



Programozási nyelvek Java

Polimorfizmus

Kozsik Tamás



Polimorfizmus: áttekintés

Polimorfizmus: típus/metódus/objektum különféle módokon való használata

Kategorizálás

- univerzális polimorfizmus (*korlátlanul kiterjeszthetően* sok mód)
 - ◇ parametrikus polimorfizmus: sablonok
 - ◇ altípusos polimorfizmus: öröklődés
- ad hoc polimorfizmus (csak *véges számú*, előre rögzített mód)
 - ◇ túlterhelés
 - ◇ explicit típuskényszerítés (cast)



Egy korábbi példa

```
public class Receptionist {
    public Time[] readWakeupTimes(String[] fnames) {
        Time[] times = new Time[fnames.length];

        for (int i = 0; i < fnames.length; ++i) {
            try {
                times[i] = readTime(fnames[i]);
            } catch (java.io.IOException e) {
                times[i] = null;    // no-op
                System.err.println("Could not read " + fnames[i]);
            }
        }

        return times; // maybe sort times before returning?
    }
    ...
}
```





A null értékek kiszűrése

```
public class Receptionist {
    public Time[] readWakeupTimes(String[] fnames) {
        Time[] times = new Time[fnames.length];
        int len = 0;
        for (int i = 0; i < fnames.length; ++i) {
            try {
                times[len] = readTime(fnames[i]);
                ++len;
            } catch (java.io.IOException e) {
                System.err.println("Could not read " + fnames[i]);
            }
        }
        return java.util.Arrays.copyOf(times, len); // possibly
    } // sorted
    ...
}
```



Tömbök előnyei és hátrányai

- Elemek hatékony elérése (indexelés)
- Szintaktikus támogatás a nyelvben (indexelés, tömbliterál)
- Fix hossz: létrehozáskor
 - ◇ Bővítéshez új tömb létrehozása + másolás
 - ◇ Törléshez új tömb létrehozása + másolás



Alternatíva: `java.util.ArrayList`

kényelmes szabványos könyvtár, hasonló belső működés

```
String[] names = { "Tim",  
                  "Jerry" };
```

```
names[0] = "Tom";  
String mouse = names[1];
```

```
String[] trio = new String[3];  
trio[0] = names[0];  
trio[1] = names[1];  
trio[2] = "Spike";  
names = trio;
```



Alternatíva: `java.util.ArrayList`

kényelmes szabványos könyvtár, hasonló belső működés

```
String[] names = { "Tim",  
                  "Jerry" };
```

```
names[0] = "Tom";
```

```
String mouse = names[1];
```

```
String[] trio = new String[3];
```

```
trio[0] = names[0];
```

```
trio[1] = names[1];
```

```
trio[2] = "Spike";
```

```
names = trio;
```

```
ArrayList<String> names =  
    new ArrayList<>();
```

```
names.add("Tim");
```

```
names.add("Jerry");
```

```
names.set(0, "Tom");
```

```
String mouse = names.get(1);
```

```
names.add("Spike");
```





Az előző példa átalakítva

```
public class Receptionist {
    ...
    public ArrayList<Time> readWakeupTimes(String[] fnames) {
        ArrayList<Time> times = new ArrayList<Time>();
        for (int i = 0; i < fnames.length; ++i) {
            try {
                times.add(readTime(fnames[i]));
            } catch (java.io.IOException e) {
                System.err.println("Could not read " + fnames[i]);
            }
        }
        return times; // possibly sort before returning
    }
}
```


Paraméterezett típus

```
ArrayList<Time> times
```

```
Time[] times
```

```
Time times[]
```

Generikus osztály

Nem pont így, de hasonlóan...!

```
package java.util;  
public class ArrayList<T> {  
    public ArrayList() { ... }  
    public T get(int index) { ... }  
    public void set(int index, T item) { ... }  
    public void add(T item) { ... }  
    public T remove(int index) { ... }  
    ...  
}
```



