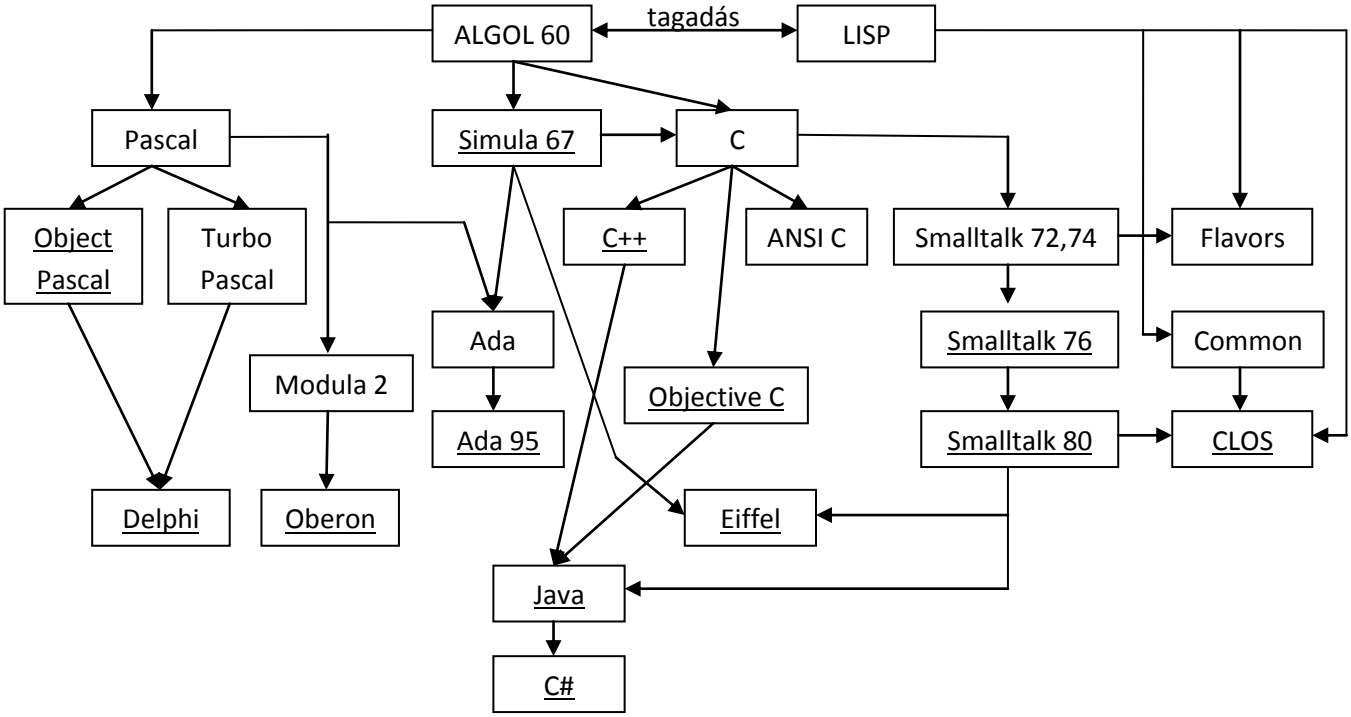
SZOFTVERFEJLESZTÉSI TECHNOLÓGIÁK

Programnyelv fa:

A nyilak irányába hatnak egymásra a nyelvek. Az ALGOL 60 előtt is voltak nyelvek (pl.: Fortran). Az aláhúzottak objektumorientált nyelvek (**későbbiekben röviden OO nyelvek**).



Programozási paradigmák (programozási minták, módszerek és technikák):

- IMPERATÍV PROGRAMOZÁS:

Az imperatív paradigma alakult ki történelmileg először. Egy imperatív programban lépésenként megadják, hogy mit kell tenni és milyen adatokkal. Konyhanyelven: egy imperatív program, olyan mint egy szakácskönyv. Először felsorolják a hozzávalókat (változók deklarálása), majd lépésekben megadják az eljárást: mit kell tenni először, hogyan kell folytatni a dolgot, esetleg ismételni kell valamely lépéseket (ciklusok), esetleg választani kell, hogyan folytassuk a munkát (elágazás). A program egy tevékenységsorozat leírása.

