok hierarchikus struktúrát alkotnak, pl. **java.lang**

Osztályok, objektumok a Java nyelvben

Osztály = séma:

objektumok reprezentációjának megadása

Objektum: egy osztály egy példánya

minden objektum valamilyen osztályból származik példányosítással

Reprezentáció:

példány adattagok, példány metódusok

Osztály: típus

```
public class Alkalmazott {
	String név;
	String beosztás;
	int fizetés;

	void fizetéstEmel(int mennyivel) {
		this.fizetés += mennyivel;
	}
}
```

```
public class Program {
	public static void main(String[] args) {
		Alkalmazott a;
		a = new Alkalmazott();
	}
}
```

Objektumok tárolása

- Dinamikus memóriakezelés szükséges
- Ada, C: mutatók (pointerek)
- Java: referenciák

```
Alkalmazott a;
```

Az `a` változóban az objektum memóriabeli címét tároljuk. A deklaráció hatására nem jön létre objektum!

Objektum létrehozása

- Az `a` változóhoz objektum hozzárendelése

```
a = new Alkalmazott();
```

- Példányosítás: valamilyen osztályból a **`new`** operátorral (memóriafoglalás a mellékhatás, a kezdőcím a kifejezés értéke)

```
new Alkalmazott()
```

- Az **`a`** referencia a **`new`** operátorral létrehozott “objektumra mutat”

Metódushívás

```
public class Alkalmazott {
	String név;
	String beosztás;
	int fizetés = 40000;
	void fizetéstEmel(int mennyivel) {
		this.fizetés += mennyivel;
	}
	void fizetéstkiír() {
		System.out.println(this.fizetés);
	}
}
```

```
public class Program {
	public static void main(String[] args) {
		Alkalmazott a;
		a = new Alkalmazott();
		a.fizetéstkiír();
		a.fizetéstEmel(30000);
		a.fizetéstkiír();
	}

}
```

Hozzáférési kategóriák (kiterjesztés (öröklés) nélkül)

- **public** (nyilvános): mindenki elérheti
- **üres** (félnyilvános): csomagon belül public, azon kívül private
- **private** (privát): csak az osztályon belül érhető el
- **protected** (védett): a kiterjesztéskor lesz szerepe

```
public class Alkalmazott {
	String név;
	String beosztás;
	int fizetés = 40000;
	void fizetéstEmel(int mennyivel) {
		this.fizetés += mennyivel;
	}
	public void fizetéstkiír() {
		System.out.println(this.fizetés);
	}
}
```

```
public class Program {
	public static void main(String[] args) {
		Alkalmazott a;
		a = new Alkalmazott();
		a.fizetéstkiír();
		a.fizetéstEmel(30000);
		a.fizetéstkiír();
	}

}
```

```
public class Alkalmazott {
	String név;
	String beosztás;
	int fizetés = 40000;
	void fizetéstEmel(int mennyivel) {
		this.fizetés += mennyivel;
	}
	public void fizetéstkiír() {
		System.out.println(this.fizetés);
	}
}
```

```
public class Program {
	public static void main(String[] args) {
		Alkalmazott a;
		a = new Alkalmazott();
		a.fizetés = 50000;
		a.fizetéstkiír();
		a.fizetéstEmel(30000);
		a.fizetéstkiír();
	}
}
```

```
public class Alkalmazott {
	String név;
	String beosztás;
	public int fizetés = 40000;
	void fizetéstEmel(int mennyivel) {
		this.fizetés += mennyivel;
	}
	public void fizetéstkiír() {
		System.out.println(this.fizetés);
	}
}
```

```
public class Program {
	public static void main(String[] args) {
		Alkalmazott a;
		a = new Alkalmazott();
		a.fizetés = 50000;
		a.fizetéstkiír();
		a.fizetéstEmel(30000);
		a.fizetéstkiír();
	}
}
```

```
public class Alkalmazott {
	String név;
	String beosztás;
	private int fizetés = 40000;
	void fizetéstEmel(int mennyivel) {
		this.fizetés += mennyivel;
	}
	public void fizetéstkiír() {
		System.out.println(this.fizetés);
	}
}
```

```
public class Program {
	public static void main(String[] args) {
		Alkalmazott a;
		a = new Alkalmazott();
		a.fizetés = 50000;
		a.fizetéstkiír();
		a.fizetéstEmel(30000);
		a.fizetéstkiír();
	}
}
```

```
public class Alkalmazott {
	String név;
	String beosztás;
	public int fizetés = 40000;
	void fizetéstEmel(int mennyivel) {
		this.fizetés += mennyivel;
	}
	private void fizetéstkiír() {
		System.out.println(this.fizetés);
	}
}
```

```
public class Program {
	public static void main(String[] args) {
		Alkalmazott a;
		a = new Alkalmazott();
		a.fizetés = 50000;
		a.fizetéstkiír();
		a.fizetéstEmel(30000);
		a.fizetéstkiír();
	}
}
```

