Programozási nyelvek és módszerek

OOP II.

OOP fő elvei

OO moduláris struktúra

Adatabsztrakció

Osztályok

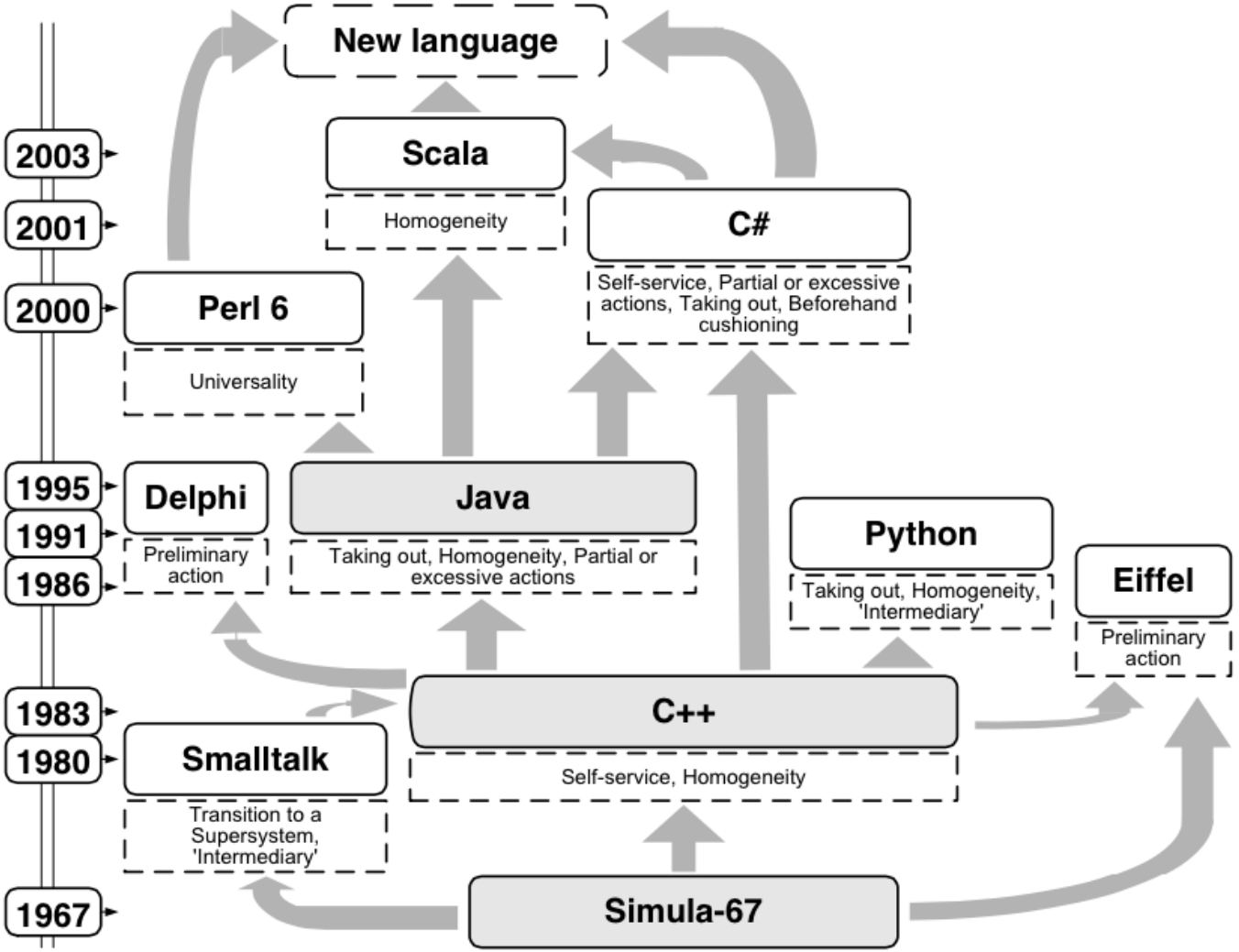
Öröklődés

Polimorfizmus és dinamikus összekapcsolás

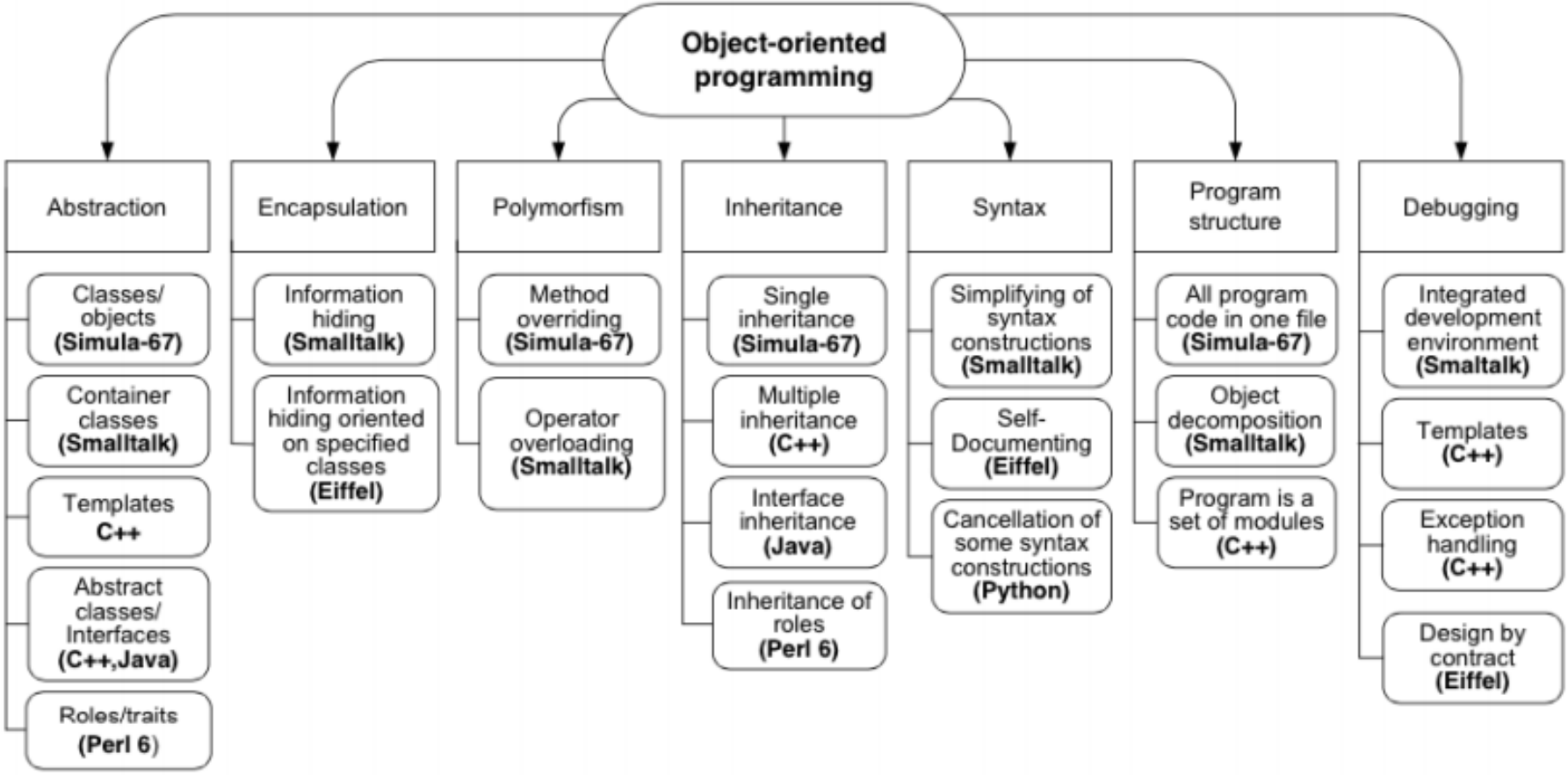
Többszörös és ismételt öröklés

Automatikus memóriakezelés

OOP nyelvek – hatások



OOP nyelvek – jellemzők



Simula-67

Simula

- https://www.tutorialspoint.com/compile_simula_online.php

Itt jelenik meg először:

- Osztály (class)
- Öröklődés
- Osztályhierarchia
- A teljesen önálló típusfogalom
  - objektumhivatkozás: `ref (< osztályazonosító >)`
  - Bármely A osztály egyértelműen meghatároz egy `ref(A)` hivatkozási típust
    - ez minden objektumhoz létezik

Simula-67

Szintaktika: a blokkok prefixelése

- Ennek hatására a blokk és az osztály fogalma összeolvad
- A blokkban az osztályban definiált publikus fogalmakat használhatjuk (SIMULATION)
- Például

```
prefix_obj begin
...
end
```

