1. **Osztályok és objektumok. Az osztály tagjai (classes objects, members)**

* az osztály egy felhasználó által létrehozott típus
* az osztály tagjai az **attribútumok (adat)** és **metódusok (viselkedés)**
* az objektumok az osztályok példányai, van állapota, viselkedése és futásidőben azonosítható
* az adattagok hozzáférése esetén megkülönböztetünk *privát, publikus* és *protected* jogosultságot
* a privát csak az osztályon belül érhető el, a publikus bárhonnan, a protected pedig az osztályban és annak leszármazottjaiban

1. **Statikus tagok (static members)**

* statikus tagok az OOP esetén nem objektum szintű adattagok, hanem osztályszintűek
* statikus tag = statikus adat + hozza tartozó statikus metódus
* Az adatok az osztály létrejöttekor lesznek létrehozva, nem szükséges példányosítani
* A példány szintű függvények használhatják a statikus adatokat mert azok???
* Instance methods may use static data. Why? – mivel a statikus adatok osztaly szintuek, ezert a peldanyok tudjak hasznalni azt
* Static methods cannot use instance data. Why? – mert a statikus metodusok csak statikus adatokkal dolgozhatnak
* a függvények is lehetnek statikusak, nem használhatják az objektumok adattagjaihoz
* statikus metódusok nem lehetnek *const* típusúak sem? A statikus metodusnak nincs this -je es a const az a this-re van hatassa. A const fuggvenyek nem valtoztathatnak az adattagokon.
* nem objektumra hívódnak meg hanem az egesz osztalyra 🡪 nem rendelkeznek *this* pointerrel
* az adattagok a futtatás teljes idejében elérhetőek, c++ esetén deklarálni a header-ben kell, inicializálni a .cpp fájlban

1. **Egységbezárás, az információ elrejtése (encapsulation, information hiding)**

* az egységbezárás adatok és a metódusok osztályba való öszezárását jelenti
* az objektum egységbezárja az állapotot a viselkedésmóddal 🡪 az objektum állapotát csak a metódusokok keresztül érhetjük el vagy módosíthatjuk, ez pedig a biztonságot szolgálja
* elrejtjük az osztály implementációját és annak részleteit
* A felhasználó egy a számára elérhető metódusokkal dolgozik, úgymond egy interface-en keresztül
* Sokkal karbantarthatóbb lesz így a kód
* az **információ elrejtése** alatt az objektum elrejti az adatait és bizonyos műveleteit ezzel elkerulve az illetektelen hozzaferest
* nem tudjuk az adatok tárolásának módját, illetve a műveletek implementálását sem
* A kulonbseg az egysegbezaras es az informacio elrejtese kozott hogy az egysegbezaras inkabb a rendszer komplexitasat hivatott elrejteni, mig az informacio elrejtese inkabb a bizotonsagra fokuszal

1. **Származtatás, örökítés (inheritance)**

* a származtatás olyan osztályok között értelmezett viszony, amely segítségével egy általánosabb típusból (ősosztály) egy sajátosabb típust tudunk létrehozni
* uj adat hozzaadasa, uj funkcionalitas hozzaadasa, metodusok felulirasa/modositasa pl print()
* az utódosztály adatokat és műveleteket örököl, kiegészíti ezeket saját adatokkal és műveletekkel, illetve felülírhat bizonyos műveleteket
* Orokiteskor a letrejovesi sorrend: Alap osztaly constructor -> orokitett osztaly constructor. Felszamolaskor forditott sorrend
* megkülönböztethető egyszeres és többszörös örökítés
* publikus, privat, protected orokites  
  public -> az ososztaly publikus es protected adattagjai ugyanazok lesznek  
  protected -> az ososztaly publikus es protected adattagjai protected-ek lesznek  
  private -> az ososztaly publikus es protected adattagjai privat-ok lesznek
* protected es publikus adattagok elerhetoek az orokitett osztalyok szamara, a privat nem

1. **Polimorfizmus. Metódusok túlterhelése és felülírása (polymorphism, overloading, overriding)**

* **polimorfizmus:** többalakúság 🡪 egy ősosztály típusú változó hivatkozhat ugyanazon közös osztályból származó osztályok példányaira
* metodusok tulterhelese – ugyanaz a nev, mas szignatura
* metodusok felulirasa – ugyanaz a nev, ugyanaz a szignatura
* lehet statikus és dinamikus:
  + *statikus*: metódusok túlterhelése, sablonok 🡪 statikus, fordításidejű kötés
  + *dinamikus*: metódusok felülírása 🡪 dinamikus, futásidejű kötés

1. **Interfészek és absztrakt osztályok (interfaces and abstract classes)**

* absztakt osztályoknak nevezünk azon osztályokat, melyeknek van legalább egy absztrakt művelete, amit majd szarmaztatott osztaly implemental vagy felulir
* interface -> felületet határoz meg és nem lehet példányosítani, a műveletek implementálását az utódosztályokra bízza, csak deklaralja a muveleteket, mindent a leszarmazottak implementalnak
* c++ esetén az absztrakt műveltet a *virtual* kulcsszó segítségével kell bekapcsolni, ez engedélyezi a későbbi felülírás lehetőségét is, implementációt az adott osztályban is tartalmaz
* c++ esetén nem léteznek úgynevezet interfészek, csupán tiszta virtuális függvényekből álló osztály, ebben az esetben nem tartozik a deklarációhoz semmiféle implementálás, az utódosztályokra van bízva: ***virtual*** *T <fun>()* ***= 0;***

1. **Kivételek és feltételek (exceptions and assertions)**

* a kivételek egy hibakezelési mechanizmust biztosítanak
* olyan hibák kezelhetőek általuk, melyekre számítani lehet
* a visszatérítési értékkel ellentétben nem biztonságosabb, mivel a hívónak muszáj kezelni, kihagyhat hívásokat a stackről illetve további információkat is tárolhat
* megtalálható néhány előre megírt kivétel, viszont mi is létrehozhatunk újakat, csupán az *exception* osztályból kell származtatni
* a feltétel vagy **assertion** egy régebbi típusú hibakezelési lehetőség
* a paraméterét kielemzi, ha ennek az értéke *false*, egy üzenetet ír ki az *std:cerr*-re, majd *abort*-ot hív (Exceptiont dob), mely megszakítja a program futását
* az assert nem minden esetben fut le, mivel szorosan kötött az NDEBUG makróhoz
* ha a makró nincs definiálva, a fordító az assert üzeneteket kihagyja a fordításból, nem lesz hatása a futásra