Java 5

Ráth István
rath@mit.bme.hu

Áttekintés

- Java 5 újdonságai
  - Generikus nyelvi eszközök
  - Foreach
  - Dobozolás
  - Enumerációk
  - Varargs
  - Annotációk
- Kitekintés: Java 6 eszközök

Generikus típusok

- Cél
  - Osztályok, metódusok konkrét típusmegkötés *nélkül*, mégis fordítási időben ellenőrzött (statikus) *típushelyességgel*
  - Vö: dinamikus tipizálás (*java.lang.Object*, *casting*)
- Megvalósítás: JSR-14
- Ős: generikus (meta)programozás, C++ template
  - Java: szűkített részhalmaz, „containers-of-type-T” (~STL *Container*)
- Referencia
  - http://java.sun.com/j2se/1.5/pdf/generics-tutorial.pdf

Generikus konténerek

- Típus nélküli konténerek

```
// Removes 4-letter words from c.
// Elements must be strings
static void expurgate(Collection c)
{
  for (Iterator i = c.iterator(); i.hasNext(); )
   if (((String) i.next()).length() == 4) i.remove
(); }
// ClassCastException possible!
```

Generikus konténerek 2

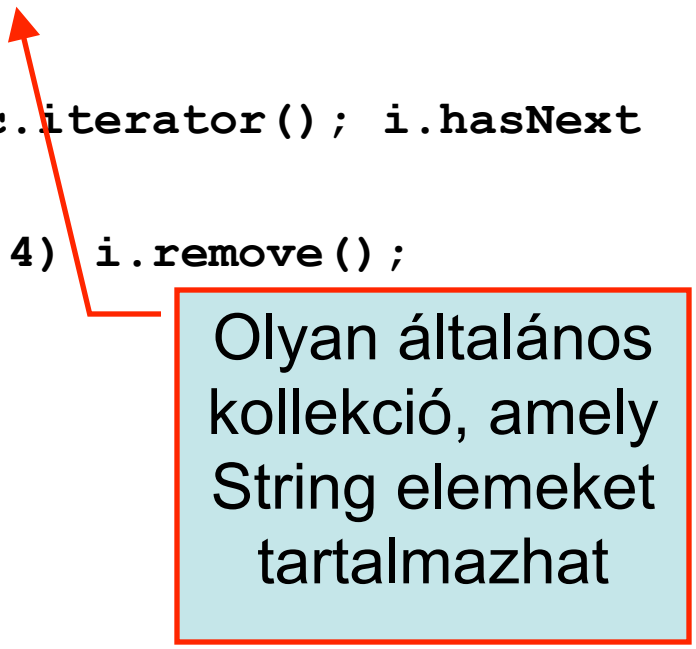
- „Típusos” konténerek

```
// Removes the 4-letter words from c
static void expurgate(Collection<String> c)
{
  for (Iterator<String> i = c.iterator(); i.hasNext
();  )
    if (i.next().length() == 4) i.remove();
}
```

Generikus konténerek 2

- „Típusos” konténerek

```
// Removes the 4-letter words from c
static void expurgate(Collection<String> c)
{
  for (Iterator<String> i = c.iterator(); i.hasNext
();  )
    if (i.next().length() == 4) i.remove();
}
```



Generikus konténerek 2

- „Típusos” konténerek

```
// Removes the 4-letter words from c
static void expurgate(Collection<String> c)
{
  for (Iterator<String> i = c.iterator(); i.hasNext
();  )
    if (i.next().length() == 4) i.remove();
}
```

Olyan iterátor, amelynek **`.next()`** metódusa String elemeket ad vissza

Olyan általános kollekció, amely String elemeket tartalmazhat

Generikus konténerek 3

- Miért jó mindez?
  - Kevesebb cast, kevesebb zárójel = áttekinthetőbb kód
  - Nagyobb biztonság: statikus típushelyesség vizsgálat (nincs ClassCastException)

Generikus konténerek 4

- Hogyan készíthetünk ilyeneket?

```
public interface List<E>
{
  void add(E x);
  Iterator<E> iterator();
}

public interface Iterator<E>
{
  E next();
  boolean hasNext();
}
```

Generikus konténerek 4

- Hogyan készíthetünk ilyeneket?