Simula-67

```
Begin
    class Rectangle(SideA, SideB);
    real sideA;
    real sideB;
    Begin
        integer Procedure GetArea;
        Begin
            GetArea:=sideA * sideB;
        End;
    End of Square;

    Ref (Rectangle) r;
    r :- new Rectangle(10, 20);
    OutFix(r.GetArea, 2, 6);
    OutImage;
End;
```

Simula-67

Öröklés (prefixeléssel)

```
Rectangle class ColorRectangle(Color);
text Color;
Begin
    Procedure Print;
    Begin
        OutText("Rectangle: ");
        OutFix(SideA, 2, 6);
        OutFix(SideB, 2, 6);
        OutText(" "); OutText(Color);
        OutImage;
    End;
End of ColorRectangle;

Ref (ColorRectangle) r;
r :- new ColorRectangle(20, 10, "Zöld");
r.Print;
```

Simula-67

Tulajdonságok

- Van GC
- Nincs többszörös öröklődés
- Van `this` pointer
- `virtual`
  - Függvények felüldefiniálásához
  - Absztrakttá teszi az osztályt, ha nem adunk meg törzset
- `inner`
  - Leszármazott osztály inicializálásakor lefutó kód
- Láthatósági védelem
  - hidden és protected prefixek az attribútumok előtt

Simula-67 Példakód

```
Begin
    class Shape;
    Virtual:
        Procedure Print is Procedure Print;;
        Procedure GetArea is integer Procedure GetArea;;
        ! Virtuális függvények;
    Begin
        OutText("Shape - Normal"); OutImage;
        ! Ez a Shape inicializálásakor fut le
    Inner;
        OutText("Shape - Inner"); OutImage;
        ! Ezek a leszármazottakhoz tartozó utasítások;
    End of Shape;
```

Simula-67 Példakód

```
Shape class Rectangle(SideA, SideB)
real sideA;
real sideB;
Begin
    integer Procedure GetArea;
    Begin
        GetArea:=sideA * sideB;
    End;
    Procedure Print;
    Begin
        OutText("Rectangle: ");
        OutFix(SideA, 2, 6); OutFix(SideB, 2, 6);
        OutImage;
    End;
    OutText("Rectangle - Normal"); OutImage;
    ! Ez a Rectangle inicializálásakor fut le;
End of Rectangle;
```

Simula-67 Példakód

```
    Rectangle class ColorRectangle(Color);
    text Color;
    Begin
        Procedure Print;
        Begin
            OutText("CRectangle: ");
            OutFix(SideA, 2, 6); OutFix(SideB, 2, 6);
            OutText(" "); OutText(Color); OutImage;
        End;
    End of ColorRectangle;

    Ref (Shape) r;
    r :- new ColorRectangle(20, 10, "Zöld");
    r.Print;
    OutFix(r.GetArea, 2, 6);
    OutImage;
End;
```

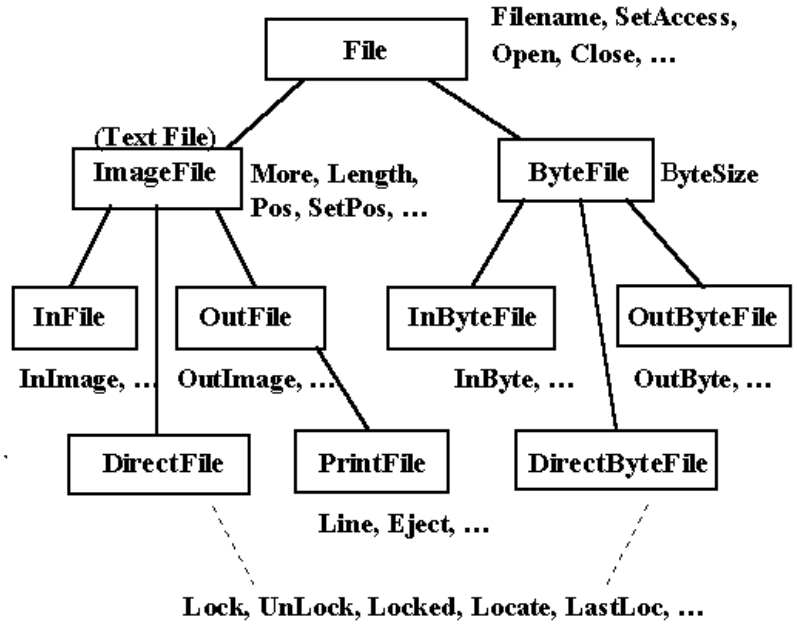
Simula-67

Az `inspect` utasítás a dinamikus összekapcsolás megvalósítása mellett:

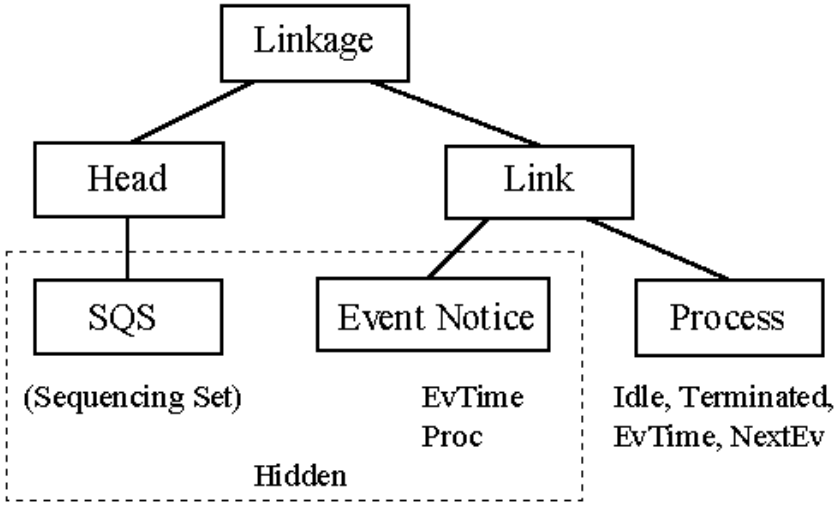
```
X in osztály_1

inspect X
  when osztály_1 do utasítás_1
  when osztály_2 do utasítás_2
  ...
  when osztály_n do utasítás_n
  otherwise utasítás_0
```

Simula-67 Standard könyvtárak

BASICIO, FILE



SIMULATION, PROCESS



SmallTalk

SmallTalk
- https://www.tutorialspoint.com/execute_smalltalk_online.php

Minden objektum
- Integer,…Character
- Az üzenetek, amelyeket az objektumoknak küldünk
- Az IDE is: ClassBrowser, Workspace, Debugger

A ST szintaxisa 1 nap alatt megtanulható
- Jó részét egy perc alatt:
  - `object message`
  - Példák
    - `myWindow drawCircle,`
    - `myAge + 1,`
    - `myWindow drawCircleofRadius: afloat, color: Red`

SmallTalk

Tulajdonságok

- RTTI (run-time type information)
  - Egy változóról nem tudjuk, hogy milyen típusú objektumra mutat, a változók típusnélküliek
- Szabványos objektumkönyvtárak
  - Array
  - String
  - File Stream
  - …
- VM
- GC

SmallTalk

Legnagyobb közös osztó keresése:

```
| a b |
a := 10.
b := 12.

[ a ~= b ] whileTrue:
    [ a < b ifTrue:
            [ b := b - a ]
        ifFalse:
            [ a := a - b ]
    ].

Transcript show: (a
printString).
^a
```

Magyarázat

ciklus: logikai értéket visszaadó blokk objektumnak küldjük

- a `whileTrue:` üzenetet, aminek az argumentuma is egy blokk

elágazás: logikai értéknek küldjük

- az `ifTrue: ifFalse:` üzenetet, blokkok az argumentumok

SmallTalk

Minden objektum

- Minden objektum egy osztály példánya

Az osztály is objektum

- Kérdés: ez minek a példánya?
- Válasz: egy metaosztály példánya
  - (metaosztály hierarchia)

SmallTalk

Az objektumok osztályát a class üzenettel kapjuk vissza

Osztályok definiálása a szülőnek küldött üzenettel történik (subclass)

Nincs többszörös öröklődés

- nincs anomália

Láthatóság

- nincsenek láthatósági predikátumok
- a belső változók minden alosztály számára láthatóak
- De kívülre nem, viszont a metódusok globálisak
  - konvenció: a megjegyzésbe private-t írhatunk

Van osztályszintű attribútum és metódus

SmallTalk

Közös ősosztály
- Object
  - objektum kiíratása
  - objektum osztályának meghatározása
  - copy
  - klónozás

Konstruktor
- osztálynak küldött üzenet

Destruktor nincs
- szemétgyűjtéssel oldja meg

Nincs absztrakt osztály, azaz a metódusokat definiálni kell, nem csak deklarálni

SmallTalk

Osztály létrehozása

```
Object subclass: #Square
    instanceVariableNames: 'sidea'
    classVariableNames: 'count'
    poolDictionaries: ''
    category: nil.
```

