

# Programozási nyelvek – Java

Hatodik előadás



**Kozsik Tamás**

ELTE Eötvös Loránd Tudományegyetem

- Absztrakció
  - Egységbe zárás
  - Információ elrejtés
- Öröklődés
- Altípusos polimorfizmus
- Dinamikus kötés



- 1 Öröklődés
- 2 Altípusos polimorfizmus
- 3 Felüldefiniálás, dinamikus kötés

# Öröklődés (inheritance)

```
class A extends B { ... }
```

- Egy típust egy másik típusból származtatunk
  - Csak a különbségeket kell megadni:  $A \triangle B$
  - Újrafelhasználás
- Itt: az A a gyermekosztálya a B szülőosztálynak
- Transzitivitás: leszármazott osztály – ősoosztály
  - alosztály: subclass, derived class
  - bázisosztály: super class, base class
- Körkörösség kizárva!



# Öröklődéssel definiált osztály

- A szülőosztály tagjai átöröklődnek
- Újabb tagokkal bővíthető (Java: extends)
- Megörökölt példánymetódusok újradefiniálhatók
  - ... és újradeklarálhatók

```
public class Date {  
    public final int year, month, day;  
    public Date( int year, int month, int day ){ ... }  
}
```

```
public class Time extends Date {  
    public final int hour, minute;  
    public Time( int y, int m, int d, int hour, int minute ){ ... }  
}
```

# A konstruktor(ok) megírandó(k)!

```
class Date {  
    private int year, month, day;  
    Date( int year, int month, int day ){  
        this.year = year; ...  
    }  
    ...  
}
```

```
class Time extends Date {  
    private int hour, minute;  
    Time( int y, int m, int d, int hour, int minute ){  
        super(y,n,d);           // szülőosztály konstruktorát  
        this.hour = hour; ...  
    }  
    ...  
}
```



# Öröklődéssel definiált interface

## Adatszerkezetek bejárásához

```
package java.util;  
public interface Iterator<E> {  
    boolean hasNext();  
    E next();  
}
```

## Új műveletekkel való kibővítés

```
package java.util;  
public interface ListIterator<E> extends Iterator<E> {  
    boolean hasPrevious();  
    E previous();  
    ...  
}
```

# Típusok közötti származtatás

- Interface extends interface
- Osztály implements interface
- Osztály extends osztály





# Többszörös öröklődés

(Multiple inheritance)

- Egy típust több más típusból származtatunk
- Javában: több interface-ből
- Problémákat vet fel



# Példák

OK

```
package java.util;  
public class Scanner implements Closeable, Iterator<String> { ... }
```

OK

```
interface PoliceCar extends Car, Emergency { ... }
```

Hibás

```
class PoliceCar extends Car, Emergency { ... }
```



# Különbség class és interface között

- Osztályt lehet példányosítani
  - `abstract class`?
- Osztályból csak egyszeresen öröközhetünk
  - `final class`?
- Osztályban lehetnek példánymezők
  - interface-ben: `public static final`
- Osztályban nem csak `public` lehet



# abstract class

- Részlegesen implementált osztály
  - Tartalmazhat abstract metódust
- Nem példányosítható
- Származtatással konkretizálhatjuk

```
package java.util;  
public abstract class AbstractList<E> implements List<E> {  
    ...  
    public abstract E get( int index ); // csak deklarálva  
    public Iterator<E> iterator(){ ... } // implementálva  
    ...  
}
```



# Konkretizálás

```
public abstract class AbstractCollection<E> implements Collection<E> {
    ...
    public abstract int size();
}
```

```
public abstract class AbstractList<E> extends AbstractCollection<E>
    implements List<E> {
    ...
    public abstract E get( int index );
}
```

```
public class ArrayList<E> extends AbstractList<E> {
    ...
    public int size(){ ... }           // implementálva
    public E get( int index ){ ... }   // implementálva
}
```

