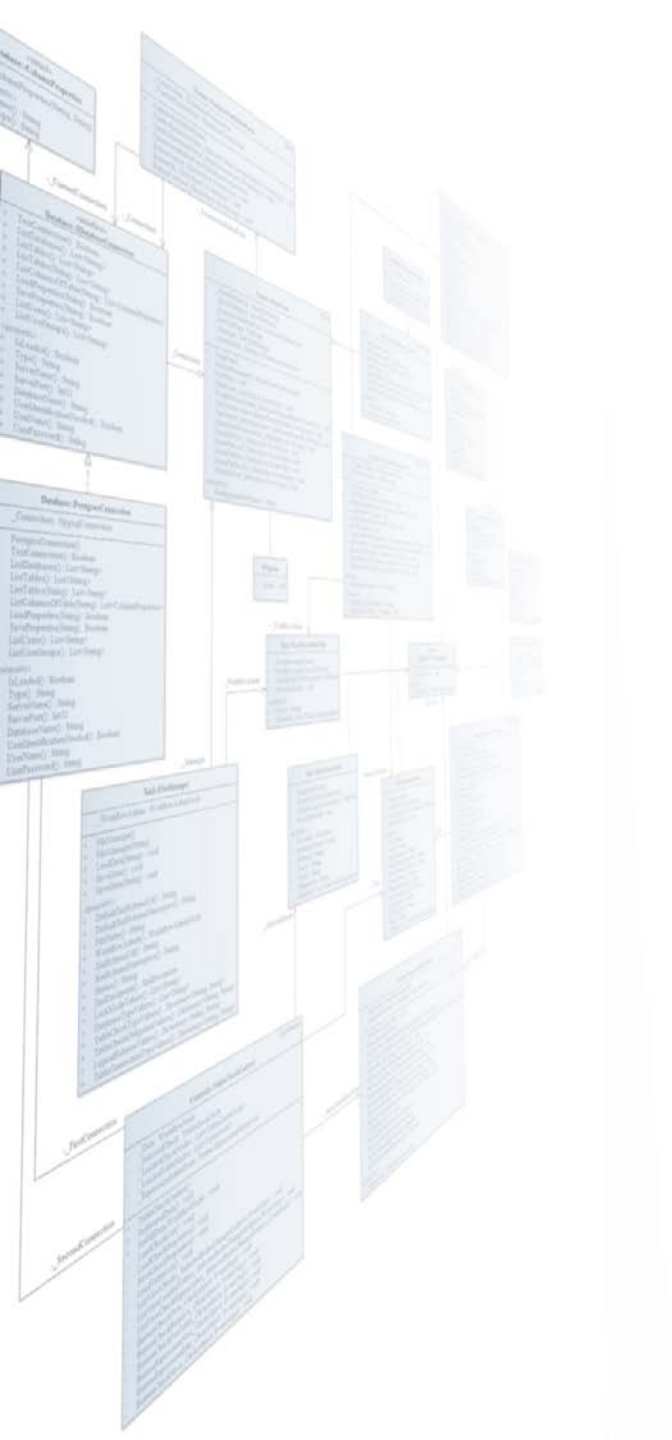


Pázmány Péter Katolikus Egyetem
Információs Technológiai és Bionikai Kar

Bevezetés a Programozásba II

1. előadás