Használatkor típusparaméter megadása

```
import java.util.ArrayList;
```

```
...
```

```
ArrayList<Time> times;
```

```
ArrayList<String> names = new ArrayList<String>();
```

```
ArrayList<String> namez = new ArrayList<>();
```

Generikus metódus

```
import java.util.*;

public class Main {
    public static <T> void reverse(T[] array) {
        int lo = 0, hi = array.length-1;
        while (lo < hi) {
            T tmp = array[hi];
            array[hi] = array[lo];
            array[lo] = tmp;
            ++lo; --hi;
        }
    }

    public static void main(String[] args) {
        reverse(args);
        System.out.println(Arrays.toString(args));
    }
}
```



Generikus metódus kézzel megadott típusparaméterrel

```
import java.util.*;

public class Main {
    public static <T> void reverse(T[] array) {
        int lo = 0, hi = array.length-1;
        while (lo < hi) {
            T tmp = array[hi];
            array[hi] = array[lo];
            array[lo] = tmp;
            ++lo; --hi;
        }
    }

    public static void main(String[] args) {
        Main.<String>reverse(args);
        System.out.println(Arrays.toString(args));
    }
}
```



Parametrikus polimorfizmus

- Több típusra is működik ugyanaz a kód
 - ◇ Haskell: függvény
 - ◇ Java: típus (osztály), metódus
- Típussal paraméterezhető kód
 - ◇ Haskell: bármilyen típussal
 - ◇ Java: referenciatípusokkal



Sablon (generics): ArrayList

Típusparaméter

Primitív típussal helytelen

```
ArrayList<int> numbers
```



Típusparaméter

Primitív típussal helytelen

```
ArrayList<int> numbers
```

Referenciatípussal helyes

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


Típusparaméter

Primitív típussal helytelen

```
ArrayList<int> numbers
```

Referenciatípussal helyes

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

Furcsamód ez is helyes

```
ArrayList<Integer> numbers = new ArrayList<>();  
numbers.add(42);           // auto-boxing: int -> Integer  
int n42 = numbers.get(1);  // auto-unboxing: Integer -> int
```



Adatszerkezetek a java.util csomagban

Sorozat

```
ArrayList<String> colors = new ArrayList<>();  
colors.add("red"); colors.add("white"); colors.add("red");  
String third = colors.get(2);
```

Adatszerkezetek a java.util csomagban

Sorozat

```
ArrayList<String> colors = new ArrayList<>();  
colors.add("red"); colors.add("white"); colors.add("red");  
String third = colors.get(2);
```

Halmaz

```
HashSet<String> colors = new HashSet<>();  
colors.add("red"); colors.add("white"); colors.add("red");  
int two = colors.size();
```

Adatszerkezetek a java.util csomagban

Sorozat

```
ArrayList<String> colors = new ArrayList<>();  
colors.add("red"); colors.add("white"); colors.add("red");  
String third = colors.get(2);
```

Halmaz

```
HashSet<String> colors = new HashSet<>();  
colors.add("red"); colors.add("white"); colors.add("red");  
int two = colors.size();
```

Leképezés

```
HashMap<String,String> colors = new HashMap<>();  
colors.put("red","piros"); colors.put("white","fehér");  
String whiteHu = colors.get("white");
```

Generikus osztály

```
public class ArrayList<T> {  
    public ArrayList() { ... }  
    public T get(int index) { ... }  
    public void set(int index, T item) { ... }  
    public void add(T item) { ... }  
    public T remove(int index) { ... }  
    ...  
}
```



Implementálás

```
public class ArrayList<T> {  
    private T[] data;  
    private int size = 0;  
    ...  
    public T get(int index) {  
        if (index < size) return data[index];  
        else throw new IndexOutOfBoundsException();  
    }  
    ...  
}
```



