Az előadás anyagának összeállításához komolyan támaszkodtam az alábbi Java könyvre: http://nagygusztav.hu/sites/default/files/csatol/java_programozas_1.3.pdf. Ajánlom mindenkinek, kiváló áttekintést ad azoknak, akik a Java fejlesztéssel szeretnének foglalkozni.

Java alapfogalmak:

osztály (class): Metódusok (methods), mezők (fields) és jellemzők (properties) egy egységbe zárt csoportja, ill. ennek (típus)deklarációja.

objektum (object): Az osztály egy példánya.

Vagy

Az **osztály** bizonyos fajta objektumok közös változóit és metódusait leíró tervrajz.

Az **objektum** változókból és kapcsolódó metódusokból felépített egység.

Az objektumok az objektumorientált technológia alapjai. Néhány példa a hétköznapi életből: kutya, asztal, tv, bicikli. Ezek a valódi objektumok két jellemzővel rendelkeznek: állapottal és viselkedéssel. Például a kutya állapotát a neve, színe, fajtája, éhessége stb. jellemzi, viselkedése az ugatás, evés, csaholás, farok-csóválás stb. lehet. A bicikli állapotát a sebességfokozat, a pillanatnyi sebesség, viselkedését a gyorsulás, fékezés, sebességváltás adhatja. A programbeli objektumok modelljei a valódi objektumoknak. Az objektum állapotát egy vagy több változóval, a viselkedését az objektumhoz rendelt metódussal (függvénnyel) írjuk le.

A valódi világban gyakran sok objektummal találkozunk ugyanabból a fajtából. Például a biciklink nagyon sok más biciklire jelentősen hasonlít. Az objektumorientált szóhasználatban azt mondjuk, hogy egy konkrét bicikli a biciklik osztályának egy példánya. A biciklik rendelkeznek állapottal (aktuális sebességfokozat, fordulatszám stb.) és viselkedéssel (sebességváltás, fékezés). Ennek ellenére minden bicikli konkrét állapota független az összes többi bicikli állapotától.

Minden ilyen objektum két fő jellemzővel rendelkezik:

- tulajdonságokkal (attribútum, adattag, változó) - pl. az autó színe, aktuális sebességfokozata, vagy az ember neve, testmagassága, aktuális tevékenysége
- viselkedési jellemzőkkel (metódus, módszer) - pl. az autó sebességet vált, fékez, ill. az ember beszél, dolgozik

Lássuk ezt a gyakorlatban:

```
class Bicycle {
```

```
	int cadence = 0;
	int speed = 0;
	int gear = 1;

	void changeCadence(int newValue) {
		 cadence = newValue;
	}

	void changeGear(int newValue) {
		gear = newValue;
		}

	void speedUp(int increment) {
		speed = speed + increment;
	}

	void applyBrakes(int decrement) {
		speed = speed - decrement;
	}
	void printStates() {
		system.out.println("cadence:"+cadence+"speed:"+speed+"gear:"+ge
	ar);
	}
}

class BicycleDemo {
	public static void main(String[] args) {
		// Create two different Bicycle objects
		Bicycle bike1 = new Bicycle();
		Bicycle bike2 = new Bicycle();

		// Invoke methods on those objects

		bike1.changeCadence(50);
		bike1.speedUp(10);
		bike1.changeGear(2);

		bike1.printStates();

		bike2.changeCadence(50);
		bike2.speedUp(10);
		bike2.changeGear(2);
		bike2.changeCadence(40);
		bike2.speedUp(10);
		bike2.changeGear(3);
		bike2.printStates();
	}
}
```

Egy objektumpéldány létrehozását az adott osztály konstruktora végzi el. Minden osztálynak van konstruktora. Ha nem hozunk létre mi, akkor maga a Java rendszer megteszi ezt helyettünk. A konstruktor hívása a new operátorral lehetséges:

```
Bicycle bike1 = new Bicycle();
```

Adatelérési szintek

Elérési szintek (láthatóság)				
Módosító	**Osztály**	**Csomag**	**Leszármazott**	**Összes osztály**
`public`	●	●	●	●
`protected`	●	●	●	●
módosító nélküli	●	●	●	●
`private`	●	●	●	●

final vagy static

Röviden: A static kulcsszóval megadott valami mindig az osztályhoz tartozik, nem az osztálypéldányokhoz, egy final valami pedig csakis egyszer definiálható, és nem módosítható.