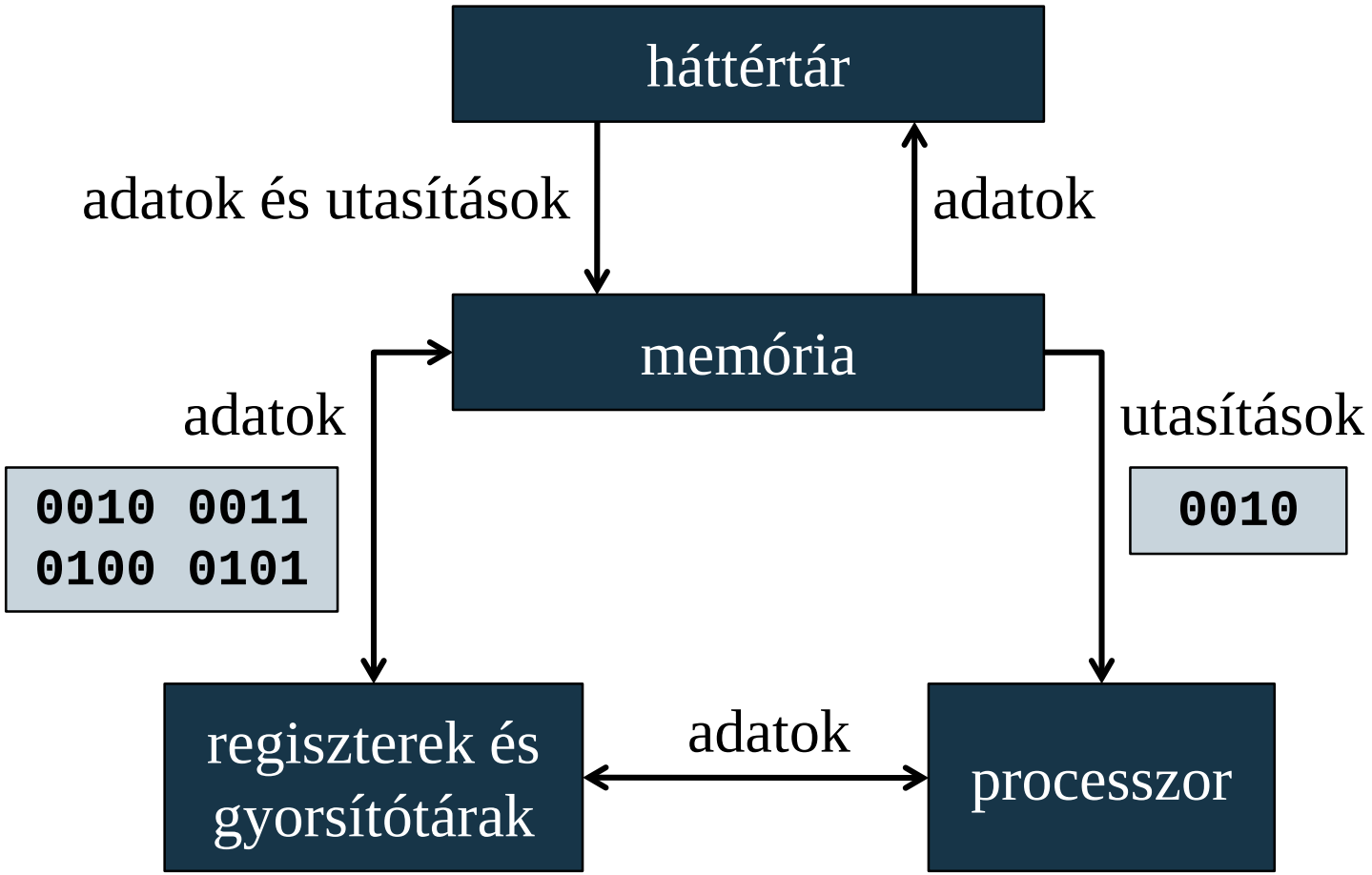
Szoftverfejlesztés, programozási paradigmák

© 2014.02.10. Giachetta Roberto
groberto@inf.elte.hu
http://people.inf.elte.hu/groberto