Ide tartozó programnyelvek: C, Pascal, ADA, Modula-2, Basic nyelvek, FORTRAN, COBOL

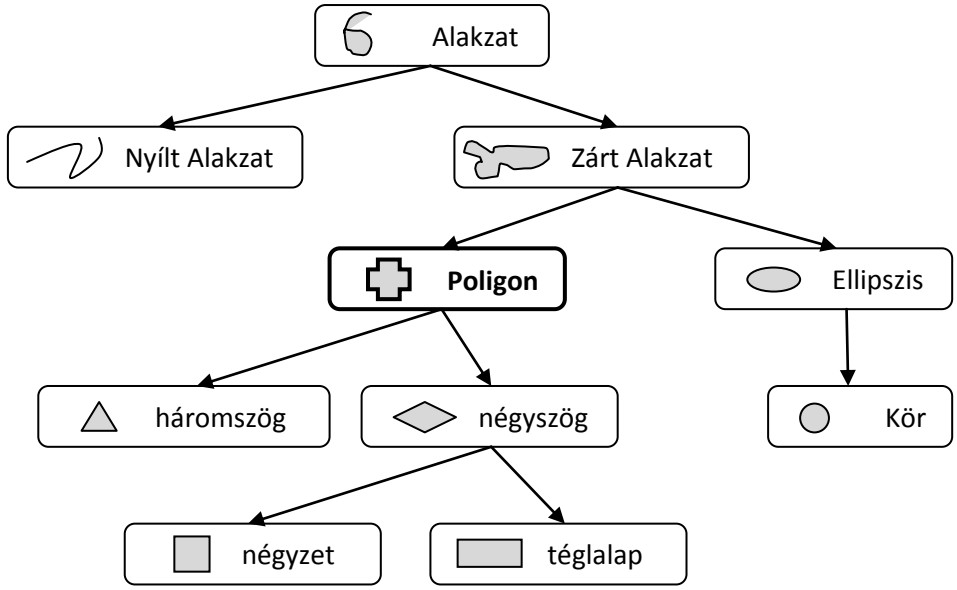
- DEKLARATÍV PROGRAMOZÁS:

- *FUNKCIONÁLIS PROGRAMOZÁS:* Középpontjában a függvények és egy átíró (redukciós) rendszer állnak. Egy funkcionális nyelvben a program a következőkből áll: típus, osztály, függvénydeklarációk, függvénydefiníciók, illetve egy kezdeti kifejezésből áll. A program végrehajtását a kezdeti kifejezés kiértékelése jelenti, amelyben tetszőleges számú akár egymásba ágyazott függvényhívás sorozat jelenhet meg.

  Ide tartozó programnyelvek: LISP, Miranda, Haskell, SASL (St. Andrews Standard Language)

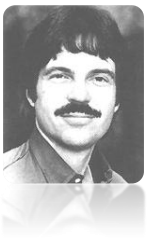
• *LOGIKAI PROGRAMOZÁS:* A Prolog megkonstruálásával születik meg. A matematikai logika fogalom és eszközrendszerén épül fel. Egy logikai program nem más, mint egy absztrakt modellre vonatkozó állítások egy halmaza. Az állítások a modell elemeinek tulajdonságait és a közöttük lévő kapcsolatokat formalizálják. A logikai programokban a programozó felsorol (deklaratív módon megad) szabályokat és tényeket. Megad ezek után egy eldöntendő kérdést, és a programozási rendszer következtető motorja (inference engine) automatikusan kikövetkezteti a szabályok és tények feldolgozásával hogy az igaz-e vagy sem.
Ide tartozó programnyelvek: Prolog, λProlog, CLACL, Fril

- OBJEKTUMORIENTÁLT PROGRAMOZÁS:



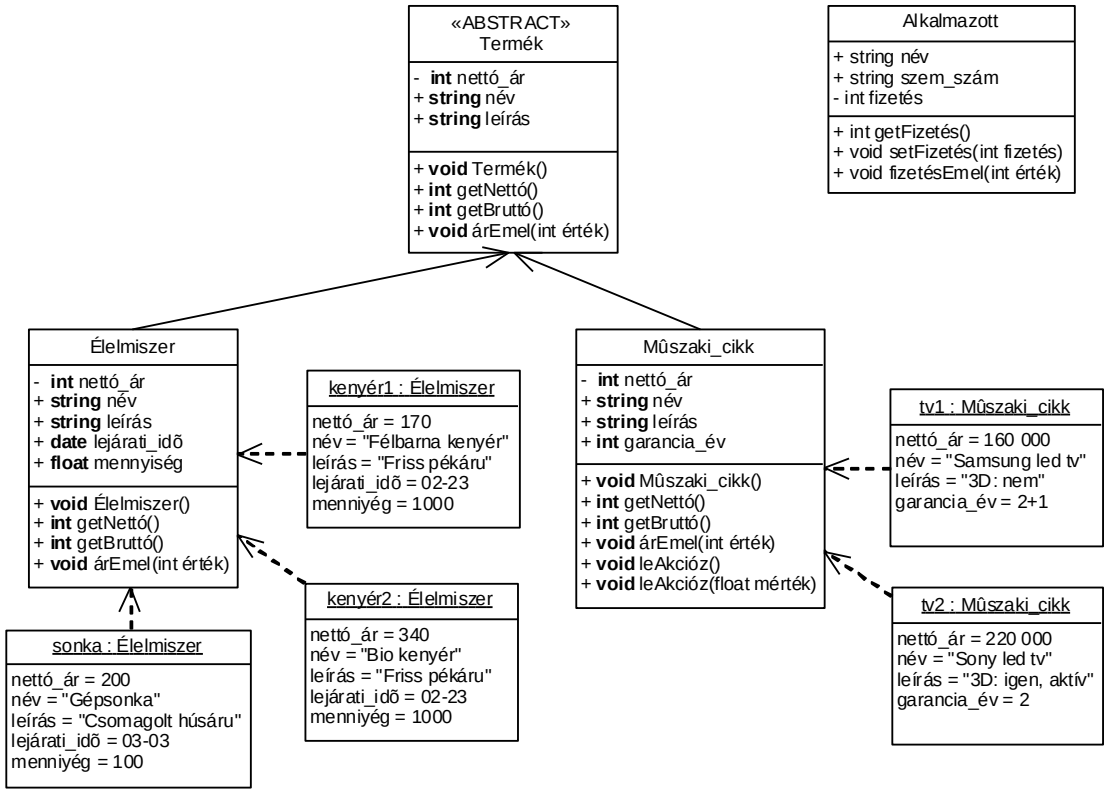
Nem a műveletek megalkotása áll a középpontban, hanem az egymással kapcsolatban álló programegységek hierarchiájának megtervezése. Az objektumorientált gondolkodásmód lényegében a valós világ lemodellezésén alapul – például egy hétköznapi fogalom, a „poligon" felfogható egy osztály (a poligonok osztályaként). A paradigma alapja az osztály. Az osztálynak vannak módszerei, attribútumai. (módszerek: eljárás, függvény; attribútum: tulajdonság). Az attribútumok az osztály statikus részét, a módszerek (funkcionális modell) a viselkedést írják le. Az osztály egy absztrakt programozási eszköz.
Az OO szemlélet szerint az adatmodell és a funkcionális modell egymástól elválaszthatatlan, külön nem kezelhető. A valós világot egyetlen modellel kell leírni és ebben kell kezelni a statikus (adat) és a dinamikus (viselkedési) jellemzőket. Ez az egységbezárás elve.
A poligon osztályból készíthetünk poligon objektumokat a példányosítás során. Minden poligon objektum ugyanazzokkal a tulajdonságokkal (pl.: élek száma, élek hossza stb.) és metódusokkal (pl.: megjelenítés(), terület()- kerületszámítás()) rendelkezik, mint amikkel a példányosító osztály rendelkezik (ez nem öröklődés).
Az objektumorientált programozásban fontos szerep jut az úgynevezett öröklődésnek, ami az osztályok egymásból való származtatását teszi lehetővé: a poligonok osztálya származhat a „zárt alakzatok" osztályból, így megörökli az zárt alakzatok tulajdonságait és metódusait, valamint kibővítheti vagy felülírhatja azokat a poligon tulajdonságaival, metódusaival.
Ide tartozó nyelvek: C++, Java, C#, Delphi