Implementálás

```
public class ArrayList<T> {  
    private T[] data;  
    private int size = 0;  
    ...  
    public void add(T item) {  
        if (size == data.length) {  
            data = java.util.Arrays.copyOf(data, data.length+1);  
        }  
        data[size] = item;  
        ++size;  
    }  
    ...  
}
```



Allokálás: fordítási hiba

```
public class ArrayList<T> {  
    private T[] data;  
    private int size = 0;  
    ...  
    public ArrayList() { this(256); }  
    public ArrayList(int initialCapacity) {  
        data = new T[initialCapacity];  
    }  
    ...  
}
```

ArrayList.java:6: error: generic array creation
data = new T[initialCapacity];



Típustörlés

type erasure

- Típusparaméter: statikus típusellenőrzéshez
- Tárgykód: típusfüggetlen (mint a Haskellben)
- Más, mint a C++ *template*
- Kompatibilitási okok
- Futás közben nem használható a típusparaméter



Így képzelhetjük el a tárgykódot

```
public class ArrayList {  
    private Object[] data;  
    ...  
    public ArrayList() { ... }  
    public Object get(int index) { ... }  
    public void set(int index, Object item) { ... }  
    public void add(Object item) { ... }  
    public Object remove(int index) { ... }  
    ...  
}
```



Kompatibilitás: nyers típus

raw type

```
import java.util.ArrayList;
...
ArrayList<String> paraméteres = new ArrayList<>();
paraméteres.add("Romeo");
paraméteres.add(12);           // fordítási hiba
String s = paraméteres.get(0);
```



Kompatibilitás: nyers típus

raw type

```
import java.util.ArrayList;
...
ArrayList<String> paraméteres = new ArrayList<>();
paraméteres.add("Romeo");
paraméteres.add(12);           // fordítási hiba
String s = paraméteres.get(0);

ArrayList nyers = new ArrayList();
nyers.add("Romeo");
nyers.add(12);
Object o = nyers.get(0);
```

Allokálás: még mindig rosszul

```
public class ArrayList<T> {  
    private T[] data;  
    private int size = 0;  
    ...  
    public ArrayList() { this(256); }  
    public ArrayList(int initialCapacity) {  
        data = new Object[initialCapacity];  
    }  
    ...  
}
```

```
ArrayList.java:6: error: incompatible types:  
                    Object[] cannot be converted to T[]  
    data = new Object[initialCapacity];  
            ^
```

where T is a type-variable:



Allokálás – így már működik

```
public class ArrayList<T> {  
    private T[] data;  
    private int size = 0;  
    ...  
    public ArrayList() { this(256); }  
    public ArrayList(int initialCapacity) {  
        data = (T[])new Object[initialCapacity];  
    }  
    ...  
}
```

```
javac ArrayList.java
```

Note: ArrayList.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

Allokálás – így már működik, de azért nem az igazi...

```
public class ArrayList<T> {  
    private T[] data;  
    private int size = 0;  
    ...  
    public ArrayList() { this(256); }  
    public ArrayList(int initialCapacity) {  
        data = (T[])new Object[initialCapacity];  
    }  
    ...  
}
```

```
javac -Xlint:unchecked ArrayList.java
```

```
ArrayList.java:6: warning: [unchecked] unchecked cast  
    required: T[]           found: Object[]
```



Kényszerítsünk máshol?

```
public class ArrayList<T> {  
    private Object[] data;  
    private int size = 0;  
    ...  
    public T get(int index) {  
        if (index < size) return (T)data[index];  
        else throw new IndexOutOfBoundsException();  
    }  
    ...  
}
```

```
javac -Xlint:unchecked ArrayList.java
```

```
ArrayList.java:10: warning: [unchecked] unchecked cast  
    required: T           found: Object
```