A program

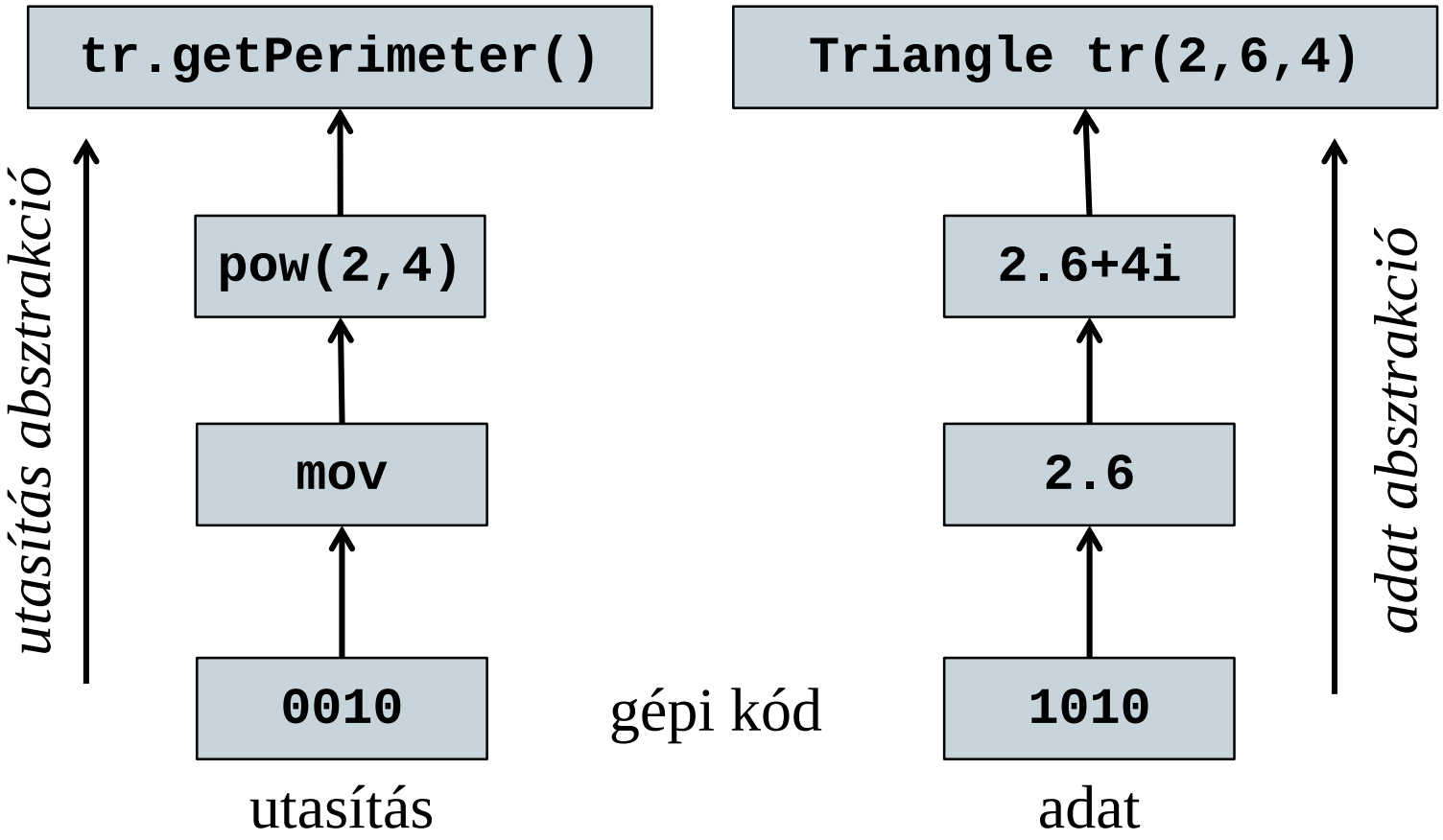
- A *program*
  - *matematikailag*: állapotterek (értékek direktszorzata) felett értelmezett reláció
  - *informatikailag*: utasítások sorozata, amelyek műveleteket hajtanak végre a megadott értékekkel, az *adat*okkal
- A programban foglalt utasítássorozatot, vagy *programkód*ot a *processzor* (CPU, GPU, …) hajtja végre
  - a processzor korlátozott utasításkészlettel rendelkezik, ezért összetett utasításokat nem képes véghezvinni
  - a végrehajtáshoz segédtárakat (regiszterek, gyorsítótárak) használ, és kommunikál a *memória*val

A program


háttértár
adatok és utasítások
adatok
memória
adatok
0010 0011
0100 0101
utasítások
0010
regiszterek és
gyorsítótárak
adatok
processzor

- A processzor által értelmezhető utasításkészletet és adathalmazt nevezzük *gépi kód*nak (*object code*)
- A programokat ennél magasabb szinten kell elkészíteni, ezért szükségünk van a működés és az adatkezelés absztrakciójára:
  - az *utasításabsztrakció* (*control abstraction*) biztosítja, hogy a processzor egyszerű, egymást követő utasításai (összeadás, mozgatás, összehasonlítás) mellett összetett parancsokat és vezérlési módszert fogalmazzunk meg
  - az *adatabsztrakció* (*data abstraction*) lehetővé teszi, hogy különböző típusokba soroljuk adatainkat, amelyek meghatározzák az értéktartományt, és a végezhető műveleteket