OO (objektum orientált) paradigma rövid története

- Kifejlesztik a SIMULA 67-et amelyben a nyelv definíciója új eszközrendszert tartalmaz, amely az OO paradigma alapja
- Alan Kay diplomamunkájában felvázolja az OO paradigmát.
- 
  A Xerox cég elindít egy nagy projektet (személyi számítógép megtervezése és megvalósítása)
  A Xerox megbízza Kay-t, hogy fejlesszen hozzá nyelvet. Így születik meg a Smalltalk. Alan Kay idézete szerint: „A legjobb módja, hogy megjósold a jövőt az, ha feltalálod”. Nevéhez fűződik még a laptop számítógépek megálmodása és a modern grafikus felhasználói felületek.
- A Smalltalk nyelv ténylegesen tartalmazza az OO paradigmát.
  Két probléma: 1, a Smalltalk implementációjához nincs elég hardver. 2, a szakma nem érti meg a paradigmát. Teljesen más a szemlélet.
- A 80-as években a Smalltalk nyelvi paradigmává válik
- A 80-as években lévő nyelveknek megjelenik egy OO változata, így jelenik meg a C++, ami nagyon sokáig meghatározó.
- A 90-es évek már az OO-ról szólnak.
- 1995-ben megjelenik a Java, a 90-es évek végére ő a meghatározó.
- 2005-ben megjelenik a C# ami szintén OO nyelv.

OO paradigma alapfogalmai:

Az OO paradigma alapfogalmait egy példán keresztül fogjuk végigvezetni, a könnyebb érthetőség miatt. A példa program vázlata: Egy hipermarketet fogunk modellezni. Termékeket árulunk. Különböző típusú termékeknek különböző paramétereik vannak. A **hipermarket** osztály/objektum diagramja a következőképpen néz ki:



OSZTÁLY (CLASS)

Az **osztály** egy adattípus, melyet úgy képzelhetünk el, mint bizonyos objektumok sablonját, amely meghatározza a konkrét objektumok viselkedését és tulajdonságait. Az osztály egy absztrakt programozási eszköz.

A hipermarket példában az *Élelmiszer,* a *Műszaki_cikk és az Alkalmazott* egy-egy osztály. A *Termék* is egy osztály, de ő absztrakt osztály (lásd később).

Az OO nyelvek implementációi gyakran egy osztály együttesként jönnek létre. Az attribútumok tetszőleges bonyolultságú adatstruktúrát írhatnak le. A módszerek szolgálnak a viselkedés megadására. Ezek általában megfelelnek az eljárás orientált nyelvek alprogramjainak (metódusok). Az osztályok neveit mindig nagybetűvel kezdjük.

PÉLDÁNYOSÍTÁS (INSTANTIATION)

Egy objektum mindig egy osztály példányaként jön létre a példányosítás során. Az objektum ettől kezdve él és tudja mely osztály példányaként jött létre. Egy adott osztály minden példánya azonos adatstruktúrával és azonos viselkedésmóddal rendelkezik. Példányosításkor:

- lefoglalódik egy memóriaterület az objektum számára
- elhelyezésre kerülnek a példány szintű attribútumok (lásd később)
- beállítódik a kezdőállapot (erre az OO nyelvek egy speciális módszert a konstruktort használják, lásd később.)

Adott osztálytípussal több objektum példányt is létrehozhatunk. A példányok saját adatterülettel rendelkeznek, de a metódusokat közösen használják. Az aktuális példány címét a *Self* vagy *this* paraméter adja, mely minden objektum utolsó nem látható paramétere.

Általában a nyelvekben a *new* operátorral lehet egy osztályból egy konkrét példányt elkészíteni. Az elkészített példányt más néven objektumnak, konkrét előfordulásnak vagy angolul instance –nak nevezzük. A példányok neveit kisbetűvel írjuk.

A hipermarket példában a *kenyér1*, *kenyér2*, *sonka*, *tv1* és *tv2* egy-egy példány. Az első három, példányosító osztálya a *Kenyér* osztály, az utolsó kettőnek a *Műszaki_*cikk osztály.

TAGFÜGGVÉNY

Egy adott osztály metódusait tagfüggvényeknek nevezzük. A tagfüggvények mindig az adott osztályból készített objektumon dolgoznak.

A hipermarket példában az *Élelmiszer* osztálynak több tagfüggvénye is van pl. az *árEmel(int érték)* tagfüggvénye arra való, hogy az adott élelmiszer árát egy adott mennyiséggel megnöveljük, de ezt a tagfüggvényt egy konkrét elemen az *Élelmiszer* osztály egy konkrét előfordulásánál (példányánál) fogjuk meghívni. Konkrét példa: Az *Élelmiszer* osztályból készítünk egy példányt a kenyér1-et. És a *kenyér1*-nek ha meg akarom növelni az árát 10Ft-al, akkor meghívom a megfelelő tagfüggvényét: *kenyér1.árEmel(10).* Így a *kenyér1* árát megnöveltem, a *kenyér2* példány ára változatlan marad.

