Asztali alkalmazások fejlesztése - Java - 04. óra

September 15

2022

Obejktum Orientált Programozás

Feladatlap

Tartalom

Objektum Orientált Programozás

A valós világ szemléletesebb leírása. Egy Objektum Orientált Program egymással kommunikáló objektumok összessége, melyben minden objektumnak megvan a jól meghatározott feladata.

Ha Objektum Orientált módon fejlesszük a programunkat, akkor következő szoftverminőségi előnyei lesznek:

- Helyesség
- Hibatűrés
- Karbantarthatóság, bővíthetőség
- Újrafelhasználhatóság
- Felhasználó-barátság
- Hordozhatóság
- Hatékonyság
- Ellenőrizhetőség
- Szabványosság
- Integritás (sérthetetlenség)

Osztály	Objektum
Egy felhasználó által készített típus, mely összetett adatszerkezet - tartalmazza az adott objektum adatait, és az azokat kezelő metódusokat.	Egy változó, melynek típusa valamely objektumosztály, vagyis az osztály egy példánya.
Autó osztály	Auto a8 = new Auto(”audi”,”a8”,2997,285,2022,”fekete”);

Egy Objektum Orientált Programozási nyelvet három fontos dolog jellemez

Encapsulation - Egységbezárás

Az adatok és a hozzájuk tartozó eljárásokat egyetlen egységben kezeljük (objektum-osztály)
Az osztály adattagjait (mezőit) tárolják az információkat, a metódusok kommunikálnak a külvilággal
Az osztály változóit csak a metódusokon keresztül változtathatjuk meg

Inheritance - Öröklés

Az objektum-osztályok továbbfejlesztésének lehetősége. Ennek során a származtatott osztály örökli ősétől azok attribútumait, és metódusait, de ezeket bizonyos szabályok mellett újakkal egészítheti ki, és meg is változtathatja a régieket.
Az eredeti osztályt ősosztálynak nevezzük (szülő), az új, továbbfejlesztett osztályt származtatott osztálynak (gyerek)

Egy ősből több leszármaztatott osztályt is készíthetünk

Polymorphism - Sokalakúság

Ugyanarra a kérelemre a különböző objektumok különbözőképpen reagálnak.
A származtatás során az ős osztályok metódusai képesek legyenek az új átdefiniált metódusok használatára újraírás nélkül is.
Ezt virtuális metódusokon keresztül érhetjük el.

Konstruktor

Speciális feladatú metódus, amely az objektum-példány alaphelyzetbe állítását végzi el (ez általában a mezők kezdőérték-beállítását jelenti). Az osztályok példányosításakor kötelező meghívni a konstruktort! Ez a szabály biztosítja, hogy csak már inicializált példánnyal lehessen dolgozni!

Destruktor

Azt a függvényt, amelyik az osztály megszűnésekor szükséges feladatokat elvégzi destruktornak vagy destruktor függvénynek nevezzük. A destruktor neve is az osztály nevével azonos, csak az elején ki kell egészíteni a ~

karakterrel. A destruktor meghívása automatikusan történik, és miután az objektumot nem használjuk, a keretrendszer automatikusan lefuttatja, és a felszabaduló memóriát visszaadja az operációs rendszernek.

Konstruktornak és destruktornak nem lehet visszaadott értéke. A destruktornak mindig publikus osztálymezőnek kell lennie.

Garbage Collection
A destruktor automatikus meghívásának a folyamatát szemétgyűjtési algoritmusnak nevezzük (garbage collection, GC).

A this mutató
Minden osztályhoz automatikusan létrejön egy mutató, aminek a neve **this**, és az éppen aktuális osztályra mutat.

Így, ha egy osztályfüggvényt meghívunk, amely valamilyen módon például a privát változókra hat, akkor az a függvény a this mutatón keresztül tudja, hogy mely aktuális privát mezők is tartoznak az objektumhoz.

