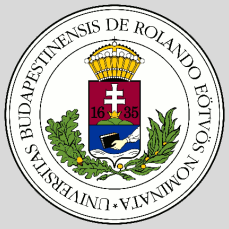
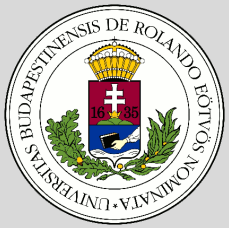


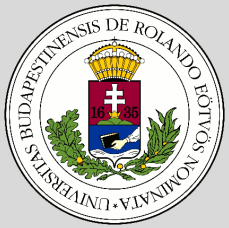
Programozási technológia I.

Statikus modell - Osztálydiagram

Dr. Szendrei Rudolf
Informatikai Kar
Eötvös Loránd Tudományegyetem



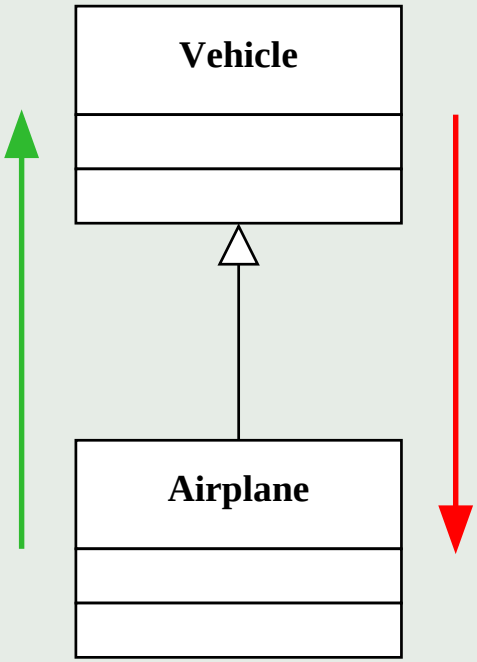
Tartalom

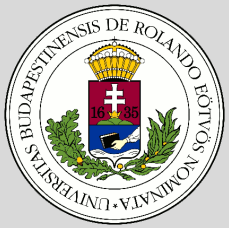


Általánosítás és specializáció

Általánosítás és specializáció

- Az általánosítás a specializáció ellentettje
- Két osztály között állhat fenn
  - Általános osztály, ősosztály, superclass
  - Általános tulajdonságokkal rendelkezik
  - Absztrakt metódusai is lehetnek
  - Speciális osztály, származtatott osztály, alosztály, subclass
  - Speciálisabb tulajdonságokkal rendelkezik
  - Átveszi az általános osztály tulajdonságait, azokat kiegészítheti, átfogalmazhatja
- „is a kind of" reláció: a repülő az egyfajta jármű

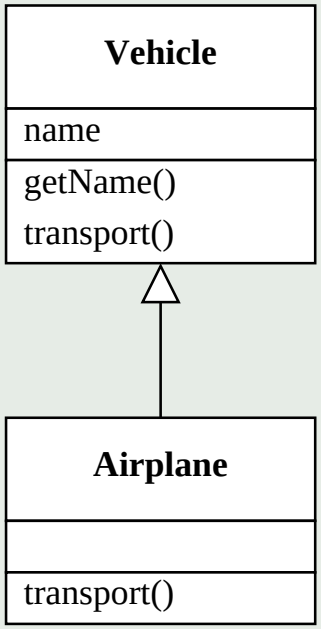


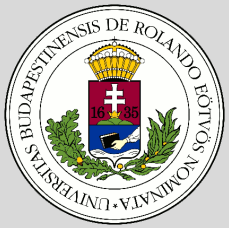


Általánosítás és specializáció

Általánosítás és specializáció - folyt.

- Az általános osztály a szokásos módon rendelkezni fog a felsorolt attribútumokkal és műveletekkel.
- A speciális osztály megörökli az általános osztály minden attribútumát és operációját, ezeket nem kell újra feltüntetni a diagramon. Csak azoknak a tulajdonságoknak kell szerepelni a diagramon, amelyek csak a speciális osztályra vonatkoznak, és a megörököltek közül azok, amelyek valamilyen változáson mennek keresztül a speciális osztályban (például a speciális osztály felüldefiniál egy műveletet).