# Öröklődésre tervezés

- Könnyű legyen származtatni belőle
- Ne lehessen elrontani a típusinvariánst



# protected láthatóság

```
package java.util;

public abstract class AbstractList<E> implements List<E> {
    ...
    protected int modCount;
    protected AbstractList(){ ... }
    protected void removeRange(int fromIndex, int toIndex){ ... }
    ...
}
```

- Ugyanabban a csomagban
- Más csomagban csak a leszármazottak

$\text{private} \subseteq \text{félnyilvános (package-private)} \subseteq \text{protected} \subseteq \text{public}$



# A private tagok nem hivatkozhatók a leszármazottban!

```
class Counter {  
    private int counter = 0;  
    public int count(){ return ++counter; }  
}
```

```
class SophisticatedCounter extends Counter {  
    public int count( int increment ){  
        return counter += increment;    // fordítási hiba  
    }  
}
```





# Javítva

```
class Counter {  
    private int counter = 0;  
    public int count(){ return ++counter; }  
}
```

```
class SophisticatedCounter extends Counter {  
    public int count( int increment ){  
        if( increment < 1 ) throw new IllegalArgumentException();  
        while( increment > 1 ){  
            count();  
            --increment;  
        }  
        return count();  
    }  
}
```



## protected

```
package my.basic.types;  
public class Counter {  
    protected int counter = 0;  
    public int count(){ return ++counter; }  
}
```

```
package my.advanced.types;  
class SophisticatedCounter extends my.basic.types.Counter {  
    public int count( int increment ){  
        return counter += increment;  
    }  
}
```



# final class

```
package java.lang;  
public final class String implements ... { ... }
```

- Nem lehet belőle leszármaztatni
- Módosíthatatlan (immutable) esetben nagyon hasznos



# Mi lehet egy interface-ben?

- Absztrakt (példány)metódus
- [Statikus metódus]
- [Példánymetódus default implementációval]
- Konstansdefiníció



# Réges-régen, egy messzi-messzi galaxisban...

```
interface Color {  
    int BLACK = 0;    // public static final  
    int WHITE = 1;  
    int GREEN = 2;  
    ...  
}
```



# Felsorolási típus

```
enum Color { BLACK, WHITE, GREEN }
```

- Nem példányosítható
- Nem származtatható le belőle
- Használható switch-utasításban
- ...



# Az öröklődés két aspektusa

- Kódöröklés
- Altípusképzés



1 Öröklődés

2 Altípusos polimorfizmus

3 Felüldefiniálás, dinamikus kötés



# Altípus fogalma

$A <: B$

## LSP: Liskov's Substitution Principle

Egy  $A$  típus altípusa a  $B$  (bázis-)típusnak, ha az  $A$  egyedeit használhatjuk a  $B$  egyedei helyett, anélkül, hogy ebből baj lenne.



