3. tétel
Polimorfizmus és dinamikus összekapcsolás. Objektum, osztály, tulajdonságok és kapcsolatok. Késői és korai típuslekötés

Polimorfizmus (többalakúság): azonos nevű műveletet más osztály objektumai másként is megvalósíthatnak

Azt a lehetőséget, hogy egy változó a program futása során több osztály objektumára is hivatkozhat, nevezzük a változók polimorfizmusának Az objektumorientált programozásban használt polimorfizmus fogalom lényegében az altípusos polimorfizmust jelenti. Az öröklődés során megengedjük, hogy a leszármazottaknál az egyes műveletek implementációja és/vagy specifikációja különbözzön az ősosztályban bevezetett művelettől.

Dinamikus összekapcsolás: a változó dinamikus típusa határozza meg a művelet végrehajtandó implementációját. A dinamikus összekapcsolás futás idejű esemény. Számos programozási nyelvben azt a műveletet, amely dinamikusan kapcsolódik az adott objektumhoz, virtuális műveletnek nevezik.

Objektum:
az objektumok egyedeket képviselnek, amely egyedek lehetnek akár fizikai, akár fogalmi, akár számítógépes dolgok

Az objektumok a modellezendő valós világ egy-egy önálló egységét jelölik. Az objektumokat meghatározza belső állapotuk és a nekik küldhető üzenetekre való reagálásuk. A reagálás jelentheti azt, hogy megváltozik az objektum belső állapota, illetve az objektum elvégez valamilyen műveletet, kijelez a külső világ számára valamilyen értéket, üzenetet küld más objektumnak, létrehoz, töröl, másol, illetve mozgat más objektumokat stb…

Az objektumokat a számunkra lényeges tulajdonságaik, viselkedési módjuk alapján megkülönböztetjük és kategóriákba**, osztályokba soroljuk**, oly módon, hogy a hasonló tulajdonságokkal rendelkező objektumok egy osztályba, a különböző vagy eltérő tulajdonságokkal rendelkező objektumok pedig külön osztályokba kerülnek. Az osztályozás folyamata tulajdonképpen az általánosítás és specializálás műveleteinek segítségével valósul meg. Az objektum információkat tárol, és kérésre feladatokat hajt végre. Ilyen értelemben az objektum nem más, mint **adatok(attribútumok)** és **metódusok (műveletek)** összessége, ez utóbbiak elvégzik az objektumra szabott feladatot vagy leírják az objektum viselkedését.

Az objektumoknak mindig van belső állapotuk (adatok pillanatnyi értékei). Metódushívások után az objektumok állapota változhat. Az objektumok emlékeznek állapotukra és a feladatvégzési folyamat mindig egy kezdőállapotból indul, és egy másik állapotba megy át. A következő állapotátmenetnél onnan folytatja a folyamatot, ahol előzőleg abbahagyta. **Azonosíthatóság:** ha adott osztálybeli objektumok belső állapota azonos még nem jelenti azt hogy a két objektum azonos. Az objektumok nincsenek egyedül, **kapcsolatban állnak egymással**, és ezek az objektumok kommunikálnak egymással, amely üzenetküldés formájában történik. Ezeket az üzeneteket általában metódushívással ábrázoljuk. Az osztályok leírásához a legmegfelelőbb eszközt az absztrakt adattípusok jelentik: jellemezni kell az objektumok értékhalmazát és rögzíteni az elérhető műveletek halmazát, definiálva ezek absztrakt tulajdonságait.

Lokális felelősség elve: minden objektum felelős önmagáért.

Kapcsolatok:

- Név adható neki, de nem kötelező
- Nevet akkor kötelező adni egy kapcsolatnak, ha ugyanazon osztályok között több kapcsolat is él
- Az kapcsolatok végeit szerepkörnek nevezzük: ahogyan az adott osztályt a vele kapcsolatban lévő osztály látja
- Egyes vélemények szerint ez pongyola megfogalmazás, a típust és az osztályt el kellene választani egymástól
- Hasonló a szerepe mint a tulajdonságnak (pl: láthatóság)

Késői-, korai típuslekötés: a virtuális függvényhívásokat csak futási időben oldja fel a C++ rendszer - innen származik a késői kötés elnevezés. Tehát azt a döntést, hogy például melyik `show` függvényt is kell aktivizálni, el lehet halasztani egészen addig a pillanatig, amikor a ténylegesen megjelenítendő objektum pontos típusa ismertté nem válik a program futása során.

A C++ alapvetően háromféleképpen tudja az azonos névvel ellátott függvényeket egymástól megkülönböztetni:

- a paraméterlisták különbözősége (az ún. paraméterszignatúra) alapján, tehát `show(int i, char c)` és `show(char *c, float f)` egymástól különböző függvények,
- az érvényességi tartományt definiáló operátor alapján, tehát a `rectangle::show` és a `triangle::show`, valamint a `::show` függvények különbözőek,
- illetve az egyes objektum-példányok függvénymezőit a mezőkiválasztó operátor(ok) segítségével.

Az ilyenfajta függvényhivatkozások feloldása mind fordítási időben történik. Ezt nevezzük *korai*, vagy *statikus összerendelésnek* (*early binding, static binding*).