Általánosítás és specializáció

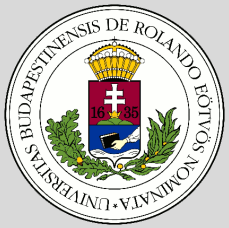
Általánosítás és specializáció - folyt.

- A specializáció megvalósítása származtatással történik

```
class Vehicle {...}
class Airplane extends Vehicle {...}
```

- A származtatás nem szimmetrikus, nem lehet reflexív
- Új osztályok létrehozásának egy módja, mellyel absztrakt és konkrét osztályok egyaránt létrejöhetnek
- A specializáció lehet többszörös, ekkor egy általánosításból több származott osztály jön létre
- Az általánosítás is lehet többszörös, amikor a származtatás több általánosítással történik

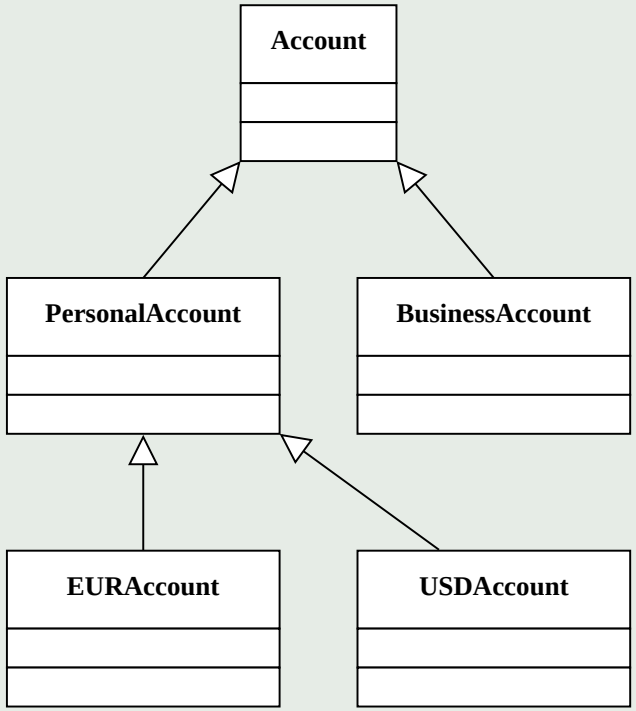
  Java-ban a többszörös általánosítás csak úgy lehetséges, ha a több általános osztály közül legfeljebb az egyik konkrét osztály, a többi pedig legfeljebb interfész

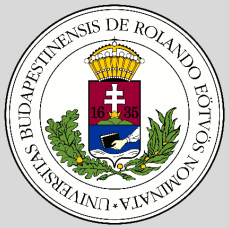


Általánosítás és specializáció

Többszörös specializáció

- Egy általános osztályból több speciális osztály is származtatható
- További specializációkkal az osztályok hierarchikus szerkezete hozható létre
- Az általános osztály megalkotása jó absztrakciós képességeket igényel. Nem egyszerű megragadni a specializáció során jól használható általános, absztrakt tulajdonságokat