Fontos megjegyzés!

```
public interface Iterator<E>
{
  E next();
  boolean hasNext();
}
```

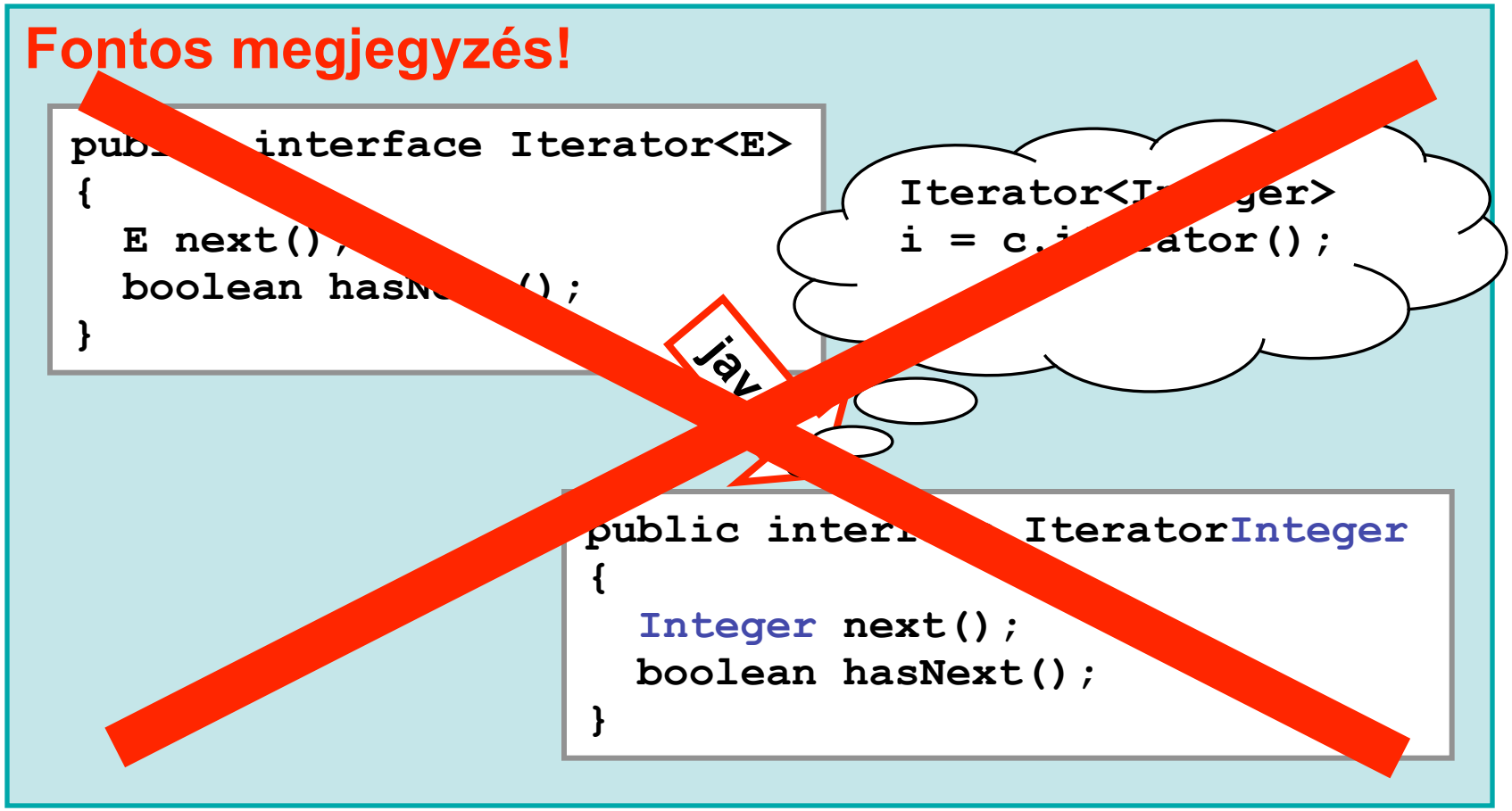
```
Iterator<Integer>
i = c.iterator();
```

javac

```
public interface IteratorInteger
{
  Integer next();
  boolean hasNext();
}
```

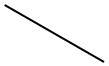
Generikus konténerek 4

- Hogyan készíthetünk ilyeneket?



Generikus konténerek 5

Generikus konténerek 5



```
List<String> ls = new ArrayList<String>();
List<Object> lo = ls;
// Compile error!
```

Generikus konténerek 5

- További „apróságok”

```
List<String> ls = new ArrayList<String>();
List<Object> lo = ls;
// Compile error!
```

Generikus konténerek 5

- További „apróságok”
  - Lehet ilyet?

```
List<String> ls = new ArrayList<String>();
List<Object> lo = ls;
// Compile error!
```

Generikus konténerek 5

- További „apróságok”
  - Lehet ilyet?

```
List<String> ls = new ArrayList<String>();
List<Object> lo = ls;
// Compile error!
```

Generikus konténerek 5

- További „apróságok”
  - Lehet ilyet?

```
List<String> ls = new ArrayList<String>();
List<Object> lo = ls;
// Compile error!
```

Generikus konténerek 5

- További „apróságok”
  - Lehet ilyet?

```
List<String> ls = new ArrayList<String>();
List<Object> lo = ls;
// Compile error!
```

  - Miért?