PÉLDÁNYSZINTŰ ATTRIBÚTUM VS. OSZTÁLY SZINTŰ ATTRIBÚTUM

Egy osztály attribútumai és módszerei lehetnek osztály és példány szintűek:

- A példányszintű att-ok minden példányosításnál elhelyezésre kerülnek a memóriában, ezek értékei együttesen adják meg a példány állapotát.
- Az osztály szintű att-ok az osztályhoz kötődnek, nem „többszöröződnek" a memóriában (közös memória területre dolgoznak)
- Az osztály szintű metódusokat, nem a példányokhoz vonatkozóan hívjuk meg, hanem az osztályhoz kapcsolódóan. pl: Osztálynév.metódusnév()

A hipermarket példában ha meghívom az *árEmel*() tagfüggvényt a *kenyér1* példánynál, akkor csak a *kenyér1* példány ára emelkedik, mivel az *Élelmiszer* osztály *nettó*_ár attribútuma egy példányszintű attribútum. Ezáltal minden példány saját árral „dolgozhat". Ha osztály szintű lenne az *Élelmiszer* osztály *nettó*_ár attribútuma, akkor egy áremelésnél, az *Élelmiszer* osztály összes példányának (*kenyér1*, *kenyér2*, *sonka*) ára megemelkedne, mivel mindegyik példány közös memóriaterületen tárolná az árat.

EGYSÉGBEZÁRÁS (ENCAPSULATION)

Az OO szemlélet szerint az adatmodell és a funkcionális modell egymástól elválaszthatatlan, külön nem kezelhető. A valós világot egyetlen modellel kell leírni és ebben kell kezelni a statikus (adat) és a dinamikus (viselkedési) jellemzőket és elzárjuk őket a külvilág elől. Ez az egységbezárás elve.

A hipermarket példában a két osztály (Élelmiszer, Műszaki_cikk) mind megvalósítják az egységbezárás elvét. Magukba zárják egy adott termékre jellemző adatokat (tulajdonságokat) és a rajtuk dolgozó metódusokat. A két dolgot együtt kell kezelni egymástól szét nem választható. Minden egyes osztály előfordulása (kenyér1, kenyér2, stb.) rendelkezik ezen adatokkal és egy közös névvel, az objektum nevével (kenyér1.leírás) hivatkozhatunk rájuk.

OBJEKTUM (OBJECT) VAGY MÁS NÉVEN PÉLDÁNY (INSTANCE)

Konkrét nyelvi eszköz. Egy objektum mindig egy osztály példányaként jön létre a példányosítás során. Egy adott osztály minden példánya azonos adatstruktúrával és azonos viselkedésmóddal rendelkezik. Minden objektumnak van címe, az a memóriaterület, ahol az adatstruktúra elemei elhelyezkednek. Az adott címen elhelyezkedő érték együttest az **objektum állapotá**nak hívjuk. A példányosítás folyamán az objektum kezdőállapotba kerül.
Az OO szemléletben az objektumok egymással párhuzamosan, egymással kölcsönhatásban léteznek.

A hipermarket példában a kenyér1, kenyér2, sonka, tv1, tv2 egy-egy objektum (példány). A kenyér1 állapotát az adja, hogy az attribútumainak aktuálisan milyen értékei vannak. Ezek az értékek időben változhatnak (pl.: megemeljük az árát) és akkor egy újabb állapotba kerül az objektum.

AKTUÁLIS PÉLDÁNY

Azt az objektumot, amelyen egy módszer éppen operál, aktuális példánynak hívjuk.

ABSZTRAKT OSZTÁLY (ABSTRACT CLASS)

Egy olyan eszköz, amellyel viselkedésmintákat adhatunk meg, amelyeket aztán valamely leszármazott osztály konkretizál. Egy absztrakt osztályban általában vannak absztrakt módszerek, ezeknek csak a specifikációja létezik, implementációjuk nem. Egy absztrakt osztályból származtatható absztrakt és konkrét osztály is. A konkrét osztály minden módszeréhez kötelező az implementáció. Egy osztály mindaddig absztrakt marad, amíg legalább egy módszere absztrakt. Az absztrakt osztályok **nem példányosíthatók**, csak örökölhetők.