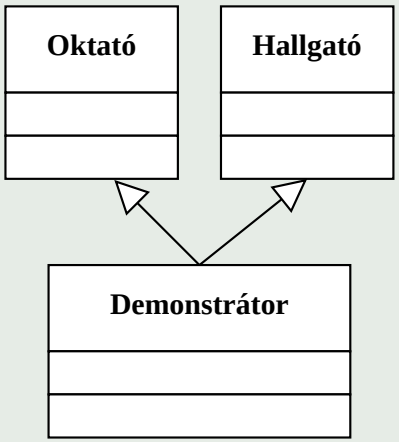




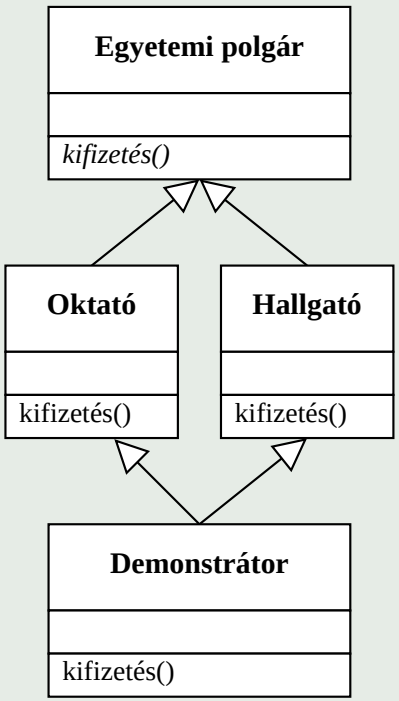
Általánosítás és specializáció

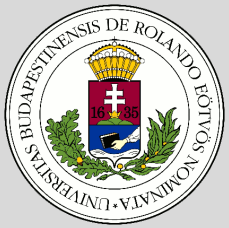
Többszörös általánosítás

- Egy speciális osztály több általános osztály tulajdonságaival is rendelkezhet



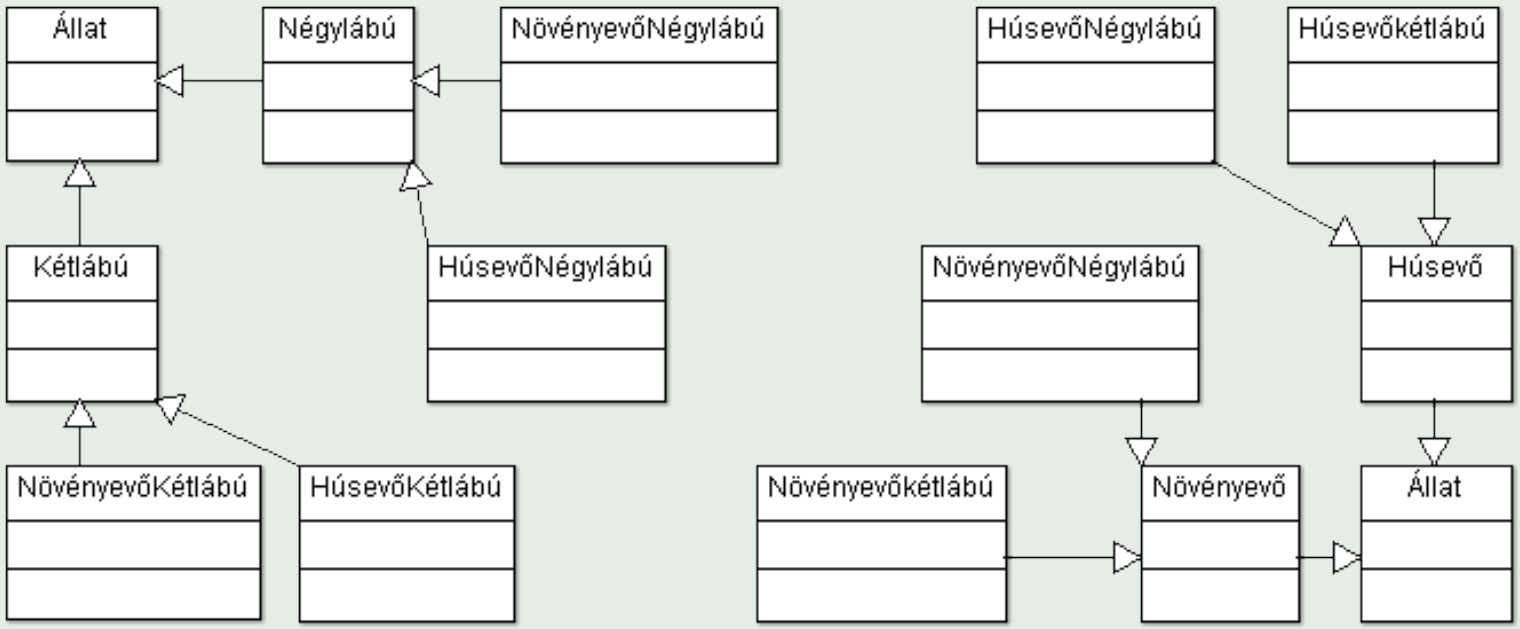
- Csak megfelelő korlátozó feltételekkel lehetséges az átvett, megörökölt információk összekeveredés nélküli kezelése



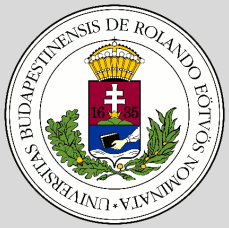


Általánosítás és specializáció

Melyik modell a helyes?