SmallTalk

Osztálymetódus – példányosításhoz

```
Square class extend [
    new [
        <category: 'instance creation'>
        | r |
        r := super new.
        r init.
        ^r
    ]
]
```

SmallTalk

Példánymetódusok (inicializáláshoz)

```
Square extend [
    init [
        <category: 'initialization'>
        sidea := 0.
    ]

    init: side [
        <category: 'initialization'>
        sidea := side.
    ]
```

SmallTalk

Példánymetódusok (alap funkciók)

```
    printOn: stream [
        <category: 'printing'>
        super printOn: stream.
        stream nextPutAll: ' with side A = '.
        sidea printOn: stream
    ]

    getArea [
        ^ sidea*sidea.
    ]
]
```

SmallTalk

Próba

```
S := Square new.
S init: 10.
S printOn: stdout.
Transcript cr.
S getArea printOn: stdout.
```

Object Pascal

Többféle következetlenség

- `object – class`
- `virtual` (sebesség) - `dynamic` (méret)
- Láthatósági megkötések alkalmazhatóak – de fájlon belül nem érvényesek!
  - De: strict private, strict protected lehetőség is van

Új láthatóság: published (publikált)

- ugyanazon elérési szabályok, mint a public részekre, de a fordító futási idejű típus-információkat is kapcsol hozzá

Objektumok attribútuma

- `property p: word read olvas write ír default 5;`

class kulcsszó

Automatikus memóriakezelés

- A Delphi referencia modellt használ az osztályok esetén
  - Az ilyen objektum mindig dinamikus, implicit referencia.

Közös ős

- Minden osztály a `TObject` ősosztály leszármazottja.

Absztrakt metódusok

- Ha a metódus deklarációjában az `abstract` kulcsszót használjuk, nem kell törzset megadni – de az ilyen osztálynak is lehet példánya!

Object Pascal

```
type
  TShape = class
  public
     constructor Create;
     function Print: string; virtual; abstract;
     function GetArea: Single; virtual; abstract;
  end;

type TRectangle = class (TShape)
  private
     sideA: Single;
     sideB: Single;
  public
    constructor Create;
    constructor Create(side_A: Single; side_B: Single);
    function Print: string; override;
    function GetArea: Single; override;
  end;
```

Object Pascal

```
constructor TShape.Create;
begin
    WriteLn('Shape Create')
end;
constructor TRectangle.Create;
begin
     self.Create(0.0, 0.0)
end;
```

Object Pascal

```
constructor TRectangle.Create(side_A: Single; side_B: Single);
begin
     self.sideA := side_A;
     self.sideB := side_B;
end;
function TRectangle.Print: string;
begin
    Result := 'Rectangle: ' + FloatToStr(self.sideA) + ' ' +
FloatToStr(self.sideB);
end;
function TRectangle.GetArea: Single;
begin
    Result := self.sideA * self.sideB;
end;
```

Object Pascal

```
var
  s: TShape;
  r: TRectangle;

begin
  s := TRectangle.Create(10, 20);
  WriteLn(s.GetArea);

  ReadLn();
end.
```

Object Pascal – Ős konstruktora

```
type TOs = class
    a, b: Integer;
    constructor Letrehoz(x, y: Integer);
end;
type TUj = class(Tos)
    c: Integer;
    constructor Letrehoz(x, y, z: Integer);
end;
constructor TOs.Letrehoz(x, y: Integer);
begin
    a:=x; b:=y;
end;
constructor TUj.Letrehoz(x, y, z: Integer);
begin
    inherited Letrehoz(x, y); c:=z;
end;
```

Object Pascal – virtuális konstruktor

```
TFoo = class
 public
  constructor Create; virtual;
 end;
TBar = class(TFoo);
 public
  constructor Create; override;
 end;
```

```
TFooClass = class of TFoo;
 // class reference type
 var
  fc: TFooClass;
  afoo: TFoo;
 begin
  fc := TFoo;
  afoo := fc.Create;
  // returns a TFoo
  fc := TBar;
  afoo := fc.Create;
  // returns a TBar
```

Object Pascal – sealed osztály

sealed – nem származtatható belőle

```
type
  TMyClass = class sealed
    procedure SomeProcedure;
end;
```

final – ha metódus

```
procedure MyMethod(const AMyParameter: Integer);
override; final;
```

Object Pascal – interfészek

```
type
IDrawable = interface
    procedure draw();
end;
```

Az interface-ben:

- csak a metódusok fejrésze van leírva
- tartalmazhat property-ket is, de ezek szintén nincsenek kidolgozva, csak a property neve van megadva, típusa, és az, hogy írható vagy olvasható,
- mezőket, konstruktort, destruktort nem tartalmazhat
- minden rész az interface-ben automatikusan publikus,
- a metódusokat nem lehet megjelölni virtual, dynamic, abstract, override kulcsszavakkal,
- az interface-eknek lehetnek ősei, melyek szintén interface-ek.

Objective-C

Objektumorientált nyelv

- a C programozási nyelvet egészíti ki
- a Smalltalk üzenetküldési lehetőségeivel.

1986-ban jelent meg, Brad Cox és Tom Love tervezték

Apple által továbbfejlesztett és támogatott

- Mostanra megvan az utódja is …

Objective-C

Osztályok

- Kötelező szétválasztani az interfészt és az implementációt.
- Az interfész megadja a specifikációt
  - Deklarálja az osztály adattagjait, metódusait
  - Megnevezi az ősosztályát.
- Az implementációban pedig definiáljuk a metódusokat
  - Azaz az osztályt magát adjuk meg.

Objective-C

Az osztály interfész része

```
@interface ClassName : ItsSuperclass
{
      float width;
      float height;
      BOOL filled;
      NSColor *fillColor;
      //...
}   //itt az adattagok
      + alloc;          // osztálymetódus előtt +
      - (void)display; // példánymetódus előtt –
      - (void)setWidth:(float)width height:(float)height;
      - makeGroup:group, ...;
@end
```

Objective-C

Az osztály implementáció része

```
#import "ClassName.h„
@implementation ClassName
    + alloc
    {           //...
    }
    - (void)display
    {           //...
    }
    - (void)setWidth:(float)width height:(float)height
    {           //...
    }
    - makeGroup:group, ...
    {
       va_list ap;
       va_start(ap, group);
       //...
    }
@end
```

Objective-C – üzenetküldés

Itt a metódusok/függvények hívása helyett az objektumok közötti üzenetküldés kerül előtérbe

- `[foo bar:parameter];`

Az hogy az üzenetet fel tudja-e dolgozni az objektum, az futásidőben dől el

- Ahogyan a típusellenőrzés sem történik meg, csak futásidőben

Az üzenetküldés során a paraméterek a függvény nevével adhatók meg

```
-(type)method:(type)param1 andParam2:(típus)param2;
```

Objective-C

Láthatóság szabályozása a következő direktívákkal lehetséges

- `@public`
- `@private`
- `@protected`
  - Alapértelmezett
- `@package`

Objective-C

Öröklődés

- Egyszeres öröklődés
- Nincs absztrakt osztály
- `NSObject` a legfelső szinten

Interface

- `@protocol`

Kategóriák

- Metódusok egy halmaza, amelyet „csomagban” lehet egy osztályhoz hozzáadni.
- Nem azonos az interfésszel, vagy öröklődéssel.

polimorfizmus, dinamikus kötés – automatikusan

Objective-C

```
@protocol Archiving
- read: (FILE *) f;
- write: (FILE *) f;
@end

@protocol ReferenceCounting
- incrementCount;
- decrementCount;
- (unsigned) refCount;
@end

@interface Shape : NSObject <Archiving, ReferenceCounting>
```

Objective-C

Kategóriák

- A kategória segítségével már meglévő osztályokhoz adhatunk hozzá metódusokat
  - akkor is, ha nincs meg az osztály forrása
- Ez egy rendkívül erős eszköz, amellyel öröklés nélkül terjeszthetjük ki az osztályok funkcionalitását
  - akár beépített osztályokét is
- Az új függvények az osztály meglévő interfészét egészítik ki
  - mixing -> mixin-s

Objective-C

```
#import "ClassName.h"
@interface ClassName ( CategoryName )
  // method declarations
@end

#import "ClassName+CategoryName.h"
   // ilyen neve legyen!
@implementation ClassName ( CategoryName )
  // method definitions
@end
```

Objective-C

Példa – NSString minden objektumának legyen isURL metódusa:

```
@interface NSString (Utilities)
- (BOOL) isURL;
@end

#import "NSString+Utilities.h"
@implementation NSString (Utilities)
- (BOOL) isURL
{
      if ( [self hasPrefix:@"http://"] )
            return YES;
      else
            return NO;
}
@end
```

Objective-C

Most már bármelyik NSString-re alkalmazhatjuk ezt a metódust0

```
NSString* string1 = @"http://pixar.com/";
NSString* string2 = @"Pixar";

if ( [string1 isURL] )
  NSLog (@"a string1 egy URL");
if ( [string2 isURL] )
  NSLog (@"a string2 egy URL");
```