Absztrakció a szoftverfejlesztésben



Absztrakció a szoftverfejlesztésben

- Az absztrakció előnyei:
  - összetett funkciók könnyebben megfogalmazhatóak
  - el lehet vonatkozni a konkrét megvalósítástól (pl. hardver)
  - a fejlesztési idő jelentősen csökken
- Az absztrakció hátrányai:
  - összetettebb utasításkészletet kell elsajátítani
  - az egyes utasítások végrehajtása nem áttekinthető
  - az absztrakciót jól, körültekintően kell megvalósítani, különben minden rá épülő tevékenység hibás lesz, illetve mellékhatásai lehetnek

- Az absztrakciót megvalósító eszközt nevezzük *programozási nyelv*nek
  - egy adott programozási nyelven megírt programkódot nevezünk a program *forráskód*jának (*source code*)
  - a programozási nyelv meghatározza az absztrakció szintjét, a használható típusok és utasítások halmazát, amely egy adott nyelvre rögzített, ám a programozó által általában kiterjeszthető
  - a nyelvet meghatározza a célja, vagyis milyen feladatkörre alkalmazható, továbbá a nyelv rendelkezik egy *kifejezőerő*vel, azaz milyen összetett számításokat képes kifejezni

- A programozási nyelvek osztályozása:
  - *alacsony szintű* (assembly): a gépi kódot egyszerűsíti szövegszerűre, de nem biztosít absztrakciót, pl.:

```
data segment ; adatok
   number dw -5 ; változó létrehozása
data ends
code segment ; utasítások
   …
   mov ax, number ; regiszterbe helyezése
   cmp ax, 0 ; regiszterérték összehasonlítása
   jge label1 ; ugrás, amennyiben nem negatív
   mov cx, 0
   sub cx, ax ; pozitívvá alakítás kivonással
   …
```

- *magas szintű*: a gépi architektúrától független utasításkészlettel, nagyobb kifejező erővel rendelkezik
  - lehetőséget ad az utasítás- és adatabsztrakcióra
  - egy egyszerű reprezentációját adja az alacsony szintű kódnak
  - pl.:

```
int main(){
    int number = -5; // változó létrehozása
    if (number < 0) // ha negatív
        number = -number; // ellentettre váltás
    …
}
```