Mindkettő jó lehet! Az öröklődést inkább a működésbeli eltérés esetén szoktuk alkalmazni...



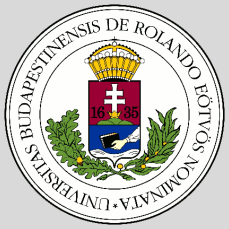
Öröklődés

Az öröklődés természetbeni érdekes esetei

- Milyen öröklődési kapcsolat áll fenn Dolly bárány és klónja között?
- Mi a helyzet a baromfi udvarban, ahol a csirke anyja egyben testvére apai ágon?
- Ha valakit túlél a klónja, akkor felbontsák a végrendeletét?

Öröklődés a programozásban

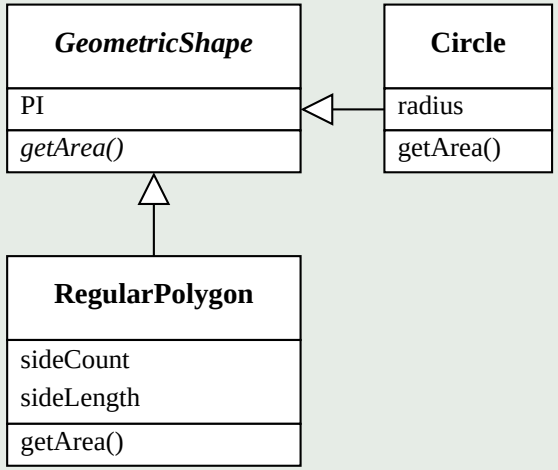
- A programozási nyelvekben szerencsénkre az öröklődés csupán funkcionális, tulajdonságbeli specializációt testesítenek meg és csak osztály szinten vizsgáljuk ezt.
- A természetbeni öröklődéseknél felvetett esetek nem az osztályok, hanem az objektumok közötti viszonyokat vizsgálja.



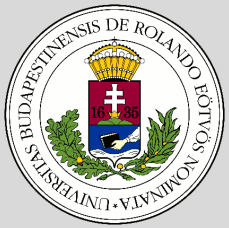
Öröklődés

Öröklődés JAVA-ban

```
public abstract class GeometricShape{
    // védett adattag
    protected final double PI = Math.PI;
    // absztrakt függvény
    public abstract double getArea();
}

public class Circle extends GeometricShape{
    protected double radius;

    @Override
    public double getArea(){
        return radius * radius * PI; }
}
```




```
public class RegularPolygon extends GeometricShape{
    protected int    sideCount;
    protected double sideLength;

    @Override
    public double getArea(){
        return 1 / 4.0 * sideCount
               * sideLength * sideLength
               * (1 / Math.tan(PI / sideCount));
    }
}
```