A hipermarket példában a *Termék* egy absztrakt osztály. Ez azt jelenti, hogy belőle nem készíthetünk példányokat. Belőle csak származtathatunk. Azért nem lehet példányosítani, mert a metódusaihoz nincs implementáció (kód rendelve). Nincs megírva pl a *getBruttó*() metódusának kódja, azt majd egy leszármazott osztálya pl az *Élelmiszer* osztály fogja meghatározni, mivel más és más lehet az egyes termékek után felszámolt adó mennyisége. Így megvalósítható az, hogy a kenyér1.getBruttó() metódusa x áfával, míg a tv1.getBruttó() metódusa y áfával számol. Más lesz a kódjuk, de ugyanúgy kell őket használni, ettől áttekinthetőbb és egységesebb lesz a kód.

INTERFÉSZ (INTERFACE)

Csak absztrakt metódusokat tartalmazhat, továbbá konstans és változódeklarációkat. Az interfész egy felületet nyújt, itt nem az öröklődés a cél, hanem az hogy az alkalmazás két pontja egy absztrakt egy absztrakt felületen (interfészen) keresztül tudjon egymáshoz kapcsolódni, és ne kelljen ezen kívül semmit se tudnia a másikról. Az absztrakt osztályt arra használják, hogy közös őse legyen a leszármazottaknak, az interfész viszont a kommunikáció eszköze. Az interfész nem példányosítható, csak megvalósítható, ami abból áll, hogy a megfelelő metódusokat implementáljuk. Egy megvalósító osztály egyszerre több interfészt is megvalósíthat.

ÖRÖKLŐDÉS (INHERITANCE)

Az öröklődés során egy már létező osztályhoz (a szuperosztályhoz, vagy ősosztályhoz) kapcsolódóan hozunk létre egy új osztályt, mely az alosztály. Az öröklés lényege, hogy az alosztály átveszi a szuperosztály minden (a bezárás által megengedett) attribútumát és módszerét. Ezen túlmenően új attribútumokat és módszereket definiálhat, az átvett eszközöket átnevezheti, az átvett neveket újradeklarálhatja a láthatósági viszonyokat és a módszereket újraimplementálhatja.

Az öröklődés lehet egyszeres és többszörös. Egyszeres öröklődés esetén egy osztálynak pontosan egy, többszörös esetén több szuperosztálya lehet. Mindkét esetben akárhány alosztály létrehozható. Természetesen egy alosztály lehet egy másik osztály szuperosztálya. Így egy osztályhierarchia jön létre. Az osztályhierarchia egyszeres öröklődés esetén fa, többszörös esetén aciklikus gráf (nem tartalmaz kört). Két kitüntetett elem a gyökér, amelynek nincs szuperosztálya, és a levél, amelynek nincs alosztálya.

A hipermarket példában az *Élelmiszer* és *Műszaki_cikk* osztály a *Termékek* osztályból származik, így megöröklik a *Termékek* osztály összes attribútumait és metódusait. Továbbá saját attribútumokkal és metódusokkal is kiegészítik azokat pl: az *Élelmiszer* osztályban az örökölt attribútumok (nettó_ár, név, leírás) kiegészülnek újabb attribútumokkal a *lejárati_idő*-vel és a *mennyiség*-gel. Ugyanez a kiegészülés figyelhető meg a metódusok esetében is a Műszaki_cikk osztálynál. Itt a megörökölt metódusokat (termék, getNettó, getBruttó, árEmel) kiegészítjük egy újjal a leAkcióz() metódussal.

POLIMORFIZMUS

Egy alosztály az örökölt módszereket újraimplementálhatja, ezeket a módszereket polimorf módszereknek nevezzük. Így különböző osztályokban azonos módszerspecifikációhoz különböző implementáció tartozik. Ha meghívunk egy ilyen módszert, akkor a kötés mechanizmus adja meg, hogy melyik implementáció fog lefutni. Két ilyen kötési mechanizmus van, a statikus és a dinamikus.

- Statikus kötés esetén már fordításkor eldől a kérdés.
- Dinamikus kötés esetén a kérdés csak futás időben dől el, a megoldás a helyettesíthetőségen alapul. Annak az objektumnak a példányosító osztályában megadott implementáció fog lefutni, amelyik ténylegesen kezelésre kerül.

