**Osztályok felépítése, elemei. Osztály definiálás szabályai alapelvei.**

Az osztály az objektum-orientált programozás egyik központi fogalma. Az adattartalmukban esetleg eltérő, de viselkedés szempontjából megegyező objektumokat tekintjük egy osztályba tartozóknak.

Az objektum-orientált programozási nyelvek rendszerint jelzik is az egyes osztályokat. Ennek kulcsszava a legtöbb esetben a *class*.

Általános gyakorlat, hogy a programozási feladat elemeit hasonló tulajdonságaik alapján osztályokba soroljuk, és ezeket feleltetjük meg a programozási nyelv osztályainak. A nyelvi osztályokat felruházzuk a probléma szempontjából lényeges jellemzőkkel, és műveletekkel, majd a műveletek tartalmát leírjuk a célnyelven, megalkotva ezzel azok törzsét. Miután a lényeges elemek osztályai elkészültek, a programozási feladat gyakorlatilag a természetes nyelvhez közeli formában fogalmazható meg a célnyelven. A programozási nyelvek osztályai általában tartalmazzák az **adattagokat** (mező), a **metódusokat**, **konstruktort** ,az **inicializáló** **blokkokat** és **tagosztály** tartalmaznak.

Az adattagok felelősek az adott objektum információinak tárolásáért, a metódusok pedig az osztály által engedélyezett műveletek megvalósulásai. Bizonyos nyelveken ezen túl feltüntethető az ősosztály, a tagok elérhetőségi módosítói, osztályfüggvények leírása, jellemzők (property-k) meghatározása, esetleg további nyelvi elemek.

Gyakori eset, hogy egyes osztályok más osztályok speciális fajtáinak, vagy általánosításának felelnek meg. Az is előfordul, hogy egy osztály viselkedését egy másik osztály segítségével tudjuk könnyen leírni (megvalósítani). Ilyen és ehhez hasonló esetekre nyújt megoldást az öröklődés használata. Ha egy osztály egy másikból származik, akkor azt leszármazott-, a másikat pedig ősosztálynak hívjuk, kulcsszava lehet extends, inherits, vagy valami hasonló.

A leszármazott támaszkodhat az ősosztály műveleteire, virtuális tagok esetében felül is definiálhatja (override) azokat.

A nyelvek nagy része csak egy ősosztályt engedélyez az osztályok számára, de ellenpélda is akad.

Az osztályok öröklődési kapcsolatai tehát általában fával, vagy gráffal írhatók le, amit osztály-hierarchiának nevezünk. A többszörös öröklődés elkerülésére számos nyelv alkalmazza az interfész (interface) fogalmát.

Az interfész olyan osztályként fogható fel, aminek sem adattartalma, sem műveleteinek konkrét megvalósítása nincs. Gyakorlatilag csak egy leírás arról, hogy milyen lehetőségeket várunk el egy olyan osztálytól ami megvalósítja (implements) az interfészt. Általában megengedett, hogy egy osztály több interfészt is megvalósítson.

Osztálydefiníció:

Osztálynév: tetszőleges nagy kezdőbetűs azonosító.

**Módosítók**:

– Hozzáférés módosító: public (az osztály bármely más osztályban használható)

– Tagosztály (beágyazott osztály): static

– Végleges osztály (nem lehet leszármaztatni): final Pl. java.lang.System, java.lang.String

– Absztrakt (nem példányosítható, de leszármaztatható) osztály: abstract

**Adattag definíció:**

Hasonlít egy lokális változó definíciójához, de lehetnek előtte módosítók:

’módosítók’ ’típus’ ’azonosítólista’

Módosítók:

– Hozzáférés módosítók: public, protected, private

– Futásidőbeli viselkedést szabályozó módosítók: transient, volatile

– Konstans mező: final

– Osztályszintű adattag, az osztály minden példányában ugyanaz az értéke: static

Típus: tetszőleges elemi, vagy referencia típus.

Azonosítólista: azonosítók vesszővel elválasztott listája.

A default hozzáférési módosítóval rendelkező adattagok csak az ugyanabban a csomagban definiált osztályok számára érhetők el. A lokális változókkal ellentétben az adattagokat különkülön szokás definiálni. A lokális változókkal ellentétben az adattagok inicializálódnak, ha a programozó nem inicializálja őket.

**Metódus definíció:**

Módosítók:

– Hozzáférés módosítók: uaz mint az adattagoknál

– Végleges metódus (nem írható felül a leszármazott osztályokban): final

– Absztrakt osztály metódusa: abstract

– Osztályszintű metódus, nem kell példányosítani az osztályt hogy meghívható legyen: static (pl. a main metódus)

Típus: tetszőleges elemi vagy referencia típus, ill. void

Név: kisbetűvel kezdődő azonosító (konvenció: cselekvést fejez ki) (pl. increaseRadius(), getArea();)

Paraméterlista (egyenlőre a C nyelvi szintaktikát használjuk);

Látható az egységbezárás alapelvének érvényesülése:Egy nyelvi elemen belül találhatók az adatok és az adatokon műveleteket végző metódusok.   
Az osztályon belül az elemek (csak) a nevükkel hivatkoznak egymásra. Az osztályon belül a metódusok korlátlanul használhatják az adattagokat és a többi metódust.

Az osztály, amit definiáltunk egy típus. A deklaráció során azonban nem jön létre ilyen típusú objektum (osztálypéldány), hanem referencia változók lesznek.   
Az osztály példányait a new Osztálynév() kifejezéssel hozhatjuk létre. Pl.: new Circle()  
A példányosítás kifejezés kiértékelésének eredménye egy adott típusú referencia, amely a létrejött objektumra mutat (hivatkozik). Ez a referencia eltárolható referencia változókban.

