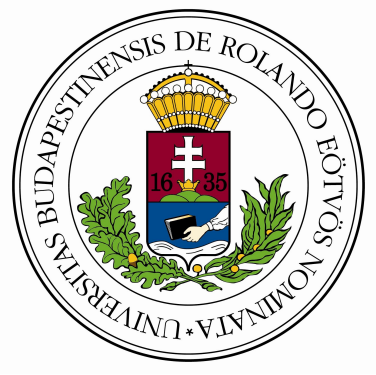
Programozási nyelvek – Java

Altípusos polimorfizmus



Kozsik Tamás

ELTE Eötvös Loránd Tudományegyetem

Az öröklődés két aspektusa

- Kódöröklés
- Altípusképzés

Outline

Öröklődés: altípusképzés

$$A \ \Delta \ B \ \Rightarrow \ A <: B$$

Öröklődés: altípusképzés

$$A \ \Delta \ B \ \Rightarrow \ A <: B$$

```
public class ExactTime extends Time { ... }
```

- Az `ExactTime` mindent tud, amit a `Time`
- Amit lehet `Time`-mal, lehet `ExactTime`-mal is
- `ExactTime` $<:$ `Time`

Öröklődés: altípusképzés

$$A \ \Delta \ B \ \Rightarrow \ A <: B$$

```
public class ExactTime extends Time { ... }
```

- Az `ExactTime` mindent tud, amit a `Time`
- Amit lehet `Time`-mal, lehet `ExactTime`-mal is
- `ExactTime` $<:$ `Time`

- $\forall T$ osztályra : $T <:$ `java.lang.Object`

Altípus

```
public class Time {
    ...
    public void aMinutePassed(){ ... }
    public boolean sameHourAs( Time that ){ ... }
}
```

```
public class ExactTime extends Time {
    ...
    public boolean isEarlierThan( ExactTime that ){ ... }
}
```

```
ExactTime time = new ExactTime();          // 0:00:00
time.aMinutePassed();                      // 0:01:00
time.sameHourAs( new ExactTime() )         // true
```

Liskov-féle helyettesítési elv

LSP: Liskov's Substitution Principle

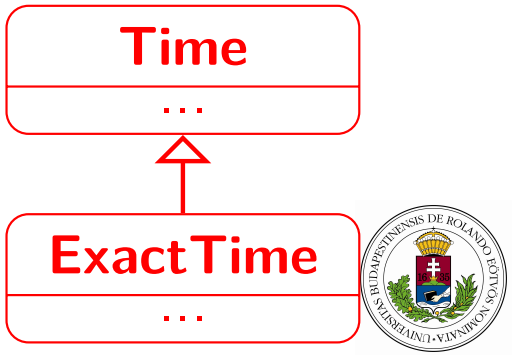
Egy A típus altípusa a B (bázis-)típusnak, ha az A egyedeit használhatjuk a B egyedei helyett, anélkül, hogy ebből baj lenne.

Polimorf referenciák

```
public class Time {
    ...
    public void aMinutePassed(){ ... }
    public boolean sameHourAs( Time that ){ ... }
}
```

```
public class ExactTime extends Time {
    ...
    public boolean isEarlierThan( ExactTime that ){ ... }
}
```

```
ExactTime time1 = new ExactTime();
Time      time2 = new ExactTime();  // upcast
time2.sameHourAs( time1 )
```



Statikus és dinamikus típus

Statikus típus: változó vagy paraméter *deklarált* típusa

- A programszövegből következik
- Állandó
- A fordítóprogram ez alapján típusellenőriz

```
Time time
```

Statikus és dinamikus típus

Statikus típus: változó vagy paraméter *deklarált* típusa

- A programszövegből következik
- Állandó
- A fordítóprogram ez alapján típusellenőriz

```
Time time
```

Dinamikus típus: változó vagy paraméter *tényleges* típusa

- Futási időben derül ki
- Változékony
- A statikus típus altípusa

```
time = ... ? new ExactTime() : new Time()
```

Outline

Felüldefiniálás

```
package java.lang;                          // java.lang.Object@4f324b5c
public class Object { ... public String toString(){ ... } ...  }
```

```
public class Time {
    ...
    @Override public String toString(){    // 8:05
        return String.format("%1$d:%2$02d", hour, minute);
    }
}
```

```
public class ExactTime extends Time {
    ...
    @Override public String toString(){    // 8:05:17
        return super.toString() + String.format(":%1$02d", second);
    }
}
```

Túlterhelés versus felüldefiniálás

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)

```
ExactTime e = new ExactTime();
Time      t = e;
Object    o = t;

System.out.println( e.toString() );     // 0:00:00
System.out.println( t.toString() );     // 0:00:00
System.out.println( o.toString() );     // 0:00:00
```

Példánymetódus hívásánál a 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 Time();
o.setHour(8);              // fordítási hiba
```

Dinamikus típus

Melyik implementációját egy felüldefiniált műveletnek?

