12. Tétel  
OOP

Az "OOP" rövidítés az "Objektum-Orientált Programozás" kifejezést jelenti. Az objektum-orientált programozás egy programozási paradigma, amely az adatok és a műveletek összekapcsolására, illetve azok hierarchikus rendezésére összpontosít. Az OOP az objektumokat és azok közötti kapcsolatokat használja a programozási feladatok modellezésére és megoldására.

Az OOP alapelvei közé tartozik az öröklődés, a polimorfizmus, az egységbezárás és az absztrakció. Ezek az elvek lehetővé teszik a kód újra felhasználhatóságát, rugalmasságát, karbantarthatóságát és könnyű kiterjeszthetőségét. Az OOP segítségével a programozók a valós világot modellezhetik és a problémákat objektumokra és azok közötti kapcsolatokra bontva oldhatják meg.

Az OOP-t számos programozási nyelv alapja, mint például Java, vagy a C#, de van olyan nyelv is, amiben csak opcionális, mint például a C++, JavaScript vagy a Python. Az objektum-orientált programozás népszerűségét a nagyobb projektek kódjainak strukturális szervezése és az újra felhasználhatósága adják.

# Osztály

Az osztály egy sablon, amely leírja, hogy az objektumok milyen adatmezőkkel és műveletekkel rendelkeznek. Az osztály definiálja az objektumok közös tulajdonságait és viselkedését is. Az objektumok az osztály alapján kerülnek létrehozásra, és az osztály az objektumok tulajdonságainak és viselkedésének absztrakt reprezentációja. Az osztály azonban nem minden esetben definiálja az adatmezők értékeit, melyek általában egyediek az egyes osztálypéldányokhoz.

# Objektum

Az objektum az OOP (Objektum-Orientált Programozás) alapvető építőköve, amely egy konkrét adatstruktúrát és az azt kezelő műveleteket egyesíti. Az objektum egy példányosított változó, amely adatait és metódusait (vagy függvényeit) tartalmazza. Az objektumokat gyakran valós világbeli entitásoknak megfelelően modellezzük, például autó, cég, vagy felhasználó lehet egy objektum.

Java-ban az objektumoknak több része van, amelyek közül néhány alapvető részletet szeretnék kiemelni:

1. **Állapot (State)**

Az objektum állapotát az adattagok (változók) jelentik. Ezek az adattagok az objektum tulajdonságait tárolják. Az állapot az objektum aktuális állapota vagy értéke, amely meghatározza annak viselkedését és jellemzőit.

1. **Viselkedés (Behavior)**

Az objektum viselkedését a metódusok határozzák meg. Ezek a függvények végzik el az objektumon belüli műveleteket vagy manipulálják az állapotát. A viselkedés az objektum által elérhető funkciókat vagy tevékenységeket jelenti.

1. **Azonosító (Identity)**

Minden objektum egyedi azonosítóval rendelkezik a JVM (Java Virtual Machine) számára. Ez az azonosító garantálja, hogy két különböző objektum soha nem lehet ugyanaz az objektum, még akkor sem, ha az állapotuk megegyezik.

1. **Osztály (Class)**

Az osztály a Java-ban az objektumok sablonja. Az osztály definiálja az objektumok állapotát és viselkedését, valamint meghatározza azokat a metódusokat és adattagokat, amelyek az objektumokon érhetők el. Az osztály egy absztrakt fogalom, amelyet az objektumok példányosításával hozunk létre.

1. **Konstruktor (Constructor)**

A konstruktor egy speciális metódus az osztályban, amelyet az objektum létrehozásakor hívódik meg automatikusan. A konstruktor felelős az objektum inicializálásáért, azaz az állapotának kezdeti értékekkel történő feltöltéséért.

1. **Példányváltozók (Instance Variables)**

Ezek az adattagok az objektum egyedi állapotát tárolják. Minden egyes objektum példány tartalmaz egy ilyen definíciót, és azok értékei különbözőek lehetnek az egyes példányok között.

Ezek az alapvető részek alkotják az objektumokat és osztályokat. Ezek segítségével lehetőségünk van a Java nyelven objektum-orientált módon modellezni és implementálni az adatokat és a műveleteket.