Java

Smalltalk-kal rokonság

GC (VM dolga)

Objektumelvűség

- A Javában gyakorlatilag minden objektum (osztály)
- Primitív típusokhoz csomagoló („wrapper”) osztály tartozik
  - **`boolean`**, **`char`**, **`byte`**, **`short`**, **`int`**, **`long`**, **`float`**, **`double`**
  - Felelős a konverzióért, kiíratásért, stb.
    - immutable osztályok
    - **`boolean`** -> `Boolean` (nagybetű a különbség)
    - **`char`** -> `Character`
    - **`int`** -> `Integer`
  - Automatikus konverzió: boxing-unboxing

Java

Nincs osztályon kívüli (globális) változó

Öröklődési fa gyökere az `Object` osztály

Nincs többszörös öröklődés

- Interfészekkel részben helyettesíthető
  - Minden függvénye publikus és absztrakt
  - Konstans változókat deklarálhatunk benne
  - Többszörös interface öröklés engedélyezett
- **`extends`** – **`implements`**

`instanceof`

- új operátor a C++ -hoz képest

Java

Módosító kulcsszavak

- **`final`**
  - Osztálynév előtt: tiltja a további származtatást
  - Metódus előtt: tiltja a felüldefiniálást
  - Változó előtt: tiltja a kezdeti értékadás után az érték megváltoztatását
    - Objektum esetén referencia
- **`abstract`**
  - Metódus előtt: a metódus deklarálható, de nem definiálható az adott osztályban, csak a származtatottban
  - Osztály előtt: az osztálynak van **`abstract`** metódusa (nem kötelező kiírni, de az osztály akkor is **`abstract`** lesz implicite)
- **`static`**
  - Osztálymetódus, illetve osztályváltozó jelzésére
- Egy osztály nem lehet egyszerre **`final`** és **`abstract`**

Java láthatóság

Hol		Kulcsszó			
Package	Osztály	public	protected	no modifier default	private
Azonos csomag	Az osztály maga	X	X	X	X
	Belső osztály	X	X	X	X
	Leszármazott	X	X	X	
	Bárki	X	X	X	
Másik csomag	Leszármazott	X	X		
	Bárki	X			

Java – inicializálás

Nincs operátor túlterhelés

- De függvény túlterhelés van

Ha egy osztálynak nincs konstruktora, a fordító a következőt generálja neki:

```
class MyClass extends SuperClass {
      MyClass() {  super();  }
}
```

Az ősosztály konstruktora a leszármazott lefutása előtt történik

- Nem kell expliciten megadni, ilyenkor az azonos paraméterezésűt futtatja

Java – inicializálás

Lehetséges inicializációkat is beszúrni a kódba

- A mezőknél kezdeti érték adásával
- Blokkokkal

Lefutási sorrend

- Ősosztály statikus inicializációja
- Leszármazott osztály statikus inicializációja
- Ősosztály inicializációs kódja(i)
- Ősosztály konstruktora
- Leszármazott osztály inicializációs kódja(i)
- Leszármazott osztály konstruktora

Java – finalize()

Destruktorok: a nyelvben nem szerepelnek

- helyette a `finalize()`
- a GC hívja felszámolás előtt

Java

```
public abstract class Shape {
    public abstract double calculateArea();
    public abstract void print();
}

public interface Polygon {
    double[] getEdges();
}
```

Java

```java
public class Circle extends Shape{
    public static final double PI = 3.1415926535;

    private double radius;

    public Circle(double r) {
        this.radius = r;
    }

    @Override
    public void print() {
        System.out.println("Circle, radius: " + this.radius);
    }

    @Override
    public double calculateArea() {
        return this.radius * this.radius * Circle.PI;
    }
}
```

Java

```
public class Parallelogram extends Shape implements Polygon {
    protected double _side_a;
    protected double _side_b;
    protected double _delta;
    protected double _height_a;

    public double getSideA() { return this._side_a; }
    public double getSideB() { return this._side_b; }
    public double getAngle() { return this._delta;  }

    public boolean isEquilateral() { return
                                     this._side_a==this._side_b; }
    public boolean isRight()  { return this._delta == 90; }

    public Parallelogram(double a, double b, double angle) {
        this._side_a = a;
        this._side_b = b;
        this._delta = angle;
        this._height_a = b *
                         Math.sin(this._delta * Math.PI / 180.0);
    }
```

Java

```
    @Override
    public void print() {
        System.out.println("Parallelogram a, b, delta: "
                + this._side_a + ", "
                + this._side_b + ", "
                + this._delta);
    }

    @Override
    public double calculateArea() {
        return this._side_a * this._height_a;
    }

    @Override
    public double[] getEdges() {
        return new double[] {_side_a, _side_b, _side_a, _side_b};
    }
}
```

Java

```
public class Main {

    public static void main(String[] args) {
        Rectangle r = new Rectangle(10, 30);
        Shape s = r;
        s.print();
        System.out.println(s.calculateArea());

        Polygon p = r;
        System.out.println(p.getEdges().length);
    }
}
```

Java – default

```
public interface Addressable
{
    String getStreet();
    String getCity();

    default String getFullAddress()
    {
        return getStreet()+", "+getCity();
    }
}
```

Java – default

```
public class Letter implements Addressable
{
    private String street;
    private String city;
    public Letter(String street, String city)
    {
        this.street = street;
        this.city = city;
    }
    @Override
    public String getCity() { return city; }
    @Override
    public String getStreet() { return street; }

    public static void main(String[] args)
    {
        Letter l = new Letter("123 AnyStreet", "AnyCity");
        System.out.println(l.getFullAddress());
    }
}
```

Java – többszörös implementáció

```
public interface Interface2 {

    default void doIt(){
        System.out.println("I2.doIt");
    }
}
public interface Interface1 {

    default void doIt(){
        System.out.println("I1.doIt");
    }
}
```

Java – többszörös implementáció

```
public class TheClass
        implements Interface1, Interface2 {

    public void doIt()
    {
        Interface2.super.doIt();
        Interface1.super.doIt();
    }
}
```

Java – Annotációk

Metaadatokat lehet a kódhoz fűzni

- Információ a fordítónak
  - Például felüldefiniálás esetén
    - `@Override`
  - Figyelmeztetések ignorálására
  - …
- Deployment-nél felhasználható adatok
- Futásidőben használható adatok, ellenőrzésre
  - `myString = (@NonNull String) str;`

C#

OOP támogatása nagyon hasonlít a Javáéhoz

- Egyszeres öröklődés
- Interfészek (itt lehet többszörös öröklődés)

Eltérések, újdonságok

- statikusan és dinamikusan kötött metódusok is megadhatók
  - `virtual` kulcsszóval meg kell jelölni a felüldefiniálható függvényt, különben elfedi a leszármazott
  - Felüldefiniált metódust jelölni kell: `override`
- A közvetlen ősosztályra a `base` kulcsszóval hivatkozhatunk
- A `sealed` kulcsszóval megakadályozhatjuk, hogy
  - egy osztályból származtassanak
  - egy metódust felüldefiniáljon a származtatott osztály

C#

Eltérések, újdonságok

- Egy metódus lehet `sealed override`
  - Ekkor a leszármazott nem definiálhatja felül
  - De elfedheti, ezt jelölnie kell a `new` kulcsszóval
    - Polimorfizmus esetén vigyázni kell!
- Az `external` módosítóval rendelkező metódusok valamilyen más nyelven vannak implementálva.
  - Éppen ezért a metódus törzsében csak egy pontosvessző áll.
  - Az ilyen metódus nem lehet `abstract`.

C#

Konstruktorok

- Közvetlenül a konstruktor törzsének végrehajtása előtt automatikusan történik egy másik konstruktor hívás is
  - Ezt nevezik konstruktor inicializálásnak is.
- Két lehetőségünk van
  - Az adott osztály egy másik konstruktorát hívjuk meg először.
  - Meghívjuk az ős valamelyik példány konstruktorát
    - Ha nem használjuk egyiket sem, akkor az ős alapértelmezett konstruktora hívódik meg.
- Azaz a következő két deklaráció ekvivalens egymással:

```
C( ... ) {...}
C( ... ): base() {...}
```

C#

Konstruktorok

- Ha egy osztálynak nem deklarálunk semmilyen példány konstruktort, akkor egy default konstruktor jön létre
  - E az alapértelmezett konstruktor inicializátorral fog rendelkezni.
- Készíthetünk `private` konstruktorokat is
  - Ebben az esetben az osztály nem példányosítható és nem lehet örököltetni sem belőle.
  - Akkor célszerű ezt használni, ha például csak statikus elemeket tartalmaz az osztályunk.
- Statikus konstruktorok
  - A konstruktor neve előtt a `static` kulcsszót kell használni.
  - A statikus konstruktorok az osztály inicializálásakor futnak le, amikor az osztály betöltődik.
  - Ezekre a konstruktorokra nem lehet hivatkozni és nem is öröklődnek.