Warning-mentesen

```
public class ArrayList<T> {  
    private Object[] data;  
    private int size = 0;  
    ...  
    @SuppressWarnings("unchecked")  
    public T get(int index) {  
        if (index < size) return (T)data[index];  
        else throw new IndexOutOfBoundsException();  
    }  
    ...  
}
```

Öröklődés (inheritance)

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

- Egy típust egy másik típusból származtatunk
 - ◇ Csak a különbségeket kell megadni: $A \Delta B$
 - ◇ Újrafelhasználás

Öröklődés (inheritance)

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

- Egy típust egy másik típusból származtatunk
 - ◇ Csak a különbségeket kell megadni: $A \Delta B$
 - ◇ Újrafelhasználás
- Itt: az A a gyermekosztálya a B szülőosztálynak
 - ◇ child class
 - ◇ parent class

Öröklődés (inheritance)

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

- Egy típust egy másik típusból származtatunk
 - ◇ Csak a különbségeket kell megadni: $A \Delta B$
 - ◇ Újrafelhasználás
- Itt: az A a gyermekosztálya a B szülőosztálynak
 - ◇ child class
 - ◇ parent class
- Transzitivitás: leszármazott osztály – ősz osztály
 - ◇ alosztály: subclass, derived class
 - ◇ bázisosztály: super class, base class

Öröklődés (inheritance)

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

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

Példa öröklődésre

```
public class Time {  
    private int hour, min;    // initialized to 00:00  
    public int getHour() { ... }  
    public int getMin() { ... }  
    public void setHour(int hour) { ... }  
    public void setMin(int min) { ... }  
    public void aMinPassed() { ... }  
}
```

Példa öröklődésre

```
public class Time {  
    private int hour, min;    // initialized to 00:00  
    public int getHour() { ... }  
    public int getMin() { ... }  
    public void setHour(int hour) { ... }  
    public void setMin(int min) { ... }  
    public void aMinPassed() { ... }  
}
```

```
public class ExactTime extends Time {  
    private int sec;    // initialized to 00  
    public int getSec() { ... }  
    public void setSec(int sec) { ... }  
    public boolean earlierThan(ExactTime that) { ... }  
}
```

Implicit szülőosztály

```
public class Time extends java.lang.Object {
    private int hour, min;    // initialized to 00:00
    public int getHour() { ... }
    public int getMin() { ... }
    public void setHour(int hour) { ... }
    public void setMin(int min) { ... }
    public void aMinPassed() { ... }
}
```

```
public class ExactTime extends Time {
    private int sec;          // initialized to 00
    public int getSec() { ... }
    public void setSec(int sec) { ... }
    public boolean earlierThan(ExactTime that) { ... }
}
```


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) { ... }  
    ...  
}
```

A konstruktorok függetlenek az öröklődéstől

```
public class Time {  
    private int hour, min;  
    public Time(int hour, int min) {  
        if (hour < 0 || hour > 23 || min < 0 || min > 59)  
            throw new IllegalArgumentException();  
        this.hour = hour;  
        this.min = min;  
    }  
    ...  
}
```

A konstruktorok függetlenek az öröklődéstől

```
public class Time {  
    private int hour, min;  
    public Time(int hour, int min) {  
        if (hour < 0 || hour > 23 || min < 0 || min > 59)  
            throw new IllegalArgumentException();  
        this.hour = hour;  
        this.min = min;  
    }  
    ...  
}
```

```
public class ExactTime extends Time {  
    private int sec;
```

A konstruktorok függetlenek az öröklődéstől

```
public class Time {  
    private int hour, min;  
    public Time(int hour, int min) {  
        if (hour < 0 || hour > 23 || min < 0 || min > 59)  
            throw new IllegalArgumentException();  
        this.hour = hour;  
        this.min = min;  
    }  
    ...  
}
```

```
public class ExactTime extends Time {  
    private int sec;  
    public ExactTime(int hour, int min, int sec) { ? }  
}
```

A `Overmekosztály`ba is kell konstruktort írni!

