3. Az objektumorientált paradigma alapfogalmai. Osztály, objektum, példányosítás. Öröklődés, osztályhierarchia. Polimorfizmus, metódustúlterhelés. A bezárási eszközrendszer. Absztrakt osztályok és interfészek. Típustagok.

Programozási paradigma (PP): Programozási mód. Alapvetően a program felépítésére használt eszközkészletet jelenti, vagyis milyen egységek képezik a program alkotóelemeit. (moduláris programozás, objektumorientált programozás, általánosított programozás, aspektusorientált programozás stb.)

Objektumorientált paradigma (OOP): Minden entitás osztályokkal van leírva/megadva. Az objektumok az osztályoknak példányai, minden objektum egységbe zár állapotokat és viselkedéseket. Állapotok alatt a mezőket, attribútumokat értjük, viselkedés alatt pedig azt, hogy mit végzünk el az adott objektum állapotán metódusokkal. Az objektumok kapcsolatot létesíthetnek egymással üzenetek átadásával.

Egységbezárás (Encapsulation): Az olyan részletek elrejtése, amelyekre csak az adott osztálynak van szüksége. Egy egyszerű, tiszta interfészt biztosítása.

Osztály: Az osztály egy felhasználói típus, amelynek alapján példányok (objektumok) hozhatók létre. Az osztály alapvetően adat és metódus (művelet) definíciókat tartalmaz.

Objektum: Információt (adatokat) tárol és kérésre műveleteket végez. Van állapota, viselkedése és futásidőben azonosítható.

Példányostás: Egy adott sablonnal meghatározott adatszerkezet műveletvégzésre alkalmas példányba történő lemásolása, létrehozása.

Öröklődés: Egy olyan mechanizmus, amely alkalmazásával egy osztályt (gyerekosztály) származtathatunk egy másikból. (szülő osztály) A modernebb programozási nyelvekben a többszörös öröklődésre nincsen lehetőségünk, hogy elkerüljük az abból adódó problémákat. (Pl.: Ha mindkét ősosztály tartalmaz ugyanolyan névvel ellátott metódust, akkor nem tudni, hogy melyik ősosztályból lesz végül örökölve) Ezáltal egy **osztályhierarchiát** hozhatunk létre, amelyben az osztályok megosztanak egymással bizonyos attribútumokat és metódusokat. Az öröklődésnek köszönhetően a kód olvashatóbb marad és könnyedén újra felhasználható.

Osztályhierarchia: Egy osztályhierarchia alatt azt értjük, hogy van egy ősosztály (superclass), amelyből minden későbbi osztály örököl tulajdonságokat. Az ősosztály olyan alapvető tulajdonságokkal rendelkezik, amelyre minden későbbi osztálynak szüksége lesz. Az ősosztályból örökölt osztályból is örökölhet később új osztály. Minél mélyebben helyezkedik el egy osztály az osztályhierarchiában, annál speciálisabb viselkedéssel/metódusokkal/adattagokkal bír.

Polimorfizmus: Többalakúság. Egy típuselméleti fogalom, amely szerint egy ősosztály típusú változó hivatkozhat ugyanazon közös ősosztályból származó (vagy ugyanazon interfészt megvalósító) osztályok példányaira. A polimorfizmus egy egységes interfészre utal, amit különböző típusok valósítanak meg. A polimorf típuson végzett műveletek több különböző típus értékeire alkalmazhatók. Polimorfizmus többféleképpen is megvalósítható:

- Ad hoc polimorfizmus: Egy függvénynek sok különböző implementációja van, amelyeket egyenként specifikálnak néhány különböző típus és kombinációja számára. Megvalósítható túlterheléssel.

- Paraméteres polimorfizmus: A kódot általánosan írják meg különböző típusok számára, és alkalmazható az összes típusra, amely megfelel bizonyos, a kódban előre megadott feltételeknek. Objektumorientált környezetben sablonnak vagy generikusnak nevezik. Funkcionális programozási környezetben egyszerűen polimorfizmusnak hívják.
- **Altípusosság: A név több különböző osztály példányait jelöli, amelyeknek a függvényt deklaráló közös őse van. Objektumorientált környezetben többnyire erre gondolnak, amikor polimorfizmusról beszélnek.**

A polimorfizmus lehet statikus és dinamikus:

- ***statikus polimorfizmus:*** metódusok túlterhelése, függvénysablonok, osztálysablonok. Statikus, fordításidejű kötés. Végrehajtása gyorsabb, mivel nem kell dinamikus kötéseket figyelni, viszont a fordítónak többet kell dolgoznia, a program lassabban és több memóriát igénybe véve fordul.
- ***dinamikus polimorfizmus:*** metódusok felülírása. Dinamikus, futásidejű kötés. Rugalmasabb, de lassítja a futást. Egy dinamikusan linkelt könyvtár az objektumokat azok pontos típusának ismerete nélkül is képes kezelni.