Közvetlen adathozzáférés megtiltása

```
public class Alkalmazott {
	String név;
	String beosztás;
	int fizetés;
	public void fizetéstBeállít(int összeg) {
		this.fizetés = összeg;
	}
	public void fizetéstEmel(int mennyivel) {
		this.fizetés += mennyivel;
	}
	void fizetéstkiír() {
		System.out.println(this.fizetés);
	}
}
```

```
public class Program {
	public static void main(String[] args) {
		Alkalmazott a;
		a = new Alkalmazott();
		a.fizetéstBeállít(200000);
		a.fizetéstkiír();
		a.fizetéstEmel(30000);
		a.fizetéstkiír();
	}
}
```

Referenciák ráállítása egy objektumra

- Referencia és objektum együttes létrehozása

```
Alkalmazott a = new Alkalmazott();
```

- Referencia ráállítása meglévő objektumra

```
Alkalmazott b = a;
```

A két referencia ugyanarra az objektumra mutat.

```
b.fizetéstEmel(10000);
```

Üres referencia

- Ha egy változó értéke **`null`**, akkor nem mutat objektumra.

  `Alkalmazott c = null;`
- A **`null`** referencia minden osztályhoz használható.
- Példányváltozók automatikus inicializálásához ezt használja a Java
- **`c.fizetéstEmel(10000);`**

  futási idejű hiba:
  `NullPointerException`

Nem változtatható referencia

```
final Alkalmazott a = new Alkalmazott();
a.fizetéstBeállít(100000);
a = new Alkalmazott();
```

A referencia “konstans”, nem lehet másik objektumra állítani, de a mutatott objektum megváltozhat.

Az objektum élettartama

- Nincs olyan utasítás, amivel objektumot explicit módon meg lehet szüntetni
- A nyelv biztonságosságát növeli
- Szemétgyűjtés: ha már nem hivatkoznak egy objektumra, akkor azt meg lehet szüntetni.

```
Alkalmazott a = new Alkalmazott();
a = null;
```

- Nem biztos, hogy megszűnik a program vége előtt

Szemétgyűjtés

- Modern nyelvekben gyakori
- Biztonságosság
  - többszörös hivatkozás esetén: ha az egyik hivatkozáson keresztül megszüntetjük az objektumot, egy másikon keresztül meg továbbra is használni próbáljuk
- Hatékonyság: idő és tár
- Ciklikus hivatkozás
- Szemétgyűjtő algoritmusok

A this pszeudováltozó

- Az osztálydefiníción belül a példánymetódusokban this névvel hivatkozhatunk az aktuális objektumra.
- A static metódusokban a this persze nem használható.
- Ez egy predefinit név.
- Noha a this.valami-hez általában nem kell a minősítés, időnként azért szükség lehet rá. És olyan is van, amikor maga a this kell (pl. átadni paraméterként).

```
boolean kevesebbetKeresMint( Alkalmazott másik ){
     return másik.többetKeresMint(this);
}

public void fizetéstBeállít( int fizetés ){
     this.fizetés = fizetés;
}
```

Névütközések

- Példányváltozó és formális paraméter neve megegyezhet. Példa: előbb... ELFEDÉS
- Metódusnév és változónév megegyezhet, mert a () megkülönbözteti őket a hivatkozáskor.

```
int fizetés;
public int fizetés(){ return fizetés; }
```