```
public class Time {  
    private int hour, min;  
    public Time(int hour, int min) { ... }  
    ...  
}
```

```
public class ExactTime extends Time {  
    private int sec;  
    public ExactTime(int hour, int min, int sec) {
```

A gyermekosztályba is kell konstruktort írni!

```
public class Time {
    private int hour, min;
    public Time(int hour, int min) { ... }
    ...
}
```

```
public class ExactTime extends Time {
    private int sec;
    public ExactTime(int hour, int min, int sec) {
        super(hour, min); // meghívandó a szülő konstruktora
        if (sec < 0 || sec > 59)
            throw new IllegalArgumentException();
        this.sec = sec;
    }
}
```



super(...)-konstruktorhívás

- Szülőosztály valamelyik konstruktora
- Megörökölt tagok inicializálása
- Legelső utasítás kell legyen

super(...)-konstruktorhívás

- Szülőosztály valamelyik konstruktora
- Megörökölt tagok inicializálása
- Legelső utasítás kell legyen

Hibás!!!

```
public class ExactTime extends Time {  
    private int second;  
    public ExactTime(int hour, int minute, int second) {  
        if (second < 0 || second > 59)  
            throw new IllegalArgumentException();  
        super(hour, minute);  
        this.second = second;  
    }  
}
```


Miért helyes? Hiányzik a **super**?!

```
public class Time extends Object {  
    private int hour, min;  
    public Time(int hour, int min) {  
        if (hour < 0 || hour > 23 || min < 0 || min > 59)  
            throw new IllegalArgumentException();  
        this.hour = hour;  
        this.min = min;  
    }  
    ...  
}
```

Implicit `super()`-hívás

```
public class Time extends Object {
    private int hour, min;
    public Time(int hour, int min) {
        super();
        if (...) throw ...;
        this.hour = hour;
        this.min = min;
    }
    ...
}
```

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



Implicit szülőosztály, implicit konstruktor, implicit super

```
class A {}
```



Implicit szülőosztály, implicit konstruktor, implicit super

```
class A {}
```

```
class A extends java.lang.Object {  
    A() {  
        super();  
    }  
}
```



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(); }
}
```



Öröklődéssel definiált osztály

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



Példánymetódus felüldefiniálása

újradefiniálás, felüldefiniálás

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

Példánymetódus felüldefiniálása

újradefiniálás, felüldefiniálás

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

```
public class Time {  
    ...  
    public String toString() {  
        return hour + ":" + min; // 8:5  
    }  
}
```



Példánymetódus felüldefiniálása

újradefiniálás, felüldefiniálás

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

Példánymetódus felüldefiniálása

újradefiniálás, felüldefiniálás

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

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



Az ajánlott @Override annotációval

újradefiniálás, redefinition, overriding

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

Az ajánlott `@Override` annotációval

újradefiniálás, redefinition, overriding

```

package java.lang;

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

```

```

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

```



super.toString() hívása

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

super.toString() hívása

```

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

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

```


super.toString() hívása

```

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

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

public class ExactTime extends Time {
    ...
    @Override public String toString() {           // 8:05:17
        return "%1:%2$02d".formatted(super.toString(), sec);
    }
}

```



Túlterhelés és felüldefiniálás

```
package java.lang;

public final class Integer extends Number {
    ...
    public static      int parseInt(String str)          { ... }
    ...
    public @Override 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

Statikus és dinamikus kiválasztódás

System.out

```
public void println(    int value) { ... }
public void println(Object value) { ... } //value.toString()
...
```

```
System.out.println(7);                // 7
System.out.println("Samurai");        // Samurai
System.out.println(new Time(21,30));   // 21:30
```





protected

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

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

protected

A private tagok nem hívhatók 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  
    }  
}
```



protected

„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

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;
    }
}
```

IK



Absztrakció: egységbe záras és információ elrejtése