C#

Destruktorok

- Az objektumok megsemmisülésekor hívódnak meg.
  - Automatikusan hívódnak meg, a GC miatt
  - Az öröklődési láncon végighaladva egymás után hívódnak meg.
    - A szemétgyűjtő nyilvántartja azon objektumok listáját, amelyek osztályában definiáltunk destruktort.

```
~MyClass(){}
```

  - A fordító ezt generálja belőle

```
protected override void Finalize()
{
  try {
   …
  }
  finally {
    base.Finalize();
  }
}
```

C#

Szemétgyűjtés felülbírálható az `IDisposable` interfész megvalósításával

- Ekkor kell egy `Dispose()` metódus.
- Nem lehetünk biztosak benne, hogy a felhasználó megbízhatóan hívja…
- A using utasítást vezették be, hogy biztosan lefusson.

C#

Property

- Definiálhatók elérők (accessor), melyek tulajdonságként viselkednek, de az írás (set) illetve olvasás (get) számára külön eljárások írhatók.
- Például, ha egy elérőre csak a get eljárást definiáljuk, akkor az elérő egy csak olvasható attribútumként fog viselkedni.

C#

Absztrakt osztályok

- Az `abstract` kulcsszó használatával azt jelezhetjük, hogy az adott osztály még nincs teljesen megvalósítva.
- Egy absztrakt osztály nem lehet `sealed`.
- Egy absztrakt metódus
  - Automatikusan `virtual`
  - Nem lehet `static`
- A származtatott osztály tartalmazza az absztrakt metódusok implementációt
  - Nem absztrakt ősmetódusra tud hivatkozni a `base` kulcsszóval

C#

Az öröklődés

- `class SubClass: BaseClass {..}`
- Utód osztályban nem lehet szűkebb a hozzáférés, mint ős osztályban
- A `base` kulcsszóval a közvetlen ősosztály metódusa elérhető
- A `virtual` függvényeknél a `new` kulcsszó használatával elfedhetjük az ős `virtual` függvényét
  - Ekkor egyidejűleg nem lehetséges az ős függvény elfedése (`override`)

C#

Interfészek

- Nincs többszörös öröklődés C#-ban, helyette interface-k vannak.
  - Az interfészek között lehetséges a többszörös öröklődés.
- Az interfész definiálhat: (De nem valósíthatja meg)
  - Metódusokat
  - Indexelőket
  - property-ket
  - eseményeket
- Egy interfésznek több más interfész is lehet őse, egy osztály pedig több interfészt is megvalósíthat.
- A interfész tagjaira az `abstract`, `virtual`, `static`, `override` módosítók nem használhatók.

C# – Példa

```
abstract class Shape {
    public abstract double CalculateArea();
    public abstract void Print();
}


interface IPolygon {
    double[] GetEdges();
}
```

C# – Példa

```
class Circle : Shape {
    static double PI = 3.1415926535;
    // const alapértelmezetten

    private double radius;

    public Circle(double r) {
        radius = r;
    }

    public override void Print() {
        Console.WriteLine("Circle, radius: " + radius);
    }

    public override double CalculateArea() {
        return radius * radius * Circle.PI;
    }
}
```

C# – Példa

```
class Parallelogram : Shape, IPolygon
{
    protected double _side_a;
    protected double _side_b;
    protected double _delta;
    protected double _height_a;

    private void UpdateHeightA()
    {
        _height_a = _side_b * Math.Sin(_delta * Math.PI / 180.0);
    }

    public double SideA
    {
        get => _side_a;
    }
```

C# – Példa

```
public double SideB
{
    get { return _side_b; }
    set
    {
        _side_b = value;
        UpdateHeightA();
    }
}

public double Angle
{
    get => _delta;
    set
    {
        _delta = value;
        UpdateHeightA();
    }
}

public virtual bool isEquilateral() { return _side_a==_side_b; }
public virtual bool isRight()  { return _delta == 90; }
```

C# – Példa

```
    public Parallelogram(double a, double b, double angle) {
        _side_a = a;
        SideB = b;
        Angle = angle;
        UpdateHeightA();
    }

    public override void Print() {
        Console.WriteLine("Parallelogram a, b, delta: "
                          + _side_a + ", "
                          + _side_b + ", "
                          + _delta);
    }

    public override double CalculateArea() {
        return_side_a *_height_a;
    }

    public double[] GetEdges() {
        return new double[] {_side_a, _side_b, _side_a, _side_b};
    }
}
```

C# – Példa

```
sealed class Rectangle : Parallelogram
{
    public Rectangle(double a, double b) : base(a, b, 90) {

    }

    public override void Print() {
        Console.WriteLine("Rectangle a, b: "
                          + _side_a + ", "
                          + _side_b);
    }

    public override bool isRight() {
        return true;
    }
}
```

C# – Példa

```
class Program
{
    static void Main(string[] args)
    {
        Parallelogram pa = new Parallelogram(10, 5, 90);
        Console.WriteLine(pa.CalculateArea());
        pa.Angle = 45;
        Console.WriteLine(pa.CalculateArea());

        Rectangle r = new Rectangle(10, 30);
        Shape s = r;
        s.Print();
        Console.WriteLine(s.CalculateArea());

        IPolygon p = r;
        Console.WriteLine(p.GetEdges().Length);
    }
}
```

C# – Függvények öröklésnél

```
public class CA {
    public virtual void f() { }
}

public class CB : CA {
    public override void f() { }    //felüldefiniálja
}

public class CC : CA {
    private new void f() { }        //elrejti, nem virtuális függvénnyel
}

public class CD : CA {
    public new virtual void f() { } // új virtuális függvény!
}

public class CE : CA {
    public sealed override void f() { } //nem lehet többször átdefiniálni
}
```

C# – Interfészek megvalósítás

```
interface I1
{
    void MyFunction();
}

interface I2
{
    void MyFunction();
}

class Test : I1, I2
{
    public void MyFunction()
    {
        Console.WriteLine("Which interface?");
    }
}
```

C#

Tetszőleges statikus típus esetén az egyetlen implementáció fut le:

```
Test t1 = new Test();
t1.MyFunction();
I1 i1 = t1;
i1.MyFunction();
I2 i2 = t1;
i2.MyFunction();
```

Ám adható a két interfész megvalósításához különböző implementációs kód …

C#

```
class Test2 : I1,I2
{
    void I1.MyFunction()
    {
        Console.WriteLine("I1 implemented!");
    }

    void I2.MyFunction()
    {
        Console.WriteLine("I2 implemented!");
    }
}
```

Ekkor a következővel probléma van, mivel nincs rá implementáció:

```
Test2 t1 = new Test2();
t1.MyFunction();
```

C#

Lehetőség van operátor-túlterhelésre is

- csak **public static** lehet

```
public static Digit operator+(Digit a, Digit b)
{
      return new Digit(a.value + b.value);
}
```

C#

Indexelők:

- Definiálhatók indexelők (indexer) is, melyekkel szögletes zárójellel indexelhető osztályok hozhatók létre.
  - Az indexelő paramétere bármilyen típus lehet.
- Bármilyen objektum úgy indexelhető, mint egy tömb.
- Az indexelő deklarálásakor meg kell adni
  - Az elemek típusát
  - A this-t a formális paraméterlistával.
  - Ez a paraméterlista határozza meg az indexelés szintaxisát.

C#

```
class Grid {
  const int NumRows = 26;
  const int NumCols = 10;
  int[,] cells = new int[NumRows, NumCols];
  public int this[char c,int colm]  {
    get
    {
      c = Char.ToUpper(c);
      if (c < 'A' || c > 'Z') throw new ArgumentException();
      if (colm < 0 || colm > = NumCols) throw new IndexOutOfRangeException();
      return cells[c - 'A', colm];
    }
    set
    {
      c = Char.ToUpper(c);
      if (c < 'A' || c > 'Z') throw new ArgumentException();
      if (colm < 0 || colm > = NumCols) throw new IndexOutOfRangeException();
      cells[c - 'A', colm] = value;
    }
  }
}
```

Eiffel

Lehetőségek

- Többszörös öröklődés
- Ismételt öröklődés
- A metódusok alapértelmezetten virtuálisak
- Van absztrakt osztály és metódus

https://www.eiffel.org/codeboard

Eiffel

Osztályhierarchia:

GENERAL

↑

PLATFORM

↑

ANY

↑

+ NONE osztály

Eiffel – Láthatóságszabályozás

```
class A
  feature {B, C}
     X: INTEGER;
  feature {ANY}
    make(p_name : STRING ) is
      require
           p_name /= ""
      do
           name := p_name ….
  feature {NONE}
     Y: ARRAY[INTEGER];
end
```

B és C látja (és utódaik)

public

private

Eiffel

rename

- átnevezés lehetősége

```
class D inherit
        C
           rename a as ac
            end
        B
  feature
  …
end
```