Metódusnevek túlterhelése

- ugyanazt a nevet használhatom több metódushoz, ha különböző a szignatúra
  - szignatúra: név plussz paraméterek típusának sorozata
  - "metódusnév túlterhelése"
  - meghíváskor az aktuális paraméterek száma és (statikus) típusa alapján dönt a fordító (nem számít a visszatérési érték, mert anélkül is meg lehet egy metódust hívni)
  - valaminek illeszkednie kell, különben fordítási hiba

Példa

```
void fizetéstEmel( int növekmény ){
   fizetés += növekmény;
}

void fizetéstEmel(){ fizetés += 5000; }

void fizetéstEmel( Alkalmazott másik ){
   if (kevesebbetKeresMint(másik))
      fizetés = másik.fizetés;
}
```

```
a.fizetéstEmel(10000);
a.fizetéstEmel();
a.fizetéstEmel(b);
```

Példa

```
void fizetéstEmel( int növekmény ){
   fizetés += növekmény
}

void fizetéstEmel(){ fizetéstEmel(5000); }

void fizetéstEmel( Alkalmazott másik ){
   if (kevesebbetKeresMint(másik))
      fizetés = másik.fizetés;
}
```

```
a.fizetéstEmel(10000);
a.fizetéstEmel();
a.fizetéstEmel(b);
```

```
public class CalculatorTest {

    public static void main(String[] args) {

        Calculator myCalculator = new Calculator();

        int totalOne = myCalculator.sum(2, 3);
        System.out.println(totalOne);

        float totalTwo = myCalculator.sum(15.99F, 12.85F);
        System.out.println(totalTwo);

        float totalThree = myCalculator.sum(2, 12.85F);
        System.out.println(totalThree);
    }
}
```

```
public class Calculator {

    public int sum(int numberOne, int numberTwo) {
        System.out.println("Method One");
        return numberOne + numberTwo;
    }

    public float sum(float numberOne, float numberTwo) {
        System.out.println("Method Two");
        return numberOne + numberTwo;
    }

    public float sum(int numberOne, float numberTwo) {
        System.out.println("Method Three");
        return numberOne + numberTwo;
    }
}
```

```
public class ElevatorTest {
   public static void main(String args[]) {
    Elevator myElevator = new Elevator();

     myElevator.openDoor();
     myElevator.closeDoor();
     myElevator.goUp();
     myElevator.goUp();
     myElevator.goUp();
     myElevator.openDoor();
     myElevator.closeDoor();
     myElevator.goDown();
     myElevator.openDoor();
     myElevator.closeDoor();
     myElevator.goDown();

     myElevator.setFloor(myElevator.TOP_FLOOR);

     myElevator.openDoor();
   }
}
```

```
public class Elevator {

    public boolean doorOpen=false;
    public int currentFloor = 1;

    public final int TOP_FLOOR = 5;
    public final int BOTTOM_FLOOR = 1;

    public void openDoor() {
        System.out.println("Opening door.");
        doorOpen = true;
        System.out.println("Door is open.");
    }

    public void closeDoor() {
        System.out.println("Closing door.");
        doorOpen = false;
        System.out.println("Door is closed.");
    }
```

```
public void goUp() {
        if (checkDoorStatus()) { // Is door open?
            closeDoor();
        }
        System.out.println("Going up one floor.");
        currentFloor++;
        System.out.println("Floor: " + currentFloor);
    }

    public void goDown() {
        if (checkDoorStatus()) { // Is door open?
            closeDoor();
        }
        System.out.println("Going down one floor.");
        currentFloor--;
        System.out.println("Floor: " + currentFloor);
    }
```

```
    public void setFloor(int desiredFloor) {
        while (currentFloor != desiredFloor){
            if (currentFloor < desiredFloor){
                goUp();
            }
            else {
                goDown();
            }
        }
    }

    public int getFloor() {
    return currentFloor;
    }

    public boolean checkDoorStatus() {
    return doorOpen;
    }
}
```