	Public	**Private**	**Protected**
Adattag, Attribútum	Olyan attribútumok, melyek jellegüknél fogva nem igényelnek speciális szabályozást, azok megváltoztatása nem okoz, nem okozhat problémát az objektum működésében.	A külvilág nem férhet ezen attribútumukhoz hozzá. Ezek általában segédváltozók, segédmezők.	Olyan attribútumok, melyekhez a távoli külvilág nem férhet hozzá (számára private), de a közeli külvilág, leszármazott osztályok metódusai hozzáférhetnek (számára public)
Metódus	Olyan metódusok, amiket a távoli külvilág meghívhat.	A metódusokat a külvilág nem hívhatja meg, csak az adott osztály metódusai hívhatják meg	Olyan metódusok, melyeket a távoli külvilág nem hívhat meg (számára private), de a közeli külvilág, a leszármazott osztályok metódusaiból meghívhatóak (számukra public)

OOP - Osztály, Objektum

01. Feladat – Pont osztály

- Készítsd el a Pont osztályt, ami egy síkbeli pontot reprezentál.
- Legyen két privát (osztályon kívülről) adattagja a két koordinátájának (X és Y) tárolására.
- Készíts egy paraméter nélküli konstruktort, ami origo (0,0) középponttal inicializálja majd a Pont objektumot.
- Készíts egy két paraméteres konstruktort, ami a megadott értékekkel inicializálja a pontot.
- Készíts egy egy paraméteres konstruktort, ami egy N megadott érték alapján véletlenszerűen inicializálja a pontot. Az x és y koordináta véletszerűen [-N..N] közötti értéket vegyen fel.
- Készítsünk metódusokat a koordináták lekérdezéséhez és módosításához.
- Készíts el a Pont megjelenítésért felelős ToString metódust, amely [x;y] alakban fogja megjeleníteni a Pont objektumot.
- Készíts metódust, amely a Pont origo-tól vett távolságát határozza meg.
- Készíts metódust, amely két Pont távolságát határozza meg.

- Készíts metódust, amely meghatározza, hogy a pont melyik síknegyedbe esik. (1: bal felső, 2: bal alsó, 3: jobb alsó, 4: jobb felső)

02. Feladat – Kör

- Készítsen egy kört reprezentáló osztályt. Lehessen inicializálni a középpont koordinátáinak és a sugárnak a megadásával, valamint csak a sugár megadásával (ilyenkor origó középpontú), valamint véletlenszerűen.
- Határozza meg a Kör kerületét.
- Határozza meg a Kör területét.
- Készítsen metódust, amelynek a segítségével egy kört lehet nagyítani, vagy kicsinyíteni.
- Lehessen megkapni a kör adatait egy stringben (x,y,r) alakban.
- Teszteléshez hozzon létre 10 példányt az osztályból, és írja ki őket a képernyőre.
- Készítsen programot, ami megkeresi a legnagyobb területtel rendelkező kört.

03. Feladat – EmberekOOP - Alaposztály

- Hozz létre egy új java konzol alkalmazást, **EmberekOOP** néven!
- Adj a projekthez egy új osztályt, **Ember** néven!
- Az osztályban definiálj 3 privát, String típusú adattagot: **nev**, **szulDatum**, **szulHely**!
- Írj az osztályban paraméteres konstruktort, amellyel létre tudunk hozni, és kezdőértékkel tudunk ellátni Ember típusú objektumpéldányokat!
- Írj egyszerű **ToString** metódust, amely Stringgé alakítja az objektumot!
- Teszteld az osztályt a Main metódusából!
- Hozz létre három különböző objektumpéldányt, és írasd ki azokat!
- Írd át az Ember osztály **ToString** metódusát úgy, hogy a kimenete tagolt legyen.

04. Feladat – EmberekOOP - Továbbfejlesztés

- Írj publikus metódust az Ember osztályban, amely visszaadja a születési évet, egész típusú értékként! A szulDatum adattag első 4 karakterét int típusúvá kell konvertálnunk! Teszteld a metódust a Main metódusában!
- Írasd ki mindhárom születési évet egymás mellé! Írj az előzőhöz hasonló módszerrel két újabb metódust, amely egész értékként visszaadja a születési hónapot, és napot!
- Írj metódust az Ember osztályban, amely a születési év és az aktuális év alapján meghatározza az életkort!

05. Feladat – EmberekOOP – Emberek osztály

- Készíts újabb osztályt, **Emberek** néven, amely segítségével több Ember objektumpéldányt is tárolhatunk! Az új osztály neve Emberek legyen.
- Vegyél fel egy List<Ember> típusú privát adattagot! Egy listát, amelyhez tetszőleges darabszámú Ember objektumot adhatsz!
- Írj konstruktort az Emberek osztályban, amely konstans tömbök alapján feltölti a listát! Legalább 5 különböző ember adatait vedd fel!
- Írj ToString metódust az Emberek osztályban! A metódus bejárja a teljes listát, és az abban található minden Ember objektum ToStringjét egybefűzi, sortöréssel!
- Példányosítsd az Emberek osztályt, és írasd ki (a Mainből)