Absztrakció

Az absztrakció a komplexitás egyszerűsítésének folyamata a felhasználó részére, amely során olyan osztályokat modellezünk, amelyek megfelelnek a problémának, kellő részletességgel. Ez magában foglalja egy objektum vagy rendszer lényeges jellemzőinek azonosítását és modellezését, a lényegtelen részletek figyelmen kívül hagyásával. A programozásban az absztrakció elérése absztrakt adattípusok és interfészek segítségével történik. Lehetővé teszi a programozók számára, hogy az egy olyan fő osztályt csináljanak, amely több rendszerbe is beépül, melyre tudnak több logikát, metódust és osztályváltozókat deklarálni. Ez elősegíti a kód újra felhasználhatóságát, karbantarthatóságát és skálázhatóságát.

## Absztrakció – interfész

Az interfész a programozásban egy olyan szerkezeti elem, amely meghatározza, hogy egy osztály vagy egy objektum milyen metódusokat és tulajdonságokat kell, hogy tartalmazzon. Az interfészben deklarált metódusoknak nincs tényleges implementációja, ezeket az implementáló osztályokban kell megvalósítani.

# Egységbe zárás

Az egységbe zárás az adattagok (változók) és a rajtuk műveleteket végrehajtó metódusok összevonására utal, az objektum egyes összetevőihöz való közvetlen hozzáférés korlátozásának érdekében. Ezeket az adattagokat eldönthetjük, hogy elérhetővé szeretnénk-e tenni egyéb objektumoknak, akár olvasásra, akár manipulálásra. Az egységbezárás biztosítja az adatbiztonságot és megakadályozza az objektum belső állapotának közvetlen hozzáférését, lehetővé téve az adatintegritás jobb kontrollját és a nem kívánt módosítások kockázatának csökkentését.

# Polimorfizmus - többalakúság

A polimorfizmus lehetővé teszi a programozók számára, hogy ugyanazt a kódot használják különböző adattípusokkal, ami növeli a kód újra felhasználhatóságát és rugalmasságát. A polimorfizmus az objektumok olyan képességét jelenti, hogy különböző formákat vagy viselkedéseket vegyenek fel a használati kontextusuktól függően. Két fő típusa van: fordítási idejű (statikus) polimorfizmus és futási idejű (dinamikus) polimorfizmus.

A fordítási idejű polimorfizmust a metódus paraméterek túlterhelésével érjük el, ahol több metódus vagy operátor ugyanazzal a névvel rendelkezik, azonban különböző paraméterekkel.

A futási idejű polimorfizmus pedig a metódus felüldefiniálással érhető el, ahol egy leszármazott osztály biztosítja egy ősosztályban definiált metódus specifikus implementációját.

Van olyan nyelv, mint például a NodeJS, ami a fordítás idejű polimorfizmust nem tudja kezelni.

# Öröklődés

Az öröklődés egy mechanizmus az objektumorientált programozásban, amely lehetővé teszi egy osztály (leszármazott vagy származtatott osztály) számára egy másik osztály (ős vagy alap) tulajdonságainak és viselkedésének öröklését. A leszármazott osztály kiterjesztheti vagy módosíthatja az örökölt tulajdonságokat és viselkedést, valamint bevezetheti saját egyedi tulajdonságait és viselkedését. Az öröklődés elősegíti a kódújrafelhasználást, mivel a közös attribútumokat és metódusokat az ősosztályban lehet definiálni, és megoszthatók több leszármazott osztály között. Emellett lehetővé teszi hierarchikus kapcsolatok létrehozását az osztályok között, ami strukturáltabbá és modulárisabbá teszi a kódot.

A "gyémánt probléma" olyan helyzetet jelent az objektum-orientált programozásban, amikor egy adott osztály örököl két másik osztályt, amelyek pedig ugyanazt az osztálytól örökölt osztályt tartalmazzák. Ez a helyzet egy hierarchiában úgy néz ki, mint egy gyémánt alakú struktúra, ahol az ősosztálytól (felül) két út vezet az azonos leszármazott osztályhoz (alul). Ez a jelenség néha problémákat okozhat az öröklés rendszerében és az osztályok közötti kapcsolatokban. Pontosan ezért sok OOP nyelvben le van tiltva a több osztály öröklése, azonban ez lehetséges C++-ban és Java-ban. C#-ban például ez Interface-ekkel valósítható meg