A static kulcsszó 4 esetben használható

- static változó
- static metódus
- static kódrészlet
- static beágyazott osztály

static változó

- A statikus változó mindig a classhoz tartozik, nem az objektumhoz (instance-hoz)
- A statikus változók mindig csak egyszer inicializálódnak, a végrehajtás kezdetekor. Ilyenkor a static változók inicializálódnak először, és csak azután a többi.
- Egy változópéldányon osztozik az összes objektum az adott osztályból.
- A static változók elérhetőek közvetlenül az osztály nevével, és nem kell hozzá példányosított objektum.
- Syntax : `Class.variable`

static metódus

- A statikus metódus mindig a classhoz tartozik, nem az objektumhoz (instance-hoz)
- statikus metódus csakis a statikus változókat éri el. Nem érhet el példányosított változókat, kivéve, ha a statikus metódous hoz létre osztálypéldányt.
- statikus metódus csakis statikus metódusokat hívhat, kivéve az általa létrehozott osztálypéldány (objektum) non-static metódosait.
- A statikus metódusok elérhetőek közvetlenül az osztály nevével, és nem kell hozzá példányosított objektum.

- Syntax : Class.methodName()
- statikus metódus nem érhető el this vagy super kulcsszó használatával.

static osztály

A Java ismeri a statikus beágyazott osztály fogalmát, ilyenkor megadhatunk egy osztályt egy másik osztály tagjaként. Egy ilyen osztályt beágyazott osztálynak hívunk, és a következőképpen néz ki:

```
class EnclosingClass {
    ...
    static class StaticNestedClass {
        ...
    }
    class InnerClass {
        ...
    }
}
```

A beágyazott osztályokat arra használjuk, hogy kifejezzük és érvényesítsük két osztály között a kapcsolatot. Megadhatunk egy osztályt egy másik osztályon belül, hogyha a beágyazott osztálynak a magába foglaló osztályon belüli környezetben van értelme. Pl. a szövegkurzornak csak a szövegkomponensen belüli környezetben van értelme.

A beágyazó osztály tagjaként a beágyazott osztály kiváltságos helyzetben van. Korlátlan hozzáféréssel rendelkezik a beágyazó osztályok tagjaihoz még akkor is, hogy ha azok privátként vannak deklarálva. Azonban ez a speciális kiváltság nem mindig speciális. A hozzáférést biztosító tagok korlátozzák a hozzáféréseket az olyan osztálytagokhoz, amelyek a beágyazó osztályon kívül esnek. A beágyazott osztály a beágyazó osztályon belül található, ebből kifolyólag hozzáférhet a beágyazó osztály tagjaihoz.

Mint ahogyan más tagokat is, a beágyazott osztályokat is statikusként, avagy nem statikusként lehet deklarálni, ezért ezeket pontosan így is hívják: statikus beágyazott osztály. A nem statikus beágyazott osztályokat belső osztályoknak hívjuk.

Ahogy a statikus metódusok és változók esetén, amelyeket mi osztálymetódusoknak és változóknak hívunk, a statikus beágyazott osztályt a beágyazó osztályával kapcsoljuk össze. Ahogy az osztálymetódusok, a statikus beágyazott osztályok sem hivatkozhatnak közvetlenül olyan példányváltozókra vagy metódusokra, amely az ő beágyazó osztályában van megadva. A példánymetódusok és változók esetén egy belső osztály az ő beágyazó osztályának a példányával kapcsolódik össze, és közvetlen hozzáférése van annak az objektumnak a példányváltozóihoz és metódusaihoz. Mivel egy belső osztályt egy példánnyal társítanak, ezért önmaga nem definiálhat akármilyen statikus tagot.