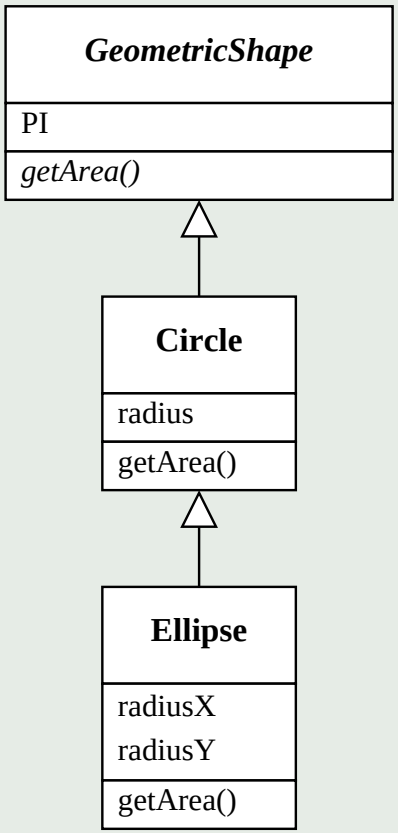


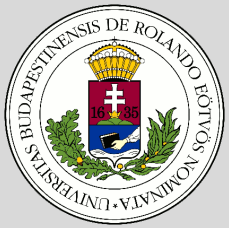
Öröklődés

Felüldefiniálás

Bármely megörökölt függvényt felül lehet definiálni (kivéve...), de csak akkor számít felüldefiniálásnak, ha a függvény szignatúrája pontosan ugyanaz, mint az ősosztályban.

```
public abstract class GeometricShape{
    protected final double PI = Math.PI;
    public abstract double getArea();
}

public class Circle extends GeometricShape{
    protected double radius;

    @Override
    public double getArea(){
        return radius * radius * PI; }
}

public class Ellipse extends Circle{
    private double radiusX = radius;
    private double radiusY;

    @Override
    public double getArea(){
        return radiusX * radiusY * PI; }
}
```





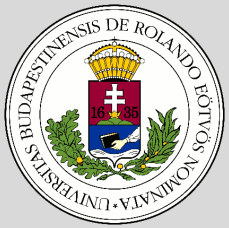
Absztrakt függvények

- Bármely függvénynek hiányozhat a megvalósítása, azaz a függvénytörzse (kivéve...)
- Ekkor a függvény absztrakt függvény lesz.

Absztrakt osztályok

- Ha egy osztály tartalmaz absztrakt függvényt, akkor maga az osztálynak is absztraktnak kell lennie, hogy jelezzük az osztály megvalósítása hiányos.
- Ha egy származtatott osztálynak absztrakt őse van, akkor vagy meg kell valósítani minden megörökölt absztrakt függvényt, vagy pedig a származtatott osztálynak is absztraktnak kell lennie.
- Absztrakt osztályból nem lehet objektumokat példányosítani.

```
                    eltejavalibtrial.ElteJavaLibTrial.GeometricShape is abstract; cannot be instantiated
                    ----
                    (Alt-Enter shows hints)
GeometricShape shape = new GeometricShape();
```

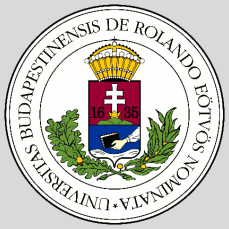


Öröklődés

Interfészek

- Az interfészek teljesen absztrakt osztályok, sőt akár üresek is lehetnek (ha csak egy típus megjelölést akarunk alkalmazni)
- Nem rendelkezhetnek megvalósítással, legfeljebb konstans attribútumokkal

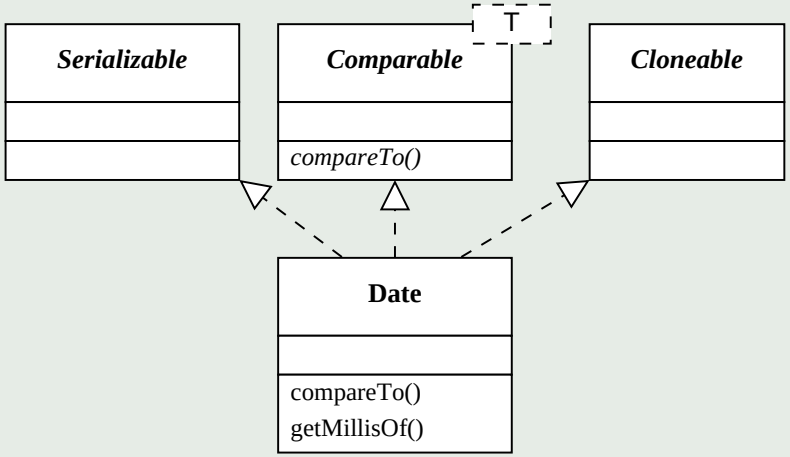
  egyik függvényének sem lehet függvénytörzse
- Ha teljesen absztrakt osztályból, interfészből származtatunk, akkor inkább megvalósításról (implementációról, realizációról) beszélünk
- A Java interfészek bizonyos értelemben hasonlóak a C++ programok header fájljaihoz
- Az osztályokhoz hasonlóan az interfészek is származtathatóak egymásból
- Java-ban csak interfészekből lehet több ősosztály (és legfeljebb egy konkrét osztályból)
- Az interfészeket konkrét osztályokkal valósítjuk meg