```
public class Rational {
    private final int numerator, denominator;
    private static int gcd(int a, int b)          { ... }
    private void simplify()                      { ... }
    public Rational(int numerator, int denominator) { ... }
    public Rational(int value)                   { this(value, 1); }
    public int getNumerator()                    { return numerator; }
    public int getDenominator()                 { return denominator; }
    public Rational times(Rational that) { ... }
    public Rational times(int that)            { ... }
    public Rational plus(Rational that) { ... }
}
```



Absztrakció: egységbe zárás és információ elrejtése

```
public class Rational {
    private final int numerator, denominator;
    private static int gcd(int a, int b)          { ... }
    private void simplify()                      { ... }
    public Rational(int numerator, int denominator) { ... }
    public Rational(int value)                   { this(value, 1); }
    public int getNumerator()                    { return numerator; }
    public int getDenominator()                 { return denominator; }
    public Rational times(Rational that) { ... }
    public Rational times(int that)           { ... }
    public Rational plus(Rational that)  { ... }
}
```

- Osztály interfésze: minden public tartalom benne
 - ◊ A konstruktorai és a metódusai jellemzően beletartoznak
 - ▶ Csak a fejléc számít: `Rational times(Rational)`
 - ◊ Az adattagjai jellemzően nem tartoznak bele

Az interface-definíció

```
public interface Rational {  
    int getNumerator();  
    int getDenominator();  
  
    Rational times(Rational that);  
    Rational times(int that);  
    Rational plus(Rational that);  
    ...  
}
```

- interfész minden metódusa automatikusan **abstract** és **public**



Az interface-definíció tartalma

Példánymetódusok deklarációja: specifikáció és lezáró ;

```
int getNumerator();
```



Az interface-definíció tartalma

Példánymetódusok deklarációja: specifikáció és lezáró ;

```
int getNumerator();
```

- Példánymetódusok deklarációja
 - ◊ Esetleg **default** implementáció
- Konstansok definíciója: **public static final**
- Statikus metódus
- Beágyazott (tag-) típus

Generikus interfész

java/util/List.java

```
package java.util;  
public interface List<T> {  
    T get(int index);  
    void set(int index, T item);  
    void add(T item);  
    ...  
}
```

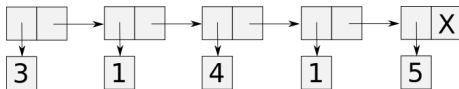
java/util/ArrayList.java

```
package java.util;  
public class ArrayList<T> implements List<T> {  
    public ArrayList() { ... }  
    @Override public T get(int index) { ... }  
    ...  
}
```

Láncolt ábrázolás

```
package java.util;
```

```
public class LinkedList<T> implements List<T> {
    private T head;
    private LinkedList<T> tail;
    public LinkedList() { ... }
    @Override public T get(int index) { ... }
    @Override public void set(int index, T item) { ... }
    @Override public void add(T item) { ... }
    ...
}
```





interface megvalósítása

```
public interface Rational {  
    int getNumerator();  
    int getDenominator();  
    Rational times(Rational that);  
}
```

```
public class Fraction implements Rational {  
    private final int numerator, denominator;  
    public Fraction(int numerator, int denominator) { ... }  
    public int getNumerator() { return numerator; }  
    public int getDenominator() { return denominator; }  
    public Rational times(Rational that) { ... }  
}
```

Több megvalósítás

Fraction.java

```
public class Fraction implements Rational {  
    private final int numerator, denominator;  
    public Fraction(int numerator, int denominator) { ... }  
    public int getNumerator() { return numerator; }  
    public int getDenominator() { return denominator; }  
    public Rational times(Rational that) { ... }  
}
```

Simplified.java

```
public class Simplified implements Rational {  
    ...  
    public int getNumerator() { ... }  
    public int getDenominator() { ... }  
    Rational times(Rational that) { ... }
```

Altípusosság