Referencia változó deklarálásakor csak a referencia számára foglalódik hely, nem jön létre objektum (az objektumot példányosítással hozzuk létre). Vagyis a pl :Circle c; utasítás hatására csak a referencia számára foglalódik hely, a Circle típusú objektum számára NEM. Referencia változó értéke egy cím, amely egy objektumot címez meg, vagy null. Hasonlít a pointerre, de a referencia változóban tárolt cím a programozó számára hozzáférhetetlen. A referencián keresztül hivatkozhatunk a hivatkozott objektum tagjaira (nem kell külön operátor hozzá). A referenciákon értelmezett többi művelet nem a hivatkozott objektumra vonatkozik, hanem magára a referenciára.

Referenciákon értelmezett műveletek:

**A hivatkozott objektum tagjaira hivatkozás (.):**

Ha a referencia értéke null, azaz c1 még nem tartalmaz obj. hivatkozást, akkor az objektum tagjaira való hivatkozás futási hibát okoz.

Azonos hatáskörben, osztályon belül csak a névvel hivatkozunk az osztály tagjaira, minden más esetben az objektum nevével minősítve.

Tagok hozzáférési kategóriái

Az információrejtés alapelv megvalósításának eszköze a hozzáférési módosítók használata.

Minden osztály tagra (adattag, metódus) létezik:

– public: bármelyik osztály számára elérhető

– protected: csak a leszármazott osztályok, valamint a saját csomagjában levő osztályok számára elérhető

– private: csak a saját osztályának definíciójában elérhető

– default (nincs megadva hozzáférési módosító): csomagszintű láthatóság; csak az azonos csomagban levő osztályok számára elérhető.

Információrejtés alapelv alkalmazása:

– Az adattagok private módosítót kapjanak.

– A metódusok közül csak az legyen public, amelyen keresztül engedélyezzük az osztály adattagjainak elérését (módosítását) a többi osztály számára.

Getter metódusok Az olyan metódusokat, amelyek egy adott jellemző értékét adják vissza, getter metódusoknak nevezzük. Figyelem! Egy adott jellemző mögött nem feltétlenül áll adattag.

Setter metódusok Az olyan metódusokat, amelyek egy adott jellemző értékét állítják be, setter metódusoknak nevezzük. A setter metódusban :

– az adattag módosítása előtt végezhetünk ellenőrzést

– a megváltozott jellemzőtől függő egyéb tulajdonságokat is módosíthatunk.

**Konstrukor:**

Konstruktor: speciális metódus, amelyet nem lehet explicite meghívni, hanem automatikusan hívódik meg az osztály példányosításakor.

Szerepe az objektum induló állapotának beállítása (pl. adattagok inicializálása).

Nem tekintjük metódusnak, mert nincs visszatérési típusa és nem lehet meghívni.

Szintaktikája:

módosító azonosító(paraméterlista) { utasítások }

Módosító: csak hozzáférési módosítója lehet (public, private, protected)

Azonosító: kötelezően az osztály neve

Visszatérési érték típusa: nincs

Paraméterlista: szintaktikája uaz mint a metódusnál;

a paraméterek példányosításkor kapnak értéket

Utasítások: return kivételével tetszőleges Java utasítás

Ha egy osztály nem éri el egy másik osztály konstruktorát, akkor nem példányosíthatja. Tehát private kontruktor esetén az osztály csak saját magát tudja példányosítani.

Konstruktor definiálása nem kötelező, de a fordító automatikusan készít egy default konstruktort, amelynek nincs paramétere és az utasítástörzse is üres. public Osztálynév();

Ha készítünk konstruktort, akkor nem jön létre default konstruktor.

Az előző kettőből következik, hogy egy osztálynak mindig van konstruktora. Ha írunk akkor az amit írtunk, ha nem akkor a default konstruktor. Példányosításkor csak olyan paramétereket adhatok meg, amilyeneket a konstruktor vár. Ha például az Empolyee osztály konstruktora egy String-et vár, akkor nem lehet paraméter nélkül, vagy más típusú paraméterrel példányosítani. Egy osztálynak lehet több konstruktora különböző paraméterlistával (metódus túlterhelés, overloading). Értelme: többféle inicializálási lehetőség biztosítása. Ha egy osztálynak több konstruktora is van, akkor a konstruktor első utasításaként meg lehet hívni az osztály egy másik konstruktorát a this(paraméterek); hívással. Értelme: az egyik konstruktorba már leírt inicializálásokat ne kelljen átmásolni a másikba. A konstruktorban tetszőleges utasítások lehetnek (nemcsak inicializálás). DE! Iinicializálására szokás használni. Az inicializáló utasítások tetszőleges számú adattagot érinthetnek. DE! A konvenció az, hogy az összes adattagot inicializáljuk.

Példányosítás: Lépései:

1. Létrejön az objektum (lefoglalódik számára a hely a memóriában).

2. Inicializálódnak az adattagok:  
 Definícióban megadott inicializálás

Automatikus inicializálás (ha a definícióban nem volt)

Inicializáló blokkban megadott inicializálás (később)

3. Lefut a megfelelő konstruktor.

4. Visszatér az objektumra mutató referenciával.

Objektum élettartam:

Példányosításkor jön létre.

Mikor szűnik meg? Automatikus szemétgyűjtő (garbage collector): a JVM indítja a programmal párhuzamosan. Minden objektumról nyilvántartja, hogy hány referencia hivatkozik rá. Amelyik objektumra egy sem, azt megszünteti.

Egy referencia objektumra hivatkozás megszűnik, ha pl. – megszűnik a változó, vagy – megváltozik a változó értéke (más objektumra való hivatkozást, vagy null-t kap értékül).