# 1. előadás

Előadó: Lengyel Zsolt (lengyel.zsolt@informatics.elte.hu)

Fogadóóra: Szoftvertechnológia tanszék 2.618, csütörtök 9-10, előre egyeztetett időpontban.

Kezdés: 11:15

Szünet: 15 perc valamikor közben

Vége: 13:45

Honlap: <http://lengyel.web.elte.hu/> (<http://lengyel.web.elte.hu/index.php?q=node/3>, <http://lengyel.web.elte.hu/index.php?q=node/60>)

Tervezés, tervek implementálása a fontos ebben a tantárgyban.

Tervezőeszközök: UML

Nyelv: JAVA

Objektumelvű alkalmazások tervezéséhez szükséges szoftvertechnológia eszközök megismerése, az elkészített terv JAVA platformú implementálásához szükséges ismeretek elsajátítása.

## Tematika

* Szept. 14: bevezetés
* Szept. 21: statikus modell, osztályok
* Szept. 28: osztálydiagram, osztályok közötti kapcsolatok
* Okt. 5: öröklődés, absztrakt osztályok, interfészek, polimorfizmus
* Okt. 12: objektumdiagram, osztályszintű elérés, adatok beolvasása
* Okt. 19: fájlkezelés, kivételkezelés
* Okt. 26: gyűjtemények és algoritmusok
* Nov. 9: tervminták
* Nov. 16: grafikus felhasználói felületek
* Nov. 23: grafikus felhasználói felületek
* Nov. 30: grafikus felhasználói felületelemek kommunikációja (program)
* Dec. 7: többrétegű alkalmazások
* Dec. 14: dinamikus modell, állapotok és átmenetek, állapotdiagram, szekvencia​diagram

## Követelményrendszer

Gyakorlati jegy az egyetlen jegy.

Minden gyakorlaton pluszminusz, lesz 2 db beadandó dokumentációval és 2 db ZH.

# Szoftvertechnológia

## Történeti áttekintés

1940 első számítógép

1950 tudományos és műszaki feladatok, első program, elektroncső

1960 algoritmusok, ismétlődő feladatok, lyukkártya, nyomtató, tranzisztor (Mai számítógépekben is vannak, de mikrométeresek.)
Már alkalmazza a hadiipar, űrkutatás is a számítógépeket. Fő céljuk: adatgyűjtés, kiértékelés, folyamatvezérlés.

1970 Ügyviteli feladatok: raktárkészlet, megrendelések, szállítások ütemezése

1980 Személyi számítógépek. (Olcsó, kicsi, megbízható) Nagy rendszerek mellett kisvállalkozások, oktatás, játék, szórakozás

1990 Kommunikáció, távmunka, multimédia

## Alkalmazásfejlesztési módszerek fejlődése

* Egyszerű alkalmazások, önálló fejlesztés
* Nagy rendszerek, fejlesztők együttműködése
* Szoftverkrízis: Ez ma is tart. Lassabban fejlődik a szoftvertechnológia, mint a számítógép maga.
Megoldása:
	+ Beláttuk, hogy a program = termék (funkció, minőség, költség, határidő)
	+ Előállítási technológia
	+ Projektmenedzselés

## Ágazatok

* Követelményelemzés (specifikáció): Cégnél van egy ember, aki elemzi a kapott feladatot.
* Architektúra: Adatbázisra szükség van-e, stb.
* Tervezés: Fenti 2 dolgot ismerve, ne csak nekivágjunk az implementációnak, hanem legyen tervünk. Fejlesztést könnyíti.
* Fejlesztés: Osztályokat írunk. Függvényei más osztályokkal fognak kommunikálni. "Ide még kéne egy osztály." felmerülhet. Ezért érdemes megtervezni az osztályokat előre. UML diagram segítségével.
* Tesztelés: Működik-e.
* Karbantartás: Felhasználónak adás után javítgatjuk, bővítgetjük a programot.