Szoftverfejlesztés

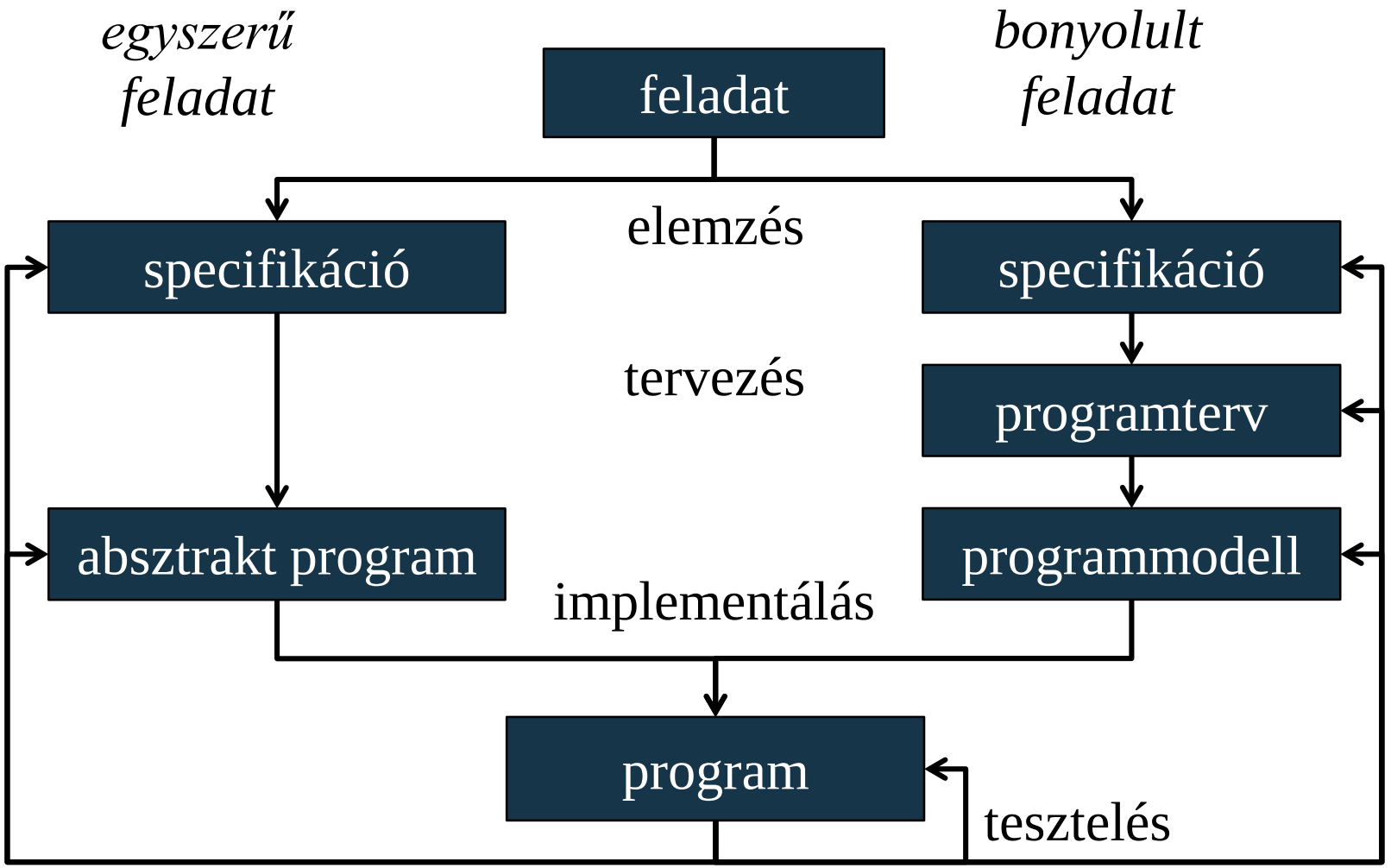
Újrafelhasználhatóság

- Az absztrakció során keletkezett összetett funkciókat alacsonyabb szintű funkciók együttes végrehajtását eredményezi
  - az így keletkezett összetett funkciót hordozni kell, és későbbi alkalmazásokban újrahasználni
- Az *újrafelhasználható programegységek* olyan magas szintű, összetett funkciók, amelyek
  - kellően általánosak és hatékonyak
  - alaposan ellenőrzöttek (teszteltek), hibamentesek
  - általában *programkönyvtár*akba (nyelvi könyvtárba) tömörülnek

- A szoftverfejlesztés a kódoláson túl több lépésből áll, amely függ a feladat bonyolultságától is:
  1. A feladatot elemezni kell, és megadni a formális megfelelőjét, vagyis a *specifikáció*t
  2. A specifikációt alapján megtervezhető a program, amely egyszerű feladatnál az *absztrakt program*, míg bonyolult feladatnál a *programterv* elkészítésével jár, amelyből elállítható a *programmodell* (egyszerűsített célprogram)
  3. A tervet implementáljuk a megfelelő programozási nyelven
  4. Az implementált programot, illetve a programkódot *tesztelés*nek vetjük alá, ami módosításokat eredményezhet az implementációban (vagy a korábbi fázisokban)

Szoftverfejlesztés

A szoftverfejlesztés folyamata



Szoftverfejlesztés

A feladat elemzése és tervezése

- Önmagában a feladat elemzése is nagyban meghatározza a programfejlesztés folyamatát, ennek két formája:
  - *felülről lefelé* (*top-down*): a főfeladatot részfeladatokra, majd azokat további részfeladatokra bontjuk
  - *alulról felfelé* (*bottom-up*): a feladatban szereplő egységeket határozzuk meg, majd azokat kombináljuk
- A tervezés során egy nyelv-független vázát kell elkészítenünk a szoftvernek, amely megadja annak közeli működését
  - lehet formális, vagy informális modell
  - használhat modellező eszközt (pl. stuktogram), vagy nyelvet (pl. UML)

- A *tesztelés* annak ellenőrzése, hogy a program teljesíti-e az előírt minőségi mutatókat, célja elsősorban a futási idejű hibák, működési rendellenességek keresése
  - a tesztelés módja szerint lehet:
    - *fekete doboz tesztelés*: a tesztelendő programrész ismeretlen, csak a hiba voltát fedezzük fel
    - *fehér doboz tesztelés*: a programrész teljes mértékben ismert, tehát így a hiba helyét is megtalálhatjuk
    - *szürke doboz tesztelés*: a belső adatok, illetve egyes algoritmusok elérhetőek, így a hiba helye behatárolható
  - a tesztelés módszere lehet *statikus* (kód kiértékelés és ellenőrzés), illetve *dinamikus* (futtatás adott tesztesetekkel)