# Öröklődés $\Rightarrow$ altípusosság

```
class A implements I
```

$$A \Delta_{ci} I \Rightarrow A <: I$$

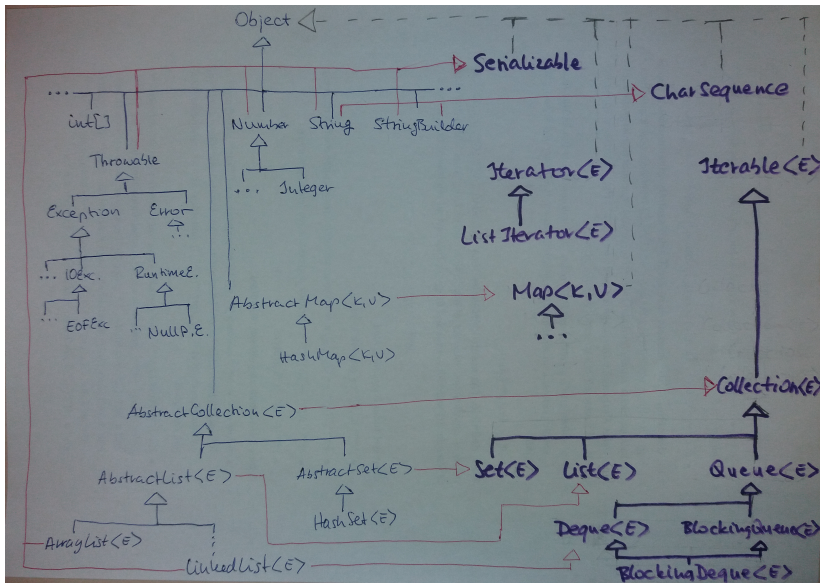
```
class A extends B
```

$$A \Delta_c B \Rightarrow A <: B$$

```
interface I extends J
```

$$I \Delta_i J \Rightarrow I <: J$$


# Típusok hierarchiája a Javában



boolean  
char  
byte  
short  
int  
long  
float  
double



# Típusok hierarchiája a Javában

körmentes irányított gráf (DAG: directed acyclic graph)



# java.lang.Object

Minden osztály belőle származik, kivéve önmagát!

```
package java.lang;

public class Object {
    public Object(){ ... }
    public String toString(){ ... }
    public int hashCode(){ ... }
    public boolean equals( Object that ){ ... }
    ...
}
```



# Minden a `java.lang.Object`-ből származik

... kivéve a primitív típusokat!

```
class A {}
```

```
class A extends java.lang.Object {  
    A(){                // generated default constructor  
        super();        // calls no-arg constructor of parent  
    }  
}
```



# Konstruktorok egy osztályban

- Egy vagy több explicit konstruktor
- Alapértelmezett konstruktor



# Konstruktor törzse

## Első utasítás

- Explicit this-hívás
- Explicit super-hívás
- Implicit (generálódó) `super()`-hívás (no-arg!)

## Többi utasítás

Nem lehet `this-` vagy `super-`hívás!





# Érdekes hiba

## Ártatlannak tűnik

```
class Base {
    Base( int n ){}
}

class Sub extends Base {}
```

## Jelentése

```
class Base extends Object {
    Base( int n ){
        super();
    }
}

class Sub extends Base {
    Sub(){ super(); }
}
```



# Altípus reláció

$$<: = (\Delta_c \cup \Delta_i \cup \Delta_{ci} \cup \Delta_o)^*$$

- $\Delta_o$  jelentése: minden a `java.lang.Object`-ből származik
- $\varrho^*$  jelentése:  $\varrho$  reláció reflexív, tranzitív lezártja
  - Ha  $A \varrho B$ , akkor  $A \varrho^* B$
  - Reflexív lezárt:  $A \varrho^* A$
  - Tranzitív lezárt: ha  $A \varrho^* B$  és  $B \varrho^* C$ , akkor  $A \varrho^* C$

Ez egy parciális rendezés (RAT)!



# A dinamikus típus a statikus típus altípusa

Ha  $A \leq B$ , akkor

- $B \ v = \text{new } A()$ ; helyes
- $\text{void } m( B \ p ) \dots$  esetén  $m(\text{new } A())$  helyes



# Specializálás

- Az altípus „mindent tud”, amit a bázistípus
- Az altípus speciálisabb lehet
- Ez az *is-egy* reláció
  - Car *is-a* Vehicle
  - Boat *is-a* Vehicle
- Emberi gondolkodás, OO modellezés



# Többszörös altípusképzés

- Egy fogalom több általános fogalom alá tartozhat
  - PoliceCar *is-a* Car **és** *is-a* EmergencyVehicle
  - FireBoat *is-a* Boat **és** *is-a* EmergencyVehicle
- Összetett fogalmi modellezés Javában: interface



# Altípusos polimorfizmus

Ha egy kódbázist megírtunk, újrahasznosíthatjuk speciális típusokra!

- Általánosabb típusok helyett használhatunk altípusokat
- Több típusra is működik a kódbázis: polimorfizmus

**Újrafelhasználhatóság!**



# Kivételosztályok hierarchiája

`java.lang.Throwable`

- `java.lang.Exception`
  - `java.sql.SQLException`
  - `java.io.IOException`
    - `java.io.FileNotFoundException`
  - ...
  - saját kivételek általában ide kerülnek
  - `java.lang.RuntimeException`
    - `java.lang.NullPointerException`
    - `java.lang.ArrayIndexOutOfBoundsException`
    - `java.lang.IllegalArgumentException`
    - ...
- `java.lang.Error`
  - `java.lang.VirtualMachineError`
  - ...