```
Object o = new Time();
System.out.println(o);  // toString() impl. kiválasztása
```

- Dinamikus típusellenőrzés

Példa öröklődésre

```
package company.hr;
public class Employee {
    String name;
    int basicSalary;
    java.time.ZonedDateTime startDate;
    ...
}
```

Példa öröklődésre

```
package company.hr;
public class Employee {
    String name;
    int basicSalary;
    java.time.ZonedDateTime startDate;
    ...
}
```

```
package company.hr;
import java.util.*;
public class Manager extends Employee {
    final HashSet<Employee> underlings = new HashSet<>();
    ...
}
```

Szülőosztály

```
package company.hr;
import java.time.ZonedDateTime;
import static java.time.temporal.ChronoUnit.YEARS;
public class Employee {
    ...
    private ZonedDateTime startDate;
    public int yearsInService(){
        return (int) startDate.until(ZonedDateTime.now(), YEARS);
    }
    private static int bonusPerYearInService = 0;
    public int bonus(){
        return yearsInService() * bonusPerYearInService;
    }
}
```

Gyermekosztály

```
package company.hr;
import java.util.*;
public class Manager extends Employee {
   // megörökölt: startDate, yearsInService() ...
   ...
   private final HashSet<Employee> underlings = new HashSet<>();
   public void addUnderling( Employee underling ){
      underlings.add(underling);
   }
   private static int bonusPerUnderling = 0;
   @Override public int bonus(){
      return underlings.size() * bonusPerUnderling + super.bonus();
   }
}
```

Dinamikus kötés megörökölt metódusban is!

```
public class Employee {
   ...
   private int basicSalary;
   public int bonus(){
      return yearsInService() * bonusPerYearInService;
   }
   public int salary(){ return basicSalary + bonus(); }
}
```

```
public class Manager extends Employee {
   ...
   @Override public int bonus(){
      return underlings.size() * bonusPerUnderling + super.bonus();
   }
}
```

Dinamikus kötés megörökölt metódusban is!

```
Employee jack = new Employee("Jack", 10000);
Employee pete = new Employee("Pete", 12000);
Manager eve = new Manager("Eve", 12000);
Manager joe = new Manager("Joe", 12000);
eve.addUnderling(jack);
joe.addUnderling(eve);              // polimorf formális paraméter
joe.addUnderling(pete);

Employee[] company = {joe, eve, jack, pete};    // <-- heterogén
                                                // adatszerkezet
int totalSalaryCosts = 0;
for( Employee e: company ){
   totalSalaryCosts += e.salary();
}
```

Dinamikus kötés

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

Mező és osztályszintű metódus nem definiálható felül

```
class Base {
    int field = 3;
    int iMethod(){ return field; }
    static int sMethod(){ return 3; }
}

class Sub extends Base {
    int field = 33;                          // elfedés
    static int sMethod(){ return 33; }       // elfedés
}
```

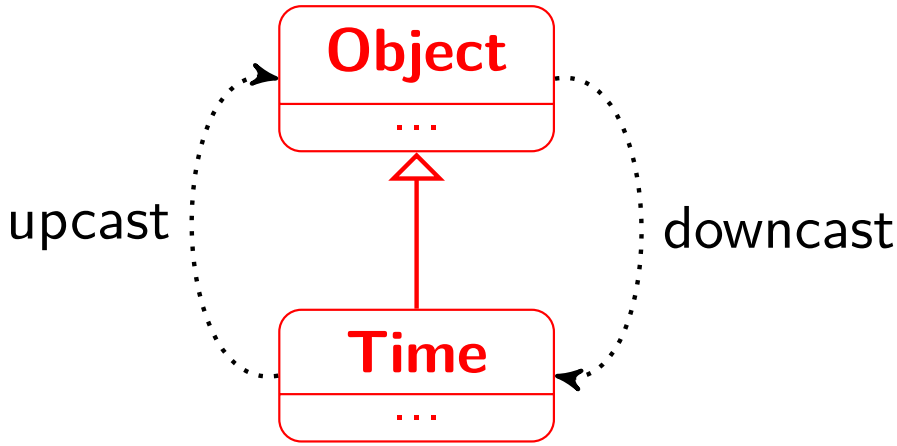
```
Sub sub = new Sub();          Base base = sub;

sub.sMethod()  ==  33         base.sMethod()  ==  3
sub.field      ==  33         base.field      ==  3
sub.iMethod()  ==  3          base.iMethod()  ==  3
```

Outline

Konverziók referenciatípusokon

- Automatikus (upcast) – altípusosság
- Explicit (downcast) – type-cast operátor



Típuskényszerítés (downcast)

- A „`(Time)o`'' kifejezés statikus típusa `Time`

Típuskényszerítés (downcast)

- A „`(Time)o`'' kifejezés statikus típusa `Time`
- Ha `o` dinamikus típusa `Time`:

```
Object o = new Time(3,20);
o.aMinutePassed();              // fordítási hiba
((Time)o).aMinutePassed();      // lefordul, működik
```

Típuskényszerítés (downcast)

- A „`(Time)o`'' kifejezés statikus típusa `Time`
- Ha `o` dinamikus típusa `Time`:

```
Object o = new Time(3,20);
o.aMinutePassed();            // fordítási hiba
((Time)o).aMinutePassed();    // lefordul, működik
```

- Ha nem, `ClassCastException` lép fel

```
Object o = "Három óra húsz";
o.aMinutePassed();            // fordítási hiba
((Time)o).aMinutePassed();    // futási hiba
```

Dinamikus típusellenőrzés

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

instanceof-operátor

```
Object o = new ExactTime(3,20,0);
...
if( o instanceof Time ){
    ((Time)o).aMinutePassed();
}
```

- Kifejezés dinamikus típusa altípusa-e a megadottnak

instanceof-operátor

```
Object o = new ExactTime(3,20,0);
...
if( o instanceof Time ){
    ((Time)o).aMinutePassed();
}
```

- Kifejezés dinamikus típusa altípusa-e a megadottnak
- Statikus típusa ne zárja ki a megadottat

```
"apple" instanceof Integer
```

instanceof-operátor

```
Object o = new ExactTime(3,20,0);
...
if( o instanceof Time ){
    ((Time)o).aMinutePassed();
}
```

- Kifejezés dinamikus típusa altípusa-e a megadottnak
- Statikus típusa ne zárja ki a megadottat

```
"apple" instanceof Integer
```

- null-ra false

Dinamikus típus ábrázolása futás közben

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

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

Outline

Típuskonverziók primitív típusok között

Automatikus típuskonverzió (tranzitív)

- `byte` < `short` < `int` < `long`
- `long` < `float`
- `float` < `double`
- `char` < `int`
- `byte b = 42;` és `short s = 42;` és `char c = 42;`

Explicit típuskényszerítés (type cast)

```
int i = 42;
short s = (short)i;
```

Puzzle 3: Long Division (Bloch & Gafter: Java Puzzlers)

```
public class LongDivision {
    public static void main(String[] args) {
        final long MICROS_PER_DAY = 24 * 60 * 60 * 1000 * 1000;
        final long MILLIS_PER_DAY = 24 * 60 * 60 * 1000;
        System.out.println(MICROS_PER_DAY / MILLIS_PER_DAY);
    }
}
```

Csomagoló osztályok

Implicit importált (`java.lang`), immutable osztályok

- `java.lang.Boolean` – `boolean`
- `java.lang.Character` – `char`
- `java.lang.Byte` – `byte`
- `java.lang.Short` – `short`
- `java.lang.Integer` – `int`
- `java.lang.Long` – `long`
- `java.lang.Float` – `float`
- `java.lang.Double` – `double`

java.lang.Integer interfésze (részlet)

```
static int MAX_VALUE    // 2^31-1
static int MIN_VALUE    // -2^31

static int compare( int x, int y )    // 3-way comparison
static int max( int x, int y )
static int min( int x, int y )
static int parseInt( String str [, int radix] )
static String toString( int i [, int radix] )
static Integer valueOf( int i )

int compareTo( Integer that )        // 3-way comparison
int intValue()
```

Auto-(un)boxing

- Automatikus kétirányú konverzió
- Primitív típus és a csomagoló osztálya között

```
Integer ref = 42;
int pri = ref;

Integer sum = ref + pri;
```

```
Integer ref = Integer.valueOf(42);
int pri = ref.intValue();

Integer sum = Integer.valueOf (
                ref.intValue()
                + pri           );
```

Auto-(un)boxing + generikusok

```
ArrayList<Integer> numbers = new ArrayList<>();
numbers.add(7);
int seven = numbers.get(0);
```

```
ArrayList<Integer> numbers = new ArrayList<>();
numbers.add( Integer.valueOf(7) );
int seven = numbers.get(0).intValue();
```

Számolás egész számokkal

```
int n = 10;
int fact = 1;
while( n > 1 ){
    fact *= n;
    --n;
}
```

Rosszul használt auto-(un)boxing

```
Integer n = 10;
Integer fact = 1;
while( n > 1 ){
    fact *= n;
    --n;
}
```

Jelentés

```
Integer n = Integer.valueOf(10);
Integer fact = Integer.valueOf(1);
while( n.intValue() > 1 ){
    fact = Integer.valueOf(fact.intValue() * n.intValue());
    n = Integer.valueOf(n.intValue() - 1);
}
```

Öröklődés – altípusosság

- `class A extends B` …
- $A <: B$
- $\forall T: \quad T <:$ `java.lang.Object`

Automatikus „konverzió'' bázistípusra (upcast)

```
String str = "Java";
Object o = str;  // OK
str = o;         // fordítási hiba
```

Kényszerítés altípusra (downcast)

```
String str = "Java";
Object o = str;  // OK
str = (String)o; // OK, dinamikus típusellenőrzés
```

ClassCastException

```
String str = "Java";
Object o = str;
Integer i = (Integer)o;
```

Típusba tartozás (altípusosság)

```
String str = "Java";
Object o = str;
Integer i = (o instanceof Integer) ? (Integer)o : null;
```

Dinamikus típusra típusegyezés

```
String str = "Java";
Object o = str;
Integer i = o.getClass().equals(Integer.class) ? (Integer)o : null;
```