Mi történik, ha elakadunk valamelyik lépésnél: Vissza kell lépni 1-2-t.

Szoftvermenedzsment: Fejlesztési folyamat, Tervezőeszközök, Szoftverminőség, Formális módszerek (programszintézis).

# Szoftvertervezés, programozási technológia

## Szoftvertervezési elvek

* Absztrakció: Általánosítunk, egyszerűsítünk. Például string konkatenáció hasonlít az összeadásra, tetszőlegesen zárójelezhető. De vannak különbségek: Az összeadás felcserélhető, az összefűzés nem. Ez arra jó, hogy korábban látott hasonló megoldási módszerekért visszanyúlhatunk. Most kapott adatbázis kezelő alkalmazás feladatunk hasonlít az előző alkalmazáskezelő alkalmazásunkra. Fel tudunk használni részeket belőle.
* Modularitás: Részekre bonthatóság. Kis részeket könnyebb kezelni. Házépítés felbontható egy halom téglára. Autógyártásnál motort szerelünk a vázba, nem kell a motorgyártással foglalkoznunk. A kis részek újra felhasználhatóak. Például láncolt listákat kezelő modul. Ez akár tervkészítésnél is használható. Előző munkánk terveit felhasználhatjuk a mostani feladat tervéhez. Másik irányba is működik.
* Architektúra: (Már volt.)
* Vezérlési hierarchia: A modern számítógépet több processzor vezérli. A futtatandó program által előírt utasításokat a processzor végzi el. Legegyszerűbb eset a szekvenciális végrehajtás. Lehet párhuzamos, vagy elosztott is.
* Adatszerkezetek: A program adatokkal dolgozik. Hatékony adatkezelést szeretnénk, sok az adat. Érdemes logikusan csoportosítani mindent. Objektum elvű tervezésben: Egy adatszerkezet egy osztály ($class$). Ennek lesznek tagjai és függvényei. A tagok a reprezentáció (belső), a függvények a külső.
* Információelrejtés: Belső működést nem kötjük a világ orrára. Az adatszerkezetet különválasztjuk a reprezentációs szinttől. Csak a műveleteket publikáljuk. Ha később megváltoztatjuk a belső szerkezetet, nem kell újraírni mindent. (Nyugodtan kicserélhetjük a belső tömböt láncolt listára, mert sosem fog senki $tömb\left[16\right]$-hoz hozzáférni.) Belső működés megváltoztatása nem jelent változást a külső szemlélő számára.

Interfészek: Absztrakt osztályok, Utazni lehet autóval, busszal, stb. Az utazásra alkalmas objektumok tartozzanak az utazás interfészbe, ami rendelkezik az $utazz$ függvénnyel. Az interfészt használó program ekkor nem kell, hogy foglalkozzon az autókkal.