# Nem ellenőrzött kivételek

- `java.lang.RuntimeException` és leszármazottjai
- `java.lang.Error` és leszármazottjai

Egyes alkalmazási területen akár ezek is kezelendők!





# Kivételkezelő ágak

```
try {  
    ...  
} catch( FileNotFoundException e ){  
    ...  
} catch( EOFException e ){  
    ...  
} // nem kezeltük a java.net.SocketException-t
```



# Speciálisabb-általánosabb kivételkezelő ágak

```
try {  
    ...  
} catch( FileNotFoundException e ){  
    ...  
} catch( EOFException e ){  
    ...  
} catch( IOException e ){ // minden egyéb IOException  
    ...  
}
```



# Fordítási hiba: elérhetetlen kód

```
try {  
    ...  
} catch( FileNotFoundException e ){  
    ...  
} catch( IOException e ){ // minden egyéb IOException  
    ...  
} catch( EOFException e ){ // rossz sorrend!  
    ...  
}
```



- 1 Öröklődés
- 2 Altípusos polimorfizmus
- 3 Felüldefiniálás, dinamikus kötés

# Példánymetódusok felüldefiniálhatók

redefine/override an instance method

```
package java.lang;  
public class Object {  
    public String toString(){ ... }  
    ...  
}
```

```
public class Date {  
    ...  
    @Override public String toString(){  
        return year + "." + month + "." + day + ".";  
    }  
}
```



# És újra felüldefiniálhatók

```
public class Date {  
    ...  
    @Override public String toString(){  
        return year + "." + month + "." + day + ".";  
    }  
}
```

```
public class Time extends Date {  
    ...  
    @Override public String toString(){  
        return year + "." + month + "." + day + ". " +  
            hour + ":" + minute;  
    }  
}
```



# Meghívható a szülőben adott implementáció

```
public class Date {  
    ...  
    @Override public String toString(){  
        return year + "." + month + "." + day + ".";  
    }  
}
```

```
public class Time extends Date {  
    ...  
    @Override public String toString(){  
        return super.toString() + " " +  
            hour + ":" + minute;  
    }  
}
```



# Túlterhelés és felüldefiniálás

```
package java.lang;
public final class Integer extends Number {
    ...
    @Override public String toString(){ ... }
    public static String toString( int i ){ ... }
    public static String toString( int i, int radix ){ ... }
    ...
}
```





# Különbségtétel

## Túlterhelés

- Ugyanazzal a névvel, különböző paraméterezéssel
- Megörökölt és bevezetett műveletek között
- Fordító választ az aktuális paraméterlista szerint

## Felüldefiniálás

- Bázisosztályban adott műveletre
- Ugyanazzal a névvel és paraméterezéssel
  - Ugyanaz a metódus
  - Egy példánymetódusnak lehet több implementációja
- Futás közben választódik ki a „legspeciálisabb” implementáció



# Dinamikus kötés

dynamic/late binding

```
Time t = new Time(2003,11,22,17,25);  
Date d = t;          // altípusosság (up-cast)  
Object o = d;  
  
System.out.println( t.toString() ); // 2003.11.22. 17:25  
System.out.println( d.toString() );  
System.out.println( o.toString() );
```

Példánymetódus hívásához használt kitüntetett paraméter dinamikus típusához legjobban illeszkedő implementáció hajtódik végre.



# A statikus és a dinamikus típus szerepe

## Statikus típus

Mit szabad csinálni a változóval?