Öröklődés

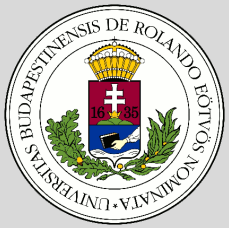
Példa interfészek használatára

- Az, hogy a Comparable interfész egyben egy generikus típus is, az az öröklődéstől teljesen független




```
public interface Comparable<T> {
    /**...*/
    public int compareTo(T o);
}
```

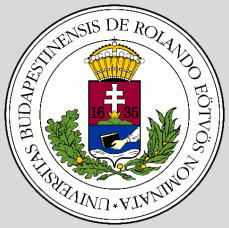
```
public class Date
    implements java.io.Serializable, Cloneable, Comparable<Date>
{
    public int compareTo(Date anotherDate)
    {
        long thisTime = getMillisOf(this);
        long anotherTime = getMillisOf(anotherDate);
        return (thisTime<anotherTime ? -1 :
                (thisTime==anotherTime ? 0 : 1));
    }
    ...
}
```



Interfészek

Modularitás, komponens elvűség

- Az interfészek segítségével a szoftverek modulokra vághatók szét.
- Az interfész a program részek összekapcsolódásának „szerződését” rögzíti.
- A nagy bonyolultságú rendszerek komponenseit sokszor teljesen független fejlesztői csoportok készítik, vagy már készen vásárolják a szükséges komponenseket (off-the-shelf).
- Java-ban az interfészeket használjuk az eseménykezelők megvalósítására is.

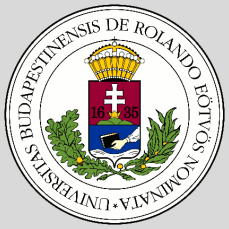


Öröklődés

Egyéb módszerek

- Java-ban az ősosztályra a `super` kulcsszóval hivatkozunk
- A függvények megjelölhetőek a `final` kulcsszóval, ha azt akarjuk, hogy azokat ne lehessen felüldefiniálni (kivétel ez alól az absztrakt függvény)

```
public class RegularPolygon extends GeometricShape {

    protected int sideCount;
    protected double sideLenght;
    private final double PI = 3.14;

    @Override
    public final double getArea() {
        return 1 / 4.0 * sideCount
                * sideLenght * sideLenght
                * (1 / Math.tan(super.PI / sideCount));
    }
}

public class Square extends RegularPolygon {

    public double getArea() {
        return sideLenght * sideLenght;
    }
}
```

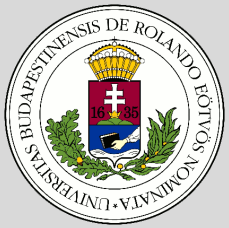


Öröklődés

Egyéb módszerek - folyt.

- Egy teljes osztály is megjelölhető a `final` kulcsszóval (ha nem absztrakt osztály), ezzel megtiltjuk, hogy származtatni lehessen belőle

```
public final class Circle extends GeometricShape {...}

public class Ellipse extends Circle {...}
```

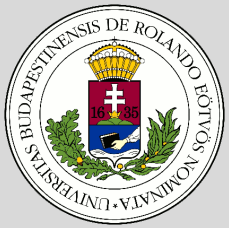


Öröklődés

Egyéb módszerek - folyt.

- Az absztrakt osztályok objektum példányosításkor helyben specializálhatóak (ekkor egy új, névtelen osztály fog létrejönni)

```
GeometricShape shape = new GeometricShape() {
    @Override
    public double getArea() {
        return 0.0;
    }
};
```



Polimorfizmus

Öröklődés és polimorfizmus kapcsolata

- Mivel az öröklődés egy „is a kind of..." reláció, ezért a speciálisabb objektumok behelyettesíthetőek az általánosabb objektumok helyére, az általános osztály tulajdonságainál fogva kezelhetőek
- Többalakúság (polimorfizmus): egy Square objektum felveheti a Square osztály tulajdonságait, de az általános GeometricShape osztály tulajdonságait is (mivel megörökli azokat)

```
GeometricShape shape = new Square();
double areaOfSquare = shape.getArea();
System.out.println(areaOfSquare);

List<GeometricShape> shapes = new ArrayList<>();
shapes.add(new Circle());
shapes.add(new Ellipse());
shapes.add(shape);

for (GeometricShape geomShape : shapes) {
    System.out.println(geomShape.getArea());
}
```