Eiffel

Példa:

```
class ACCOUNT
creation
    make
feature
    balance: INTEGER
    owner: STRING
    number: STRING

    deposit( sum: INTEGER )
    require
          sum > 0
    do
          balance := balance + sum
    ensure
          balance = old balance + sum
    end -- deposit
```

Eiffel

```
withdraw( sum: INTEGER )
require
      sum > 0
      sum <= balance
do
      balance := balance - sum
ensure
      balance = old balance - sum
end – withdraw
```

Eiffel

```
feature
 make( owner_, number_: STRING )
    require
          valid_owner:   owner_ /= Void
          valid_number: number_ /= Void
    do
          owner  := owner_
          number := number_
    ensure
          owner = owner_ and number = number_
          balance = 0
    end -- make

invariant
    balance >= 0
    owner /= Void
    number /= Void
end --class ACCOUNT
```

Eiffel

```
class SAVINGS_ACCOUNT
inherit
    ACCOUNT
        rename
            make as account_make
        end
creation
    make
feature
    interest: INTEGER
    set_interest( interest_: INTEGER )
      require
          interest_ >= 0
      do
          interest := interest_
      ensure
          interest = interest_
    end -- set_interest
```

Eiffel

```
pay_interest
        do
                deposit(balance*interest//100)
        ensure
                balance = old (balance + balance*interest//100)
        end
```

Eiffel

```
feature
    make( owner_, number_: STRING; interest_: INTEGER )
    require
            valid_owner:   owner_ /= Void
            valid_number: number_ /= Void
            valid_interest: interest_ >= 0
    do
            account_make(owner, number_)
            set_interest(interest_)
    ensure
            owner = owner_ and number = number_
            balance = 0
            interest = interest_
    end -- make
invariant
    interest >= 0
end --class SAVINGS_ACCOUNT
```

Eiffel – többszörös öröklődés

A leszármazott mondja meg, hogy hogyan szeretné az örökölt attribútumokat és metódusokat felhasználni (anomália megoldása)

Ez lehet:

- átnevezés
  - `rename` kulcsszó
- export státusz megváltoztatása a származtatottban
- metódus felüldefiniálása
  - pl. csak az előfeltételt, vagy az implementációt
- műveletek absztrakttá tétele
  - `deferred` - késleltetett
- ismételt öröklődés
  - ismételt öröklődésnél csak egyet örököl, vagy megmondhatja, hogy legyen mindkettő, ha akarja

Eiffel

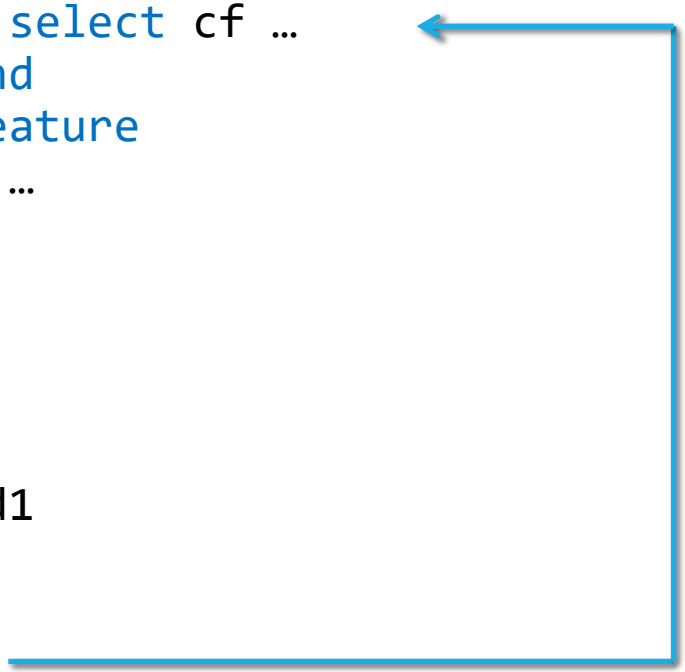
```
class A …
  feature
    f …
end
```

```
class B inherit A
   redefine
        f …
   end
   feature
        f …
   …
end
```

```
class C inherit A
   redefine
        f …
   end
   feature
        f …
   …
end
```

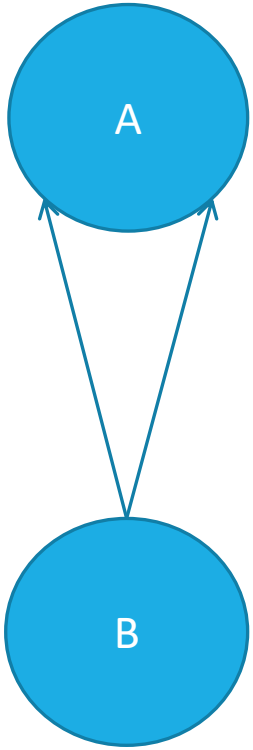
Eiffel (select)

```
class D inherit
      B
        rename f as bf …
      end
      C
        rename f as cf
        select cf …
      end
      feature
        …
end
…
a1:A;
d1:D
…
create d1
d1.bf
a1:=d1
a1.f
```



Eiffel – ismételt öröklődés

```
class A
 feature
   f …
end -- class A


class B
  inherit A
    redefine f
      select f
  end
  inherit  A
    rename f as old_f
  end
  feature
    f …
    do
      old_f …
    end - f
 end -- class B
```



Eiffel – Példa

```
deferred class
    Shape

    feature
        calculateArea: INTEGER
            deferred
            end

        printName
            deferred
            end
end
```

Eiffel – Példa

```
class
    Square
    inherit
        Shape
    create
        make
    feature {NONE}
        sideA: INTEGER
    feature
            make (a: INTEGER)
            do
                sideA := a
            end
            calculateArea: INTEGER
              do
                  Result := sideA * sideA
              end
             printName
              do
                  print("Square ")
                  print(sideA)
                  print("%N")
              end
end
```

Eiffel – Példa

```
class
    APPLICATION
inherit
    ARGUMENTS_32
create
    make
feature {NONE} -- Initialization
    make
        local
            s : Shape
            sq: Square
        do
            create sq.make(5)
            s := sq
            s.printName
            print(s.calculateArea)
        end
end
```

Eiffel

Lehet bevezetni kovariáns metódusokat az altípusban

Példa:

```
class
      TeliSportolo
      feature
         szobatars (s: TeliSportolo)
         do
             print("TeliSportolo szobatars%N")
             s.printname
         end
         printname
         do
             print("TeliSportolo vagyok%N")
         end
end
```

Eiffel

```
class
      Sielo
         inherit TeliSportolo
         redefine
            szobatars, printname
         end
      feature
         szobatars (s: Sielo)
         do
             print("Sielo szobatars%N")
             s.printname
         end
         printname
         do
             print("Sielo vagyok%N")
         end
end
```

Eiffel

Főprogram

```
ts: TeliSportolo
tt: TeliSportolo
si: Sielo

create ts
ts.szobatars(ts)
create si
si.szobatars(si)
-- nem lehetséges, a formális paraméter miatt
-- si.szobatars(ts)

tt := si
-- problémás! (Nem típusbiztos!)
tt.szobatars(ts)
```

Eiffel

Gyakran a kovariancia kódismétlést okoz

- Ezt Anchorokkal lehet kiküszöbölni, az előzőekkel azonos eredményt lehet elérni:

```
class
      TeliSportolo
         feature szobatars (s: like Current)
         do
              print("TeliSportolo szobatars%N")
              s.printname
         end
        …
end
class
      Sielo
         inherit TeliSportolo
         redefine
              printname
         end
      …
end
```

Python

Szkriptnyelv, interpretált.