- Statikus típusellenőrzés

```
Object o = new Date(1970,1,1);  
o.setYear(2000); // fordítási hiba
```

## Dinamikus típus

- Melyik implementációját egy felüldefiniált műveletnek?

```
Object o = new Date(1970,1,1);  
System.out.println(o); // toString() impl. kiválasztása
```

- Dinamikus típusellenőrzés



# Típuskényszerítés (down-cast)

- A „(Date)o” kifejezés statikus típusa Date
- Ha o dinamikus típusa Date, akkor működik
- Ha nem, ClassCastException lép fel (futási hiba)



# Dinamikus típusellenőrzés

- Futás közben, dinamikus típus alapján
- Pontosabb, mint a statikus típus
  - Altípus lehet
- Rugalmasság
- Biztonság: csak ha explicit kérjük (type cast)

```
Object o = new Time(2003,11,22,17,25);
```

```
...
```

```
if( o instanceof Date ){  
    ((Date)o).setYear(2000);  
}
```



# Statikus és dinamikus típus: összefoglalás

Változók, paraméterek, kifejezések esetén

## Statikus

- Deklarált
- Osztály/interface
- Állandó
- Fordítási időben ismert
- Általánosabb
- Statikus típusellenőrzéshez
- Biztonságot ad

## Dinamikus

- Tényleges
- Osztály
- Változhat futás közben
- Futási időben derül ki
- Speciálisabb
- Dinamikus típusellenőrzéshez
- Rugalmasságot ad



# Dinamikus típus ábrázolása futás közben

- `java.lang.Class` osztály objektumai
- Futás közben lekérhető

```
Object o = new Time(2003,11,22,17,25);  
Class c = o.getClass();    // Time.class  
Class cc = c.getClass();   // Class.class
```



# Dinamikus kötés megörökölt metódusban is!

```
public class Date {  
    ...  
    @Override public String toString(){ ... }  
    public void print(){  
        System.out.println( toString() );  
    }  
}
```

```
public class Time extends Date {  
    ...  
    @Override public String toString(){ ... }  
}
```

```
Object o = new Time(2003,11,22,17,25);  
((Date)o).print();
```



# Dinamikus kötés: csak példánymetódusra

- *Felüldefiniálni* csak példánymetódust lehet
  - ha nem final
- *Megvalósítani* abstract-ot, pl. interface-ből

Kell a kitüntetett paraméter (dinamikus típusa)



# Mező és osztálysztintű metódus nem definiálható felül

Elfedési szabályok...



# Példa: equals-metódus

- *Tartalmi* egyenlőségvizsgálat referenciatípusokra

```
Date d1 = new Date(1970,1,1);  
Date d2 = new Date(1970,1,1);  
System.out.println( d1 == d2 );  
System.out.println( d1.equals(d2) );
```

- Egy equals metódus sok (rész)implementációval
  - Együttesen adnak egy összetett implementációt
- Szerződése betartandó
  - Determinisztikus
  - Ekvivalencia-reláció (RST)
  - Igaz ez: `a == null || a.equals(null) == false`
  - Konzisztens a hashCode metódussal



# Szabályos felüldefiniálás

```
package java.lang;

public class Object {
    ...
    public boolean equals( Object that ){ return this == that; }
    public int hashCode(){ ... }
}
```

```
public class Date {
    ...
    @Override public boolean equals( Object that ){
        if( that != null && getClass().equals(that.getClass()) ){
            Date d = (Date)that;
            return year == d.year && month == d.month && ... ;
        } else return false;
    }
    public int hashCode(){ return 512*year + 32*month + day; }
}
```

# Jellemző hiba

```
package java.lang;
public class Object {
    ...
    public boolean equals( Object that ){ return this == that; }
    public int hashCode(){ ... }
}
```

## Fordítási hiba a @Override-nak köszönhetően

```
public class Date {
    ...
    @Override public boolean equals( Date that ){
        return that != null && year == that.year && && ... ;
    }
    public int hashCode(){ return 512*year + 32*month + day; }
}
```

# Nagyon valószínű, hogy bug, és egyben rossz gyakorlat

```
package java.lang;
public class Object {
    ...
    public boolean equals( Object that ){ return this == that; }
    public int hashCode(){ ... }
}
```

## Túlterhelés, nincs dinamikus kötés

```
public class Date {
    ...
    public boolean equals( Date that ){
        return that != null && year == that.year && && ... ;
    }
    public int hashCode(){ return 512*year + 32*month + day; }
}
```