A hipermarket példában ha a *Termékek* osztály nem lenne absztrakt osztály és a *getBruttó()* metódusához lenne implementáció rendelve, amely egy alap áfa értékkel számolna, akkor minden belőle származtatott osztály ezzel az áfával számolna, de ha valamely leszármazottban mégis szükséges lenne ezt az értéket/számítási módot megváltoztatni, akkor az örökölt implementációt felül lehet írni, annak érdekében hogy más adóval számoljon. Ekkor a *getBruttó()* metódust polimorf metódusnak nevezzük.

KLIENS OSZTÁLY

Az egymással előd-leszármazott viszonyban nem levő osztályokat kliens osztályoknak hívjuk (nincs öröklődési kapcsolat).

A hipermarket példában az Alkalmazott osztály kliens osztály. Nincs szülő-leszármazott viszonyban más osztállyal.

ÜZENETKÜLDÉS

Az objektumok kommunikációja üzenetküldés formájában történik. Minden objektum, példányosító osztálya meghatározza azt az interfészt, amely definiálja, hogy más objektumok számára az ő példányainak mely attribútumai és metódusai látszanak (erre szolgál a bezárási eszközrendszer lásd később).
Tehát egy objektum küld egy üzenetet egy másik objektumnak (ez általában egy számára látható metódus meghívásával és az üzenetet fogadó objektum megnevezésével történik), a másik pedig megválaszolja azt (a módszer visszatérési értéke, vagy változó paraméter segítségével). Az üzenet hatására lehet, hogy az objektum megváltoztatja az állapotát.

A hipermarket példában tegyük fel, hogy egy alkalmazott megbeszéli a hipermarket vezetőségével, hogy ő hazavinne egy tv-t, de úgy hogy most a tv árának csak a felét fizetné ki, a másik felét pedig kéri, hogy a következő havi béréből vonják le. Ez az alkalmazott legyen az *alkalmazott1* példány a kérdéses tv pedig a tv1. Ekkor a béréből való levonás a következőképpen alakul:

```
alkalmazott1.setFizetés( alkalmazott1.getFizetés() – tv1.getBruttó() / 2 )
```

Ebben az esetben a két objektum (alkalmazott1 és tv1) egymással kommunikál a metódusaik paramétereivel és visszatérési értékeikkel.

TÚLTERHELÉS (OVERLOAD)

Az OO nyelvek általában megengedik a metódusnevek túlterhelését. Ez annyit jelent, hogy egy osztályon belül több azonos nevű de természetesen eltérő implementációjú módszereket tudunk létrehozni. Ekkor a hivatkozások feloldásához a specifikációknak különbözniük kell a paraméterek számában, vagy azok típusában.

A hipermarket példában a *Műszaki_cikk* osztályban a *leAkcióz()* metódusnév túl van terhelve. A két metódus neve megegyezik, de a paraméterlistája és implementációja nem. Az üres paraméterlistával rendelkező *leAkcióz()* metódus mondjuk egy 10%-os akciót valósít meg és ennyivel csökkenti a termék árát. Az egy paraméterrel rendelkező *leAkcióz(float mérték)* metódus pedig a paraméterben megadott mennyiséggel csökkenti a termék árát.
Ez így egy kényelmi szolgáltatás. Ha nem adunk meg paramétert a függvényhívásnál, akkor az első eset fog megtörténni és egy átlagos akcióról lesz szó, ha viszont megadjuk a paramétert, akkor konkrétan annyival lesz csökkentve a termék ára.

BEZÁRÁSI ESZKÖZRENDSZER

Az absztrakt adattípus (osztály) az attribútumainak és metódusainak láthatóságát szabályozhatja, ezt valósítja meg a bezárási eszközrendszer. Általában 3 szintű bezárás van:

- **privát** láthatóság: kívülről nem látható. Az eszközöket csak az adott osztály látja. Jelölése: -
  A hipermarket példában az *Élelmiszer* osztályból készített példányok *nettó_ár* attribútumát csak maguk az *Élelmiszer* osztályból példányosított objektumok érhetik el. A *műszaki_cikk* osztályból készített objektumok nem látják az *Élelmiszer* osztály példányainak *nettó_ár* attribútumait, mivel azok privátak. Ahhoz az attribútumhoz csak a *getNettó()* metóduson keresztül férhetnek hozzá.
- **védett** láthatóság: az osztály és a leszármazottjai férhetnek hozzá az eszközökhöz. Jelölése: #
- **publikus** láthatóság: minden osztály láthatja az osztály eszközeit. A kliens osztályok is. Jelölése: +