- Elsősorban a kód olvashatóságára koncentrál, valamint a programozó koncepciójának rövid megfogalmazására
- Az OOP-t támogatja – szükséges mértékben

Tulajdonságok

- Az osztályok is objektumok
- Osztályhoz tartozó nevet bármikor és bárhol be lehet vezetni
- Absztrakt függvény nincs
  - Dekorátorral jelezhető
- Láthatósági szabályozás nincs
  - Névkonvenciók
- Többszörös öröklődés

Python

Tulajdonságok

- Új adattagot bármikor bevezethetünk és törölhetünk
  - `self.adattag = érték`
  - `del self.adattag`
- Property-k
  - Decorator-okkal megadhatók propertyk
  - get és set függvények külön

Python

Tulajdonságok

- Konstruktor
  - ha létezik egy `__init__()` inicializáló metódusa az osztálynak, akkor példányosításkor az objektum létrehozása után meghívódik
- __del__
  - Az objektumhoz rendelt memória terület felszabadításakor lefutó függvény
- Osztályváltozó
  - Osztály definíciójában (és nem függvényében) deklarált változó
- Osztálymetódus
  - `@classmethod` annotációval ellátott metódus, első paramétere az osztály `cls`
- Dinamikus típusosság
  - Polimorfizmusról a megszokott formában nincsen szó
  - Dinamikus kötésről sem, de névfeloldás futásidőben

Python

Öröklődés

- Többszörös öröklődés lehetséges
- Névfeloldásra két módszer

Python2

- Ha egy hivatkozást nem talál az aktuális osztályban, akkor első közvetlen ősben keresi
  - Ha ott sem, akkor az első ős ősében
- Ezután ha még mindig nem találta, akkor a második közvetlen ősben
  - Stb.
- Lényegében mélységi keresés

Python

Öröklődés

- Többszörös öröklődés lehetséges
- Névfeloldásra két módszer

Python3

- Ha egy hivatkozást nem talál az aktuális osztályban, akkor közvetlen ősökben keresi
  - Az ősosztályok definiálási sorrendjében
- Ha nem találta, akkor az ősök közvetlen őseiben folytatja
  - Stb.
- Lényegében szélességi keresés

Python – Példa

```
class Shape:

    def print(self):
        raise NotImplementedError

    def calculateArea(self):
        raise NotImplementedError


class Circle(Shape):
    PI = 3.1415926535

    def __init__(self, radius):
        self._radius = radius

    def print(self):
        print("Circle r: " + self._radius)

    def calculateArea(self):
        return self._radius * self._radius * Circle.PI
```

Python – Példa

```
def polygon_function(self):
    return self._sideA, self._sideB, self._sideA, self._sideB


class Parallelogram(Shape):

    def __init__(self, a, b, deg):
        self._sideA = a
        self._sideB = b
        self._delta = deg
        self._update_height()

    def _update_height(self):
        self._height_a = self._sideA * math.sin(self._delta * math.pi / 180.0)

    @property
    def sideA(self):
        return self._sideA

    @property
    def sideB(self):
        return self._sideB

    @sideB.setter
    def sideB(self, value):
        self._sideB = value
        self._update_height()
```

Python – Példa

```
    @property
    def angle(self):
        return self._delta

    @angle.setter
    def angle(self, value):
        self._delta = value
        self._update_height()

    get_edges = polygon_function

    def is_equilateral(self):
        return self._sideA == self._sideB

    def is_right(self):
        return self._delta == 90

    def print(self):
        print("Parallelogram a, b, delta: " ,
                    self.sideA , " " , self.sideB , " " , self.angle)

    def calculateArea(self):
        return self.sideA * self._height_a
```

Python – Példa

```
class Rhombus (Parallelogram):
    def __init__(self, a, deg):
        Parallelogram.__init__(self, a, a, deg)

    def print(self):
        print("Rhombus a, delta: " , self.sideA , " " , self.angle)

    def calculateArea(self):
        return self.sideA * self.sideA * math.sin(self.angle * math.pi/180.0)

    def is_equilateral(self):
        return True

class Rectangle (Parallelogram):
    def __init__(self, a, b):
        Parallelogram.__init__(self, a, b, 90)

    def print(self):
        print("Rectangle a, b: " , self.sideA , " " , self.sideB)

    def is_right(self):
        return True
```

Python – Példa

```
class Square (Rectangle, Rhombus):

    def __init__(self, a):
        Rectangle.__init__(self, a, a)

    def print(self):
        print("Square a: " , self.sideA)

    def calculateArea(self):
        return self.sideA * self.sideA
```

Python – Példa

```
n1 = Square(30)
n1.print()
print(n1.calculateArea())

n2 = Rhombus(30, 45)
n2.print()
n2.angle = 90
print(n2.is_right())
print(n2.calculateArea())

n2 = n1
print(n2.get_edges())
```

Python

```
class Grand:
    def call(self):
        print("Grandparent")

class Parent1(Grand):
    pass

class Parent2(Grand):
    def call(self):
        print("Parent 2")

class Child(Parent1, Parent2):
    def test(self):
        self.call()

c = Child()
c.test()
```

Python

```
$ python2 demo.py
>> Grandparent

$ python3 demo.py
>> Parent 2
```

De lehetséges expliciten megadni

```
class Child(Parent1, Parent2):
    def test(self):
        Grand.call(self)
```

Ruby https://repl.it/languages/ruby

Teljesen objektumorientált
- Smalltalk, Python hatások, hasonlóságok
- Minden típus osztály, így minden változó objektum
  - Erősen típusos nyelv!
- Az osztályok is objektumok
- Az `Object` osztály minden más típusnak őse
  - Numerikus típusok hierarchiába szervezve
  - Logikai típus nincs
    - A true és a false a TrueClass / FalseClass egyke (Singleton) osztályok egyetlen példánya
- GC
  - Nincs destruktor
- Egyszeres öröklődés
- Dinamikus öröklődés

Ruby

Jelölési konvenciók

- A nagybetűvel kezdődő változók konstansnak számítanak
  - osztálynév nagybetűvel kezdődik
- A kisbetűvel kezdődő nevű változók lokálisak
- A tagváltozók @-cal kezdődnek.
- A osztályszintű (statikus) változók @@-cal kezdődnek.
- A globális változók $ jellel kezdődnek.

```
class Vector
 def initialize( x, y )
  @x = x
  @y = y
 end
end
```

Ruby

A konstruktor az initialize nevű függvény.

- A Rubyban nincs függvénytúlterhelés
  - Egy osztálynak csak egy konstruktora lehet
  - Ha több konstruktorra van szükségünk, akkor ezeket más nevű függvényekként kell megvalósítanunk, és explicite kell őket meghívnunk az objektumra.
- Az initialize függvény paraméterezése nincs előre megszabva
  - annyi paramétert kaphat, amennyit megszabunk

Egy függvény automatikusan visszaadja a legutoljára kiértékelt kifejezést

```
def add a, b
  a + b
end
```

Ruby

Getter függvények

```
class Vector
 def initialize( x, y )
  @x = x
  @y = y
 end
 def x
  @x
 end
 def y
  @y
 end
end
```

Setter függvény

```
 def x=(x)
    @x = x
 end
```

Példányosítás és lekérdezés

```
v = Vector.new(2, 3)
puts v.x, v.y
```

Ruby

Egyszeres öröklődés

- Minden tagfüggvény virtuális.
- A tagváltozók privát hozzáférhetőségűek.
- A tagfüggvények hozzáférhetőségét befolyásolhatjuk a private, public és protected minősítőkkel.
  - Az alapértelmezés a publikus.
- Származtatásnál csak a metódusok öröklődnek
  - Az adattagok nem, azok mindig objektumokhoz vannak kötve.

Ruby

```ruby
class Person
  def initialize(nev)
    @nev = nev
  end
end

class Employee < Person
  def initialize(nev, fizetes)
    super(nev)
    @fizetes = fizetes
  end
end
```

Ruby

Lehet példányból dinamikusan örököltetni, és módosítani

- Az örököltetéskor létrejön az ős egy alosztálya
- A példányra mutató változó dinamikusan megváltoztatja a típusát erre az osztályra

Ruby

```ruby
class Foo
     def bar
          puts "Foo::bar"
     end
end
f = Foo.new
g = Foo.new

class << f
  def only_f
    puts "f::only_f"
  end
end
f.only_f
g.only_f
```

Alternatíva

```ruby
def f.only_f
     # …
end
```

Ruby

Modulok: a többszörös öröklődés helyett

- Modul neve nagybetűvel kezdődik
- Nem példányosítható, az osztályok modulokat foglalhatnak magukba
- A befogadott modul metódusai az osztály példánymetódusai lesznek
- Mixineknek nevezik
- A modulok tartalmazhatnak metódusokat, állandókat, más modulokat, vagy osztályokat
- Örökölhetnek (include) más moduloktól, de osztályoktól nem

Ruby

```ruby
module A
  def a
    puts "A1"
  end
end
module B
  def b
    puts "B1"
  end
end
class Test
  include A
  include B
end
obj = Test.new
obj.a     # => A1
obj.b     # => B1
```

PHP

PHP 5-től van OOP támogatás

- Egyszeres öröklődés
- Interfészek
  - Többet is implementálhat
  - Többszörös öröklődés lehetséges
- Láthatóság
  - public, protected, private
  - Alapértelmezett a public
- Statikus esetben nincs $this, helyette self és parent
- Absztrakt osztályok szokásos módon
- Trait-ek

PHP

```php
abstract class Shape
{
    abstract public function print();
    abstract public function calculateArea();
}

interface Polygon
{
    public function getEdges();
}
```

PHP

```php
class Circle extends Shape
{
    public static $PI = 3.1415926535;

    private $radius;

    public function __construct($r) {
        $this->radius = $r;
    }

    public function print() {
        print("Circle r: " . $this->radius . "\n" );
    }

    public function calculateArea() {
        return $this->radius * $this->radius * Circle::$PI;
    }
}
```

PHP

```php
class Parallelogram extends Shape implements Polygon
{
    protected $sideA;
    protected $sideB;
    protected $delta;
    protected $heightA;

    public function getSideA() { return $this->sideA; }
    public function getSideB() { return $this->sideB; }
    public function getAngle() { return $this->delta; }

    public function isEquilateral()
         { return $this->sideA===$this->sideB; }
    public function isRight()
         { return $this->delta == 90; }
```