Szoftverfejlesztés

A szoftverfejlesztés optimalizálása

- A szoftverfejlesztés során a legfőbb cél, hogy a kész program megfeleljen a *funkcionális* és *minőségi követelmények*nek
  - emellett, a fejlesztők számára fontos, hogy a kész szoftver fejlesztése a lehető legoptimálisabb legyen
- A szoftverek tervezésének és programozásának módszerét nevezzük *programozási paradigmá*nak
  - meghatározza a programozási stílust, az absztrakciós szintet
  - meghatározza az alkalmazható programozási nyelvek körét is, és fordítva
  - sok programozási nyelv több paradigmát is támogatnak, ezek a *multiparadigma* nyelvek

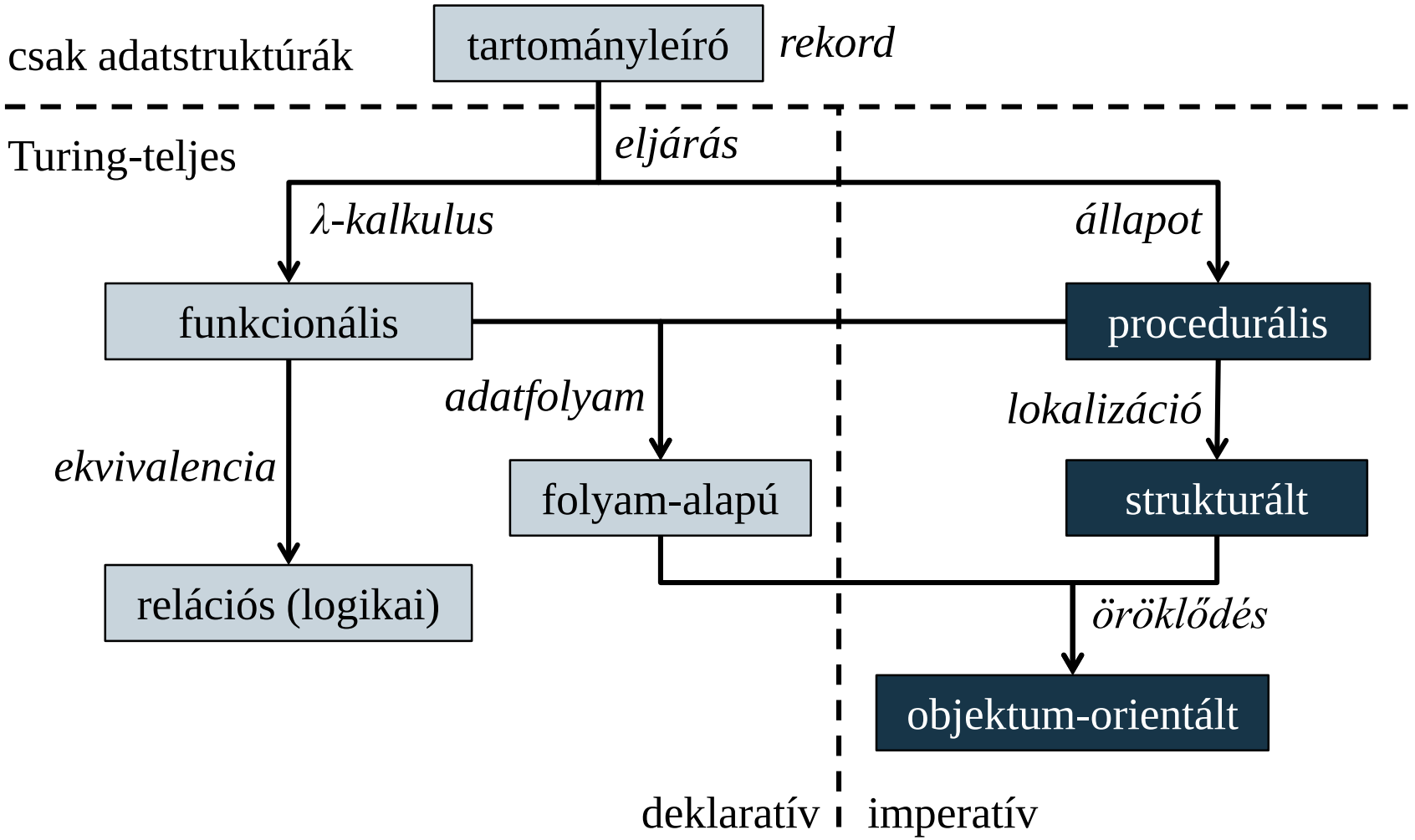
Programozási paradigmák

A paradigmák csoportosítása

- A programozási paradigmákat két csoportba soroljuk:
  - *imperatív*: a programot állapotváltozások sorozatával írja le
    - az utasításokat szekvenciálisan hajtja végre
    - megfelel a gépi szintű végrehajtásnak
  - *deklaratív*: a program a tartalmát, megjelenését írja le, nem pedig a funkció végrehajtásának módját
    - nem alkalmaz változókat, csak konstans értékeket
    - az utasításokat nem feltétlenül szekvenciálisan hajtja végre, automatikusan párhuzamosít
    - magasabb szintű kifejezőerővel rendelkezik

Programozási paradigmák

A jelentősebb paradigmák



Programozási paradigmák

Procedurális programozás

- *Procedurális* (*Procedural*):
  - a programot *alprogram*okra (*subroutine*) bontja, és minden alprogram meghatározott részfeladatot végez el
  - az alprogramoknak két típusa lehet:
    - *eljárás* (*procedure*): valamilyen utasítássorozatot futtat, végeredmény nélkül
    - *függvény* (*function*): valamilyen matematikai számítást végez el, és megadja annak eredményét
  - az alprogramok programkonstrukciókkal épülnek fel, meghívhatnak más alprogramokat, és kommunikálhatnak velük

Programozási paradigmák

Procedurális programozás

- a vezérlést a főprogram szolgáltatja, az kezeli a teljes programban jelen lévő adatokat
- nyelvek: *Fortran*, *C*, *BASIC*, *Pascal*

- pl. (C++, vektor összegzése függvénnyel):

```
int Sum(vector<int> values){
   // a függvény paraméterben megkapja a vektort
   int sum = 0;
   for (int i = 0; i < value.size(); i++)