```
class Fraction implements Rational { ... }  
class ArrayList<T> implements List<T> { ... }  
class LinkedList<T> implements List<T> { ... }
```

- Fraction <: Rational
- Simplified <: Rational
- Minden T-re: ArrayList<T> <: List<T>
- Minden T-re: LinkedList<T> <: List<T>

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.





Az interface egy típus

```
List<String> names;
```

```
static List<String> noDups(List<String> names) {  
    List<String> retval = ...;  
    ...  
}
```



Nem példányosítható

Nem készíthető példány **interface** típusból

- Nincsen ábrázolása
- Nincsen konstruktora

```
List<String> names = new List<String>(); // fordítási hiba
```

Nem példányosítható

Nem készíthető példány **interface** típusból

- Nincsen ábrázolása
- Nincsen konstruktora

```
List<String> names = new List<String>(); // fordítási hiba
```

Az osztály is egy típus, és példányosítható

```
ArrayList<String> names = new ArrayList<String>();
ArrayList<String> nicks = new ArrayList<>();
```



Nem példányosítható

Nem készíthető példány **interface** típusból

- Nincsen ábrázolása
- Nincsen konstruktora

```
List<String> names = new List<String>(); // fordítási hiba
```

Az osztály is egy típus, és példányosítható

```
ArrayList<String> names = new ArrayList<String>();
ArrayList<String> nicks = new ArrayList<>();
```

Jó stílus: típusozás **interface**-szel, példányosítás osztállyal

```
List<String> names = new ArrayList<>();
```


Statikus és dinamikus típus

Változó (vagy paraméter) „deklarált”, illetve „tényleges” típusa

```
List<String> names = new ArrayList<>();
```

```
static List<String> noDups(List<String> names) {  
    ... names ...  
}
```

```
List<String> shortList = noDups(names);
```



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
    ...
}
```



Részleges megvalósítás

```
public abstract class AbstractCollection<E> ... {
    ...
    public abstract int size();
    public boolean isEmpty() {
        return size() == 0;
    }
    public abstract Iterator<E> iterator();
    public boolean contains(Object o) {
        Iterator<E> iterator = iterator();
        while (iterator.hasNext()) {
            E e = iterator.next();
            if (o==null ? e==null : o.equals(e)) return true;
        }
        return false;
    }
}
```

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
}
```

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 { ... }
```

Hipotetikusan

```
class Base1 {  
    int x;  
    void setX(int x) { this.x = x; }  
    ...  
}  
  
class Base2 {  
    int x;  
    void setX(int x) { this.x = x; }  
    ...  
}  
  
class Sub extends Base1, Base2 { ... }
```

Hipotetikusan: diamond-shaped inheritance

```
class Base0 {  
    int x;  
    void setX(int x) { this.x = x; }  
    ...  
}  
  
class Base1 extends Base0 { ... }  
  
class Base2 extends Base0 { ... }  
  
class Sub extends Base1, Base2 { ... }
```




Különbség class és interface között

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

Többszörös öröklés interfészekből

```
interface Base1 {  
    abstract void setX(int x);  
    ...  
}  
  
interface Base2 {  
    abstract void setX(int x);  
    ...  
}  
  
class Sub implements Base1, Base2 {  
    void setX(int x) { ... }  
    ...  
}
```



Az öröklődés két aspektusa

- Kódöröklés
- Altípusképzés



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) { ... }
}
```

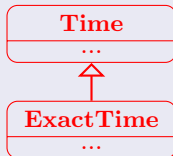
```
ExactTime time = new ExactTime();           // 0:00:00
time.aMinutePassed();                       // 0:01:00
time.sameHourAs(new ExactTime())           // true
```

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()`



Felüldefiniálás

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



Felüldefiniálás

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

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

Felüldefiniálás

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

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

public class ExactTime extends Time {
    ...
    @Override public String toString() {           // 8:05:17
        return "%1:%2$02d".formatted(super.toString(), sec);
    }
}
```