## Tervezési szempontok

* Újrafelhasználhatóság: Érdemes egy tervet úgy elkészíteni, hogy hasonló problémákhoz ezt a tervet újra fel lehessen használni. Minél jobb a programunk modularitása, annál könnyebb lesz újra felhasználni az egyes alkotóelemeket.
*(Beadandó, ZH nem kell, hogy újrafelhasználható legyen. Beadandónál azért az osztályok jól legyenek.)*
* Bővíthetőség: Ezt is a modularitás határozza meg. Miután a felhasználó kezébe került a program, szeretne még valami plusz funkcionalistást bele. Vagy általánosabb célra felhasználni. Fájdalommentesen belerakni a meglévő rendszerbe csak úgy lehet, ha a terv minél inkább lehetővé teszi ezt. Ha rossz a terv, akkor is tudjuk bővíteni, de nehezebb.
*(Szintén nem fontos)*
* Hibatűrés, robusztusság: Jó minőségű program. Bármilyen hardver vagy felhasználói hiba történik, jól kell tűrnie a programnak. Stabilan kell futnia. Feladata válogatja, hogy menyire fontos a stabilitás. Mobiltelefonra, szerverre például: Ne fagyjon le a program. Mobiltelefont újraindítani nem jó dolog. Szervert még inkább nem. De nem mindig érdemes erre szánni rengeteg időt és energiát.
*(Viszonylag fontos, fogunk tanulni kivételkezelést, külső és belső hibákkal való foglalkozást.)*
* Karbantarthatóság: Új funkciók belerakása, vagy felgyorsítása esetén ne kelljen újratervezni és írni mindent. Újabb hardveren is fusson.
* Modularitás: (Már volt.)
*(Fontos.)*
* Megbízhatóság: Míg a hibatűrés a külső, a megbízhatóság a belső dolgokkal foglalkozik.
*(Ne dobjon kivételt a program.)*
* Kompatibilitás: Újabb hardveren is fusson. 1.6-os JAVA-ban máshogy volt pár dolog, mint 1.7-ben. Két verzió közötti változásokat szokás dokumentálni. (deprecated function: Most még lehet használni, de később ki lesz véve. Ne használjuk.)
*(Különösebben nem fontos. Ez alatt a két félév alatt nem lesz jelentős változás.)*
* Biztonság: Ne tegyen kárt a számítógépben, a tárolt adatokat ne írja felül, ne nyúlkáljon más programok memóriájába, ne károsítsa a hálózatra csatlakoztatott gépeket.
*(Félig-meddig. Nagy bajt úgysem fogunk tudni csinálni.)*
* Használhatóság, felhasználhatóság: A felhasználó számára tükrözze az összes fontos információt. Ne legyen megtévesztő. A gombok azt csinálják, ami rájuk van írva. Könnyen legyen navigálható, logikusan legyen felépítve. Konzolos alkalmazás írja ki, hogyan kell használni, mik az elfogadott paraméterek.
*(Ötletes grafikus felületek legyenek a programunkban, ne csak a minimális feltételnek megfelelő.)*

## Modellezés, modellezőeszközök

### Modell

Szempontoknak megfelelő terv, mely alapján a program elkészíthető.

Például formaterv autóknál.

### Modellező nyelv

Strukturált információt kifejezni képes mesterséges nyelv.

Osztályok és függvényeik, kapcsolatok, példányosított objektumok. Program működése során mi fog történni.

### Unified Modeling Language (UML)

Grafikusan látjuk, mi mibe van beágyazva, mi kapcsolódik mihez, stb.

Lehet szerkezeti diagram (Csomagok, generált típusok, stb.) vagy viselkedési diagram (Milyen állapotokban vannak az osztályok, állapotátmenetek, stb.)

Akár automatikusan lehet UML-ből forráskódot generálni. (Osztálydiagram megmondja, milyen függvények vannak benne. Ebből forráskódot, vázat generálni könnyű.)

## Tervminták

Önállóan tervezni hatékony, de újrafelhasználható tervhez ötleteket kitalálni nehéz.

A régebben már bevált tervek alapján könnyebb dolgozni.

Ilyen tervek megismeréséhez, elsajátításához sok gyakorlat kell.

Ezek a tervrészletek a tervminták. Prototípus tervminta.

Tervezéskor hasonlóan használhatóak, mint programozás közben az újrafelhasználható kódrészletek.

Erről félév közepén lesz egy teljes előadás.

## UML tervezőeszközök

* ArgoUML
* StarUML
* Umbrello (Én ezt használom)
* PowerDesigner
* Microsoft Visio
* Visual Paradigm
* (Eclipse UML Tools)
* (NetBeans)

## JAVA fejlesztőeszközök és környezetek

* JAVA RTE (Futtatáshoz)
* JDK (Fordításhoz)
* JDCOCS (Dokumentáció)
* NetBeans (Ezt használjuk)

Egyéb programok:

* Eclipse (Ezt is használom)
* JDeveloper
* JCreator
* JBuilder
* IntelliJ IDEA
* stb.

## NetBeans tippek és trükkök

Majd fent lesz a neten ez a dia. TODO copy/paste