      sum = sum + values[i];
   // ciklussal hozzávesszük ez elemeket
   return sum; // visszatérési érték az összeg
}
```

Programozási paradigmák

Strukturált programozás

- *Strukturált* (*Structured*):
  - a program részegységekre (*csomag*okra, vagy *blokk*okra) tagolódik, minden egység rendelkezik egy belépési ponttal, és egy kilépési ponttal
  - a programegységeknek van egy kívülről látható része (*interfész*), és egy belső megvalósítása (*implementáció*)
  - a programban használt adatstruktúrák a programegységeknek megfelelően strukturálódnak
  - támogatja a kivételkezelést, tiltja a programkódban történő ugrálást (goto)
  - nyelvek: *Pascal*, *C*, *ADA*

Programozási paradigmák

Strukturált programozás

- pl. (C++, verem típus):

```
class Stack {
private: // rejtett rész, implementáció
   int* values; // attribútumok
   int top;
public: // látható rész, interfész
   Stack(){ values = new int[10]; top = 0; }
   // konstruktor
   ~Stack() { delete[] values; } // destruktor
   void Push(int v) { // metódus
      if (top < 10) { values[top] = v; top++; }
   }
   …
};
```

Programozási paradigmák

Objektum-orientált programozás

- *Objektum-orientált* (*Object-oriented*):
  - a feladat megoldásában az alulról-felfelé megközelítést alkalmazza, alapja az *egységbe zárás* és az *öröklődés*
  - a programot egymással kommunikáló objektumok adják, amelyek valamilyen relációban állnak egymással
  - manapság a legnépszerűbb programozási paradigma, a programozási nyelvek jelentős része támogatja
  - objektumorientált támogatással rendelkező nyelvek: *C++*, *Objective-C*, *Matlab*, *PHP*, *Python*, *Perl*, …
  - tisztán objektumorientált nyelvek: *Smalltalk*, *Java*, *C#*, *Eiffel*, *Ruby*, …

Programozási paradigmák

Objektum-orientált programozás

- pl. (C++, grafikus felületű ablak):

```
class DemoWindow : public QWidget {
    // ablak osztály
public:
   DemoWindow(QWidget* parent = 0) {
      // a konstruktor megkaphatja a szülőt
      setBaseSize(200, 120);
      setWindowTitle("Demo Window");
      qButton = new QPushButton("Quit", this);
      // gomb példányosítása
   }
private:
   QPushButton* qButton; // gomb az ablakon
};
```