Top level osztály sosem lehet statikus.

final class

final osztályból nem lehet származtatni. Nagyon sok Java alaposztály final, például `java.lang.System` és `java.lang.String.` Ennek oka a biztonságos kezelés

final metódus

Az ilyen metódusokat nem lehet felülírni (override). Ezzel elkerülhetőek a nem várt viselkedések a leszármaztatott osztályokban.

final változó

Egy final változó értékét nem lehet megváltoztatni az inicializálás után. Más nyelvekben ezt konstans változóknak is hívják.

Változók típusai

Egészek

Típus	Leírás	Méret/formátum
byte	bájt méretű egész	8-bit kettes komplemens
short	rövid egész	16-bit kettes komplemens
int	egész	32-bit kettes komplemens
long	hosszú egész	64-bit kettes komplemens

Valós számok

Típus	Leírás	Méret/formátum
float	egyszeres pontosságú lebegőpontos	32-bit IEEE 754
double	dupla pontosságú lebegőpontos	64-bit IEEE 754

Egyéb típusok

Típus	Leírás	Méret/formátum
char	karakter	16-bit Unicode karakter
boolean	logikai érték	true vagy false

Változónevek

A program változónevekkel azonosítja a változóértékeket. A Java nyelvben a következők érvényesek a nevekre:

- Valódi azonosító legyen, tetszőlegesen hosszú Unicode karaktersorozat, de az első karakter csak betű lehet.
- Nem lehet foglalt szó, logikai literál (true vagy false) vagy null.
- Egyedinek kell lenni az érvényességi tartományban, viszont más tartománybeli változóval megegyezhet.

Konvenció (tehát nem kötelező, de szokás), hogy a változóneveket kisbetűvel kezdjük, az osztályneveket pedig naggyal. Ha a név több szóból áll össze, a közbülső szavak kezdőbetűit mindig naggyal írjuk.

```
/*************************************************************/
/* Author: CS307 Course Staff                                */
/* Date: February 14, 2005                                   */
/* Description: Demos constructors, static vs instance methods,   */
/*              and method overloading.                      */
/*************************************************************/
public class DemoClass
{
    private int x;

    public DemoClass()
    {
        // assign default value
        x = 0;
    }

    public DemoClass(int x)
    {
        // use this.x to refer to the instance variable x
        // use x to refer to a local variable x (more specifically,
        // method parameter x)
        this.x = x;
    }

    public DemoClass(DemoClass otherDemo)
    {
        // copy the value from the otherDemo
        this.x = otherDemo.x;
    }

    // static method (aka class method)
    public static void s1() {
        return;
    }
    // instance method
    public void i1() {
        return;
    }

    // static calling static OK
    // static calling instance is a compile-time error
    public static void s2() {
//        i1();     // compile-time error
        s1();       // DemoClass.s1
        return;
    }

    // instance calling static OK
    // instance calling instance OK
    public void i2() {
        s1();       // DemoClass.s1();
        i1();       // this.i1();
        return;
    }

    // call various versions of overload() based on their
    // list of parameters (aka function signatures)
```

```
    public void overloadTester() {
        System.out.println("overloadTester:\n");

        overload((byte)1);
        overload((short)1);
        overload(1);
        overload(1L);
        overload(1.0f);
        overload(1.0);
        overload('1');
        overload(true);
    }

    public void overload(byte b) {
        System.out.println("byte");
    }
    public void overload(short s) {
        System.out.println("short");
    }
    public void overload(int i) {
        System.out.println("int");
    }
    public void overload(long l) {
        System.out.println("long");
    }
    public void overload(float f) {
        System.out.println("float");
    }
    public void overload(double d) {
        System.out.println("double");
    }
    public void overload(char c) {
        System.out.println("char");
    }
    public void overload(boolean b) {
        System.out.println("boolean");
    }

    public static void main(String[] args) {
        DemoClass dc = new DemoClass();
        dc.overloadTester();
    }
}

// end of DemoClass.java
```