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() implementáció  
                        // 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> workers = 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 {
    // inherited: startDate, yearsInService() ...
    ...
    private final Set<Employee> workers = new HashSet<>();
    public void addWorker(Employee worker) {
        workers.add(worker);
    }
    private static int bonusPerWorker = 0;
    @Override public int bonus() {
        return workers.size() * bonusPerWorker + super.bonus();
    }
}
```

Dinamikus kötés megörökölt metódushoz 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 workers.size()*bonusPerWorker + 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.addWorker(jack);
joe.addWorker(eve);           // polimorf formális paraméter
joe.addWorker(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 osztálvszintű 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

Típuskonverziók primitív típusok között

Automatikus típuskonverzió (tranzitív)

- byte \rightarrow short \rightarrow int \rightarrow long
- long \rightarrow float
- float \rightarrow double
- char \rightarrow int
- byte b = 42; és short s = 42; és char c = 42;

Explicit típuskényszerítés (type cast)

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



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    // 231-1
static int MIN_VALUE    // -231
```

```
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

- 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  
);
```

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



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);  
    }  
}
```

null kicsomagolása

A kicsomagolás mindig működik

```
int val = 42;
```

```
Integer value = val;
```

null kicsomagolása

A becsomagolás mindig működik

```
int val = 42;  
Integer value = val;
```

A kicsomagolás *majdnem* mindig jól működik

```
Integer value = 42;  
int val = value;
```


null kicsomagolása

A becsomagolás mindig működik

```
int val = 42;  
Integer value = val;
```

A kicsomagolás *majdnem* mindig jól működik

```
Integer value = 42;  
int val = value;
```

... kivéve, amikor nem

```
Integer value = null;  
int val = value;           // NullPointerException
```



Költséges auto-(un)boxing

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

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

Költséges auto-(un)boxing

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

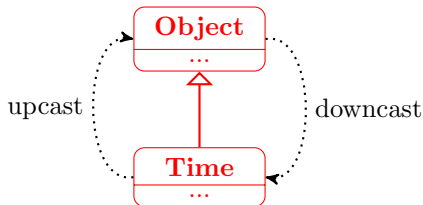
```
Integer n = 10;
Integer fact = 1;
while (n > 1) {
    fact *= n;
    --n;
}
```

Jelentése:

```
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);
}
```

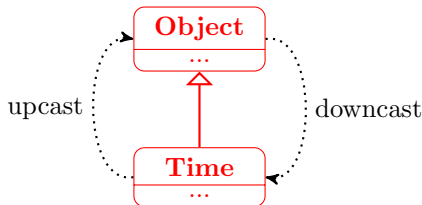
Konverziók referenciatípusokon

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



Konverziók referenciatípusokon

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



Öröklődés – altípusosság: `class A extends B ...`

- $A <: B$
- $\forall T : T <: \text{java.lang.Object}$



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.aMinPassed();           // fordítási hiba  
((Time)o).aMinPassed();   // lefordul, működik
```



Típuskényszerítés (downcast)

- A “(Time)o” kifejezés statikus típusa **Time**
- Ha o dinamikus típusa **Time**:

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

- Ha nem, **ClassCastException** lép fel

```
Object o = "Három óra húsz";
o.aMinPassed();           // fordítási hiba
((Time)o).aMinPassed();   // 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 t) {  
    t.aMinPassed();  
}
```

- Kifejezés dinamikus típusa altípusa-e a megadottnak

instanceof-operátor

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

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

```
"apple" instanceof Integer    // compilation error
```

instanceof-operátor

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

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

```
"apple" instanceof Integer    // compilation error
```

- 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
```

Típusba tartozás (altípusosság)

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

Dinamikus típusra típusegyeztetés

```
Integer i = o.getClass().equals(Integer.class) ?
              (Integer)o : null;
```