Általában statikus az ad hoc és a paraméteres polimorfizmus, és dinamikus az altípusos. Ezzel szemben elérhető a statikus altípusosság metaprogramozással, pontosabban CRTP-vel (curiously recurring template pattern).

Metódustúlterhelés: A metódustúlterhelés lehetővé teszi számunkra, hogy több, ugyan azzal a névvel ellátott metódus létezzen, amennyiben eltérő paraméterekkel rendelkeznek. (Ugyan úgy, mint Java-ban a konstruktor túlterhelés)

Háromféleképpen tudunk túlterhelni egy metódust:

1. A paraméterek száma eltérő:
```
add(int, int)
add(int, int, int)
```
2. A paraméterek típusa eltérő:
```
add(int, int)
add(int, float)
```
3. A paraméterek sorrendje eltérő:
```
add(int, float)
add(float, int)
```

Fontos megjegyezni, hogy ez nem vonatkozik a metódus visszatérési értékére. Tehát a következő példa nem működik:

```
int add(int, int)
float add(int, int)
```

A bezárási eszközrendszer (ezt ki hívja így?)**:**
A bezárási eszközrendszer teszi lehetővé egy osztály adattagjainak és metódusainak láthatóságának szabályozását. Java-ban például 4 féle láthatóság áll rendelkezésünkre:

- **private:** Csak az adott osztályban látható, amelyikben létrehozták.
- **no modifier/package private:** Kizárólag az adott package-n belül látható, amelyikben létrehozták. (Ha nem adunk meg Java-ban láthatóságot, ez az alapértelmezett)
- **protected:** Package private + más package-kben létrehozott alosztályok(subclassok) számára is látható.
- **public:** Mindenhonnan látható.

	Class	Package	Subclass (same pkg)	Subclass (diff pkg)	World
`public`	+	+	+	+	+
`protected`	+	+	+	+	
no modifier	+	+	+		
`private`	+				

+ : accessible
blank : not accessible

Absztrakt osztályok: Az absztrakt osztályok nem példányosíthatóak, viszont örökölhetnek belőle alosztályok. Az absztrakt osztályok metódusait implementálhatjuk, vagy csak megadhatjuk őket, amiket később az alosztályok fognak implementálni vagy felülírni.

Interfészek: Az interfészek lényegében az absztrakció és egységbezárás vegyített változata. Az interfészek metódusait az osztály leírásában nem implementálhatjuk. Egy interfészt bármennyi osztály implementálhat. Az interfészen belül csak a metódusok nevét, paramétereit és visszatérési értékét adhatjuk meg, amelyeket az interfészt megvalósító osztályokban kell implementálni.

Típustagok:

Egyedtípuson a típusok tagjaiként előforduló entitásokat értjük. Minden egyednek van neve (name) és hozzáférhetősége (accessibility), utóbbi hét előredefiniált érték valamelyike lehet.

Ötféle egyedtípust lehet típustag, ezek rendre a mezők (field), metódusok (method), beágyazott típus (nested type), tulajdonságok (property) és események (event).

Az egységes típusrendszerben formalizált összes típus közül a legtöbbnek bármennyi tagja is lehet.Szaknyelven szólva, a típustagot a halmaz korlátozza (konstruktor, véglegesítő, statikus konstruktor, beágyazott típus, operátor, metódus, tulajdonság, indexelő, mező, írásvédett mező, konstans, esemény). Az egységes típusrendszer különféle „díszítőelemeket” definiál, amelyeket hozzárendelhetünk egy adott taghoz. Minden tag rendelkezik például egy adott láthatósági jellemzővel (pl. nyilvános, privát, védett stb.). Néhány tag absztrakt tagként deklarálható, amely a származtatott típusokra polimorf viselkedést kényszerít, valamint virtuális tagként, amely alapértelmezett (de felüldefiniálható) megvalósítást definiál. Sőt a legtöbb tagot statikusként (osztályszinten kötött) vagy példányként (objektumszinten kötött) is konfigurálhatjuk.