A hipermarket példában az előbb megnézett példában az üzenetküldésnél a *tv1* megvásárlása az alábbi módszerrel nem lenne megvalósítható, ha a *tv1.getBruttó()* metódusa privát lenne, mert akkor azt kívülről nem lehetne elérni.

```
alkalmazott1.setFizetés( alkalmazott1.getFizetés() – tv1.getBruttó() / 2 )
```

KONSTRUKTOR (CONSTRUCTOR)

Az osztályban szereplő speciális metódus, neve megegyezik az osztály nevével, általában túlterhelhető paraméterei azt a célt szolgálják, hogy segítségükkel kezdő értéket rendeljünk a példány szintű tagokhoz példányosításkor. A konstruktor, példányosításkor hívódik meg.

A hipermarket példában az *Élelmiszer* osztály konstruktora az *élelmiszer()* metódus (tagfüggvény), a *Műszaki_cikk* osztály konstruktora pedig a *műszaki_cikk()* metódus. Amikor az *Élelmiszer* osztályból a példányosítás során létrehozzuk a *sonka* példányt, akkor meghívódik az *élelmiszer()* konstruktor és a sonka adattagjait beállítja egy kezdőállapotba.

DESTRUKTOR (DESTRUCTOR)

Ez is egy speciális metódus, mint a konstruktor. Amikor az objektum megszűnik, azelőtt hívódik meg. Feladata, hogy az esetlegesen lefoglalt memóriaterületeket felszabadítsuk.

A hipermarket példában nincs ilyen metódus.

KIVÉTELKEZELÉS (EXCEPTION HANDLING)

Célja a futás közben fellépő hibák kezelése, ezeket a hibákat kivételnek (exception) nevezzük. Kivétel akkor lép fel, ha valamilyen hiba történik egy metódus futása során vagy a programozó saját maga kivált egy kivételt, ekkor egy kivétel objektum jön létre, amely információkat tartalmaz a kivétel fajtájáról és a program aktuális állapotáról. A kivétel kiváltása után a rendszer egy megfelelő kivételkezelőt keres, és annak adja át a vezérlést, a kivétel lekezelése után a vezérlés nem adódik vissza arra a helyre ahol kiváltódott a kivétel. Egy kivételkezelő akkor megfelelő típusú a kivétel kezeléséhez, ha a kezelő által kezelt kivétel típusa megegyezik a kiváltott kivétel típusával. A kivételkezelő blokkok egymásba ágyazhatók, ezért egyszerre több alkalmas kivételkezelő is lehet, ekkor arra tevődik a vezérlés, amelyik a legközelebb áll a kiváltott kivétel helyéhez.

A hipermarket példában tegyük fel, hogy a termékeket egy felületen keresztül a felhasználó felveheti és kereshet közöttük. A termékeket tárolni kell. Amikor felvesz, egy új terméket akkor le kell menteni a merevlemezre az összes terméket, hogy későbbi használatra is megmaradjanak. De előfordulhat hogy mentés közben valami hiba lép fel, mondjuk elfogy a hely a merevlemezen, vagy nincs írási jog. Ekkor keletkezik egy kivétel, ami leírja a hiba okát. Ez lesz a kivételobjektum. Ezt kell a programban valahol elkapni és lekezelni. A lekezelés állhat abból, hogy a felhasználónak küldünk egy üzenetet hogy elfogyott a hely, mentsen máshová. Ezáltal a hiba nem okozza a program összeomlását, így nem lesz adatvesztés.

EGYSÉGESSÉG

Egyes tiszta OO nyelvek az egységesség elvét vallják. Ezen nyelvekben egyetlen programozási eszköz van, az objektum. Ezekben a nyelvekben tehát minden objektum, a módszerek és osztályok is.

HIBRID NYELVEK

A hibrid nyelvek valamilyen más paradigma (eljárásorientált, funkcionális, logikai, stb.) mentén épülnek fel, és az alap eszközrendszerük egészül ki OO eszközökkel. Ezen nyelvekben mindkét paradigma mentén lehet programozni. (C++, Object Pascal)

TISZTA OO NYELVEK

Csak OO paradigma mentén lehet programot írni bennük. (Java, Smalltalk, Eiffel)