PHP

```
    public function __construct($a, $b, $angle) {
        $this->sideA = $a;
        $this->sideB = $b;
        $this->delta = $angle;
        $this->heightA = $b *
                         sin($this->delta * pi() / 180.0);
    }

    public function print() {
        print("Parallelogram a, b, deg: " .
                  $this->sideA . " " . $this->sideB . " " .
                  $this->delta . "\n" );
    }

    public function calculateArea() {
        return $this->sideA * $this->heightA;
    }

    public function getEdges() {
        return [$this->sideA, $this->sideB,
                $this->sideA, $this->sideB];
    }
}
```

PHP

```php
class Rectangle extends Parallelogram
{
    public function __construct($a, $b) {
        parent::__construct($a, $b, 90);
    }

    public function print() {
        print("Rectangle a, b: " . $this->sideA . " " .
                $this->sideB . "\n" );
    }

    public function calculateArea() {
        return $this->sideA * $this->sideB;
    }
}

Shape: $r = new Rectangle(10, 20);
$r->print();
print($r->calculateArea() . "\n");
```

PHP

Trait

- Kód újrafelhasználás a cél
- Metódusok egybefoglalása, akár törzzsel is…
- Nem lehet önállóan példányosítani.

```
trait TrA{
    function a1() { print("TrA->a1()\n"); }
    function a2() { print("TrA->a2()\n"); }
}

class ClB {
    use TrA;
}

ClB: $v = new ClB();
$v->a1(); // TrA->a1()
```

PHP

Trait precedencia

- Az osztályban definiált metódus felülírja a trait-ből kapottat
  - A trait nevével lehet hivatkozni a trait-belire
- A trait felülírja az őstől örökölt metódust
  - A „parent”-tel a lehet hivatkozni az ősre
  - (Abban az esetben, ha egy osztályon belül van egy olyan származtatott tagunk, aminek a neve megegyezik egy, az osztályon belül használt traiten belüli tag nevével, akkor a traiten belüli tag felülírja az osztály származtatott tagját.)
- Az as kulcsszó
  - A metódus láthatóságát megváltoztatja
  - Vagy alias nevet lehet létrehozni

PHP

```php
trait TrA{
    function a1() { print("TrA->a1()\n"); }
    function a2() { print("TrA->a2()\n"); }
}

class ClB {
    use TrA;

    public function a2() {
        print("ClB->a2()\n");
        TrA::a2();
    }
}

ClB: $v = new ClB();
$v->a2(); // ClB->a2() TrA->a2()
```

PHP

```php
trait TrA{
    function a1() { print("TrA->a1()\n"); }
    function a2() { print("TrA->a2()\n"); }
}

class ClB {
    use TrA { a2 as hidden; }

    public function a2() {
        print("ClB->a2()\n");
    }
}

ClB: $v = new ClB();
$v->a2();      // ClB->a2()
$v->hidden(); // TrA->a2()
```

PHP

```php
trait TrA{
    function a1() { print("TrA->a1()\n"); }
    function a2() { print("TrA->a2()\n"); }
}

class Base {
    public function a2() {
        print("Base->a2()\n");
    }
}

class ClB extends Base {
    use TrA;
}

ClB: $v = new ClB();
$v->a2();      // TrA->a2()
```

PHP

Traitek esetén névütközés előfordulhat

```
trait TrA{
    function a1() { print("TrA->a1()\n"); }
    function a2() { print("TrA->a2()\n"); }
}

trait TrB{
    function a1() { print("TrB->a1()\n"); }
    function a2() { print("TrB->a2()\n"); }
    function a3() { print("TrB->a3()\n"); }
}
```

PHP

```php
class ClC {
    use TrA, TrB {
        TrA::a1 insteadof TrB;
        TrB::a2 insteadof TrA;
        TrB::a1 as private hidden;
    }
}
```

PHP

Traitekben lehet még

- Absztrakt metódusok
- Statikus adattagok és statikus metódus
  - Statikus adattag csak metóduson belül, statikus metódus osztálymetódusként viselkedik

```
trait TrA{
    function a1() { print("TrA->a1()\n"); }
    function a2() { print("TrA->a2()\n"); }
    abstract function b1();
    static function s1() { print("TrA->static\n"); }
}
```

PHP

Használatuk

```
class ClStat {
    use TrA;
    function b1() {
        print("ClStat->b1() implemented\n");
    }
}

ClStat: $v = new ClStat();
$v->b1();
ClStat::s1();
```

Kotlin

Java OOP képességein alapul

- Kiegészítésekkel
- Következetesebb, rövidebb szintaxissal

Tulajdonságok

- Az osztályok alapértelmezés szerint nem származtathatók tovább
  - Open
- Elsődleges és másodlagos konstruktorok
  - Elsődleges célja a rövidebb kód, a mezők közvetlen felsorolásával és inicializálsával
  - Másodlagos, minden további esetre
  - Elsődlegesből legfeljebb egy lehet, másodlagosból több is
  - Konstruktor delegáció

Kotlin

Tulajdonságok

- Közös ős: Any
- Felüldefiniálást jelezni kell
  - `override`
- Felüldefiniálás megakadályozható
  - `final`
- Tulajdonságok és getter, setter függvények definiálhatók
  - Mögöttes változót automatikusan – szükség esetén – hozza létre
- Egyszeres öröklődés
  - Interfészek között többszörös öröklődés
  - Interfészben lehet tulajdonságot megadni
  - Tulajdonságok is felüldefiniálhatók
- Absztrakt osztályok

Kotlin

Tulajdonságok

- Adatosztályok
  - `data class`
- Singleton
  - `object` a `class` helyett
- Static adatok
  - `companion object`

Kotlin

```
abstract class Shape {
    abstract fun calculateArea(): Double
    abstract fun print()
}
interface Polygon {
    fun getEdges(): DoubleArray?
}
```

Kotlin

```
class Circle(var radius: Double): Shape() {

    override fun print() {
        println("Circle, radius: $radius")
    }
    override fun calculateArea(): Double =
                    radius * radius * Circle.PI

    companion object {
        const val PI: Double = 3.1415926535
    }
}
```

Kotlin

```
open class Parallelogram constructor(
    protected var _sideA: Double,
    protected var _sideB: Double,
    private var _delta: Double) : Shape(), Polygon {
    private var _heightA: Double = updateHeight()

    var sideA: Double
        get() = _sideA
        set(value) { _sideA = value }

    var sideB: Double
        get() = _sideB
        set(value) { _sideB = value;
                     _heightA = updateHeight()}

    var angle: Double
        get() = _delta
        set(value) { _delta = value;
                     _heightA = updateHeight()}
```

Kotlin

```
    private fun updateHeight(): Double =
        _sideB * kotlin.math.sin(this._delta *
                                 kotlin.math.PI / 180.0)

    open fun isEquilateral(): Boolean = (sideA == _sideB)

    open fun isRight(): Boolean = (_delta == 90.0)

    override fun print() = println("Parallelogram a, b,
                        delta: $_sideA, $_sideB, $_delta")

    override fun calculateArea(): Double =
                                        _sideA * _heightA

    override fun getEdges(): DoubleArray? {
        return doubleArrayOf(_sideA, _sideB,
                             _sideA, _sideB)
    }
}
```

Kotlin

```
class Rectangle(
        sideA: Double,
        sideB: Double):
    Parallelogram(sideA, sideB, 90.0) {

    override fun print() {
        println("Rectangle a, b:
                    $sideA, $sideB")
    }

    override fun isRight(): Boolean {
        return true
    }
}
```

Kotlin

```
fun main(args: Array<String>) {
    var s: Shape = Rectangle(10.0, 20.0)
    s.print()

    var p: Parallelogram =
                Parallelogram(10.0,20.0,45.0)
    p.angle = 90.0
    p.print()
}
```

Kotlin

Kiterjesztés adható egy osztályhoz öröklés nélkül

```
fun MutableList<Int>.swap(index1: Int,
                          index2: Int) {
    val tmp =       this[index1] // the list
    this[index1] = this[index2]
    this[index2] = tmp
}

val l = mutableListOf(1, 2, 3)
l.swap(0, 2)
```

Kotlin

Hivatkozás ősre (interfész esetén is)

```
interface A {
    fun foo() { print("A") }
    fun bar()
}
interface B {
    fun foo() { print("B") }
    fun bar() { print("bar") }
}
class C : A {
    override fun bar() { print("bar") }
}
```

Kotlin

Hivatkozás ősre (interfész esetén is)

```
class D : A, B {
    override fun foo() {
        super<A>.foo()
        super<B>.foo()
    }
    override fun bar() {
        super<B>.bar()
    }
}
```

Funkcionális és logikai programozás

KÖVETKEZŐ ALKALOMMAL