Generikus konténerek 5

- További „apróságok”
  - Lehet ilyet?

```
List<String> ls = new ArrayList<String>();
List<Object> lo = ls;
// Compile error!
```

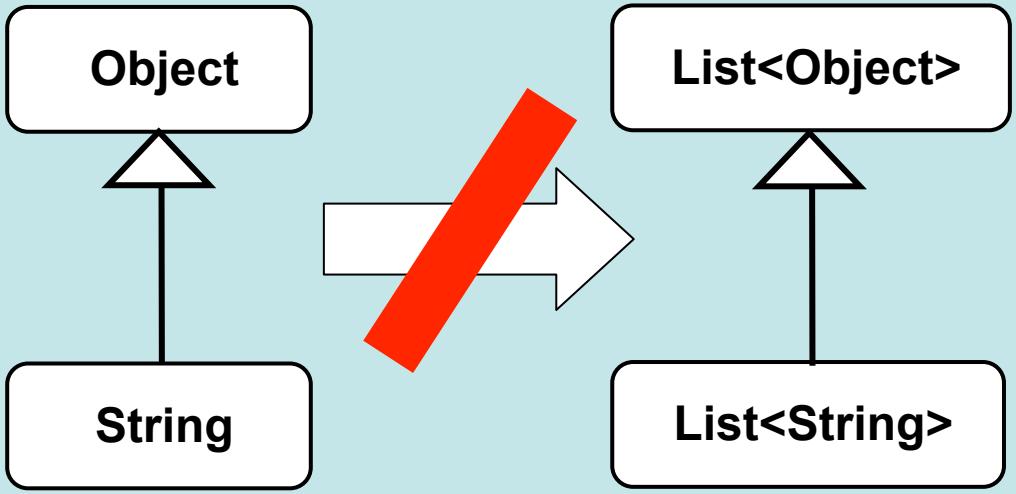
  - Miért?

```
lo.add(new Object());
String s = ls.get(0);
// attempts to assign an Object to a
// String!
```

Generikus konténerek 5

- További „apróságok”

Általánosságban:



Generikus konténerek 6

- Wildcards

```
// Generify!
void printCollection(Collection c)
{
  Iterator i = c.iterator();
  for (k = 0; k < c.size(); k++)
  {
    System.out.println(i.next());
  }
}
```

Generikus konténerek 6

- Wildcards

```
// Generified
void printCollection(Collection<?> c)
{
  for (Object e : c)
  {
    System.out.println(e);
  }
}
}
```

Generikus konténerek 7

- Bounded wildcards

```
// Generified and bounded
void printCollection(Collection<? extends Base> c)
{
  for (Base b : c)
  {
    System.out.println(b.someMethod());
  }
}
// Base: upper bound
```

Generikus konténerek 7

- Bounded wildcards

```
interface Comparator<T>
{
  int compare(T fst, T snd);
}

interface TreeSet<E>

{

  setComparator(Comparator<? super E> c);

}
// E: lower bound
```

Generikus konténerek 8

Generikus konténerek 8

```
static void fromArrayToCollection(Object[] a,
Collection<?> c)
{
  for (Object o : a) { c.add(o); }
}
// compile error
```

Generikus konténerek 8

- Generikus metódusok

```
static void fromArrayToCollection(Object[] a,
Collection<?> c)
{
  for (Object o : a) { c.add(o); }
}
// compile error
```

Generikus konténerek 8

- Generikus metódusok
  - Lehet ilyet?

```
static void fromArrayToCollection(Object[] a,
Collection<?> c)
{
  for (Object o : a) { c.add(o); }
}
// compile error
```

Generikus konténerek 8

- Generikus metódusok
  - Lehet ilyet?

```
static void fromArrayToCollection(Object[] a,
Collection<?> c)
{
  for (Object o : a) { c.add(o); }
}
// compile error
```

Generikus konténerek 8

- Generikus metódusok
  - Lehet ilyet?

```
static void fromArrayToCollection(Object[] a,
Collection<?> c)
{
  for (Object o : a) { c.add(o); }
}
// compile error
```

Generikus konténerek 8

- Generikus metódusok
  - Lehet ilyet?

```
static void fromArrayToCollection(Object[] a,
Collection<?> c)
{
  for (Object o : a) { c.add(o); }
}
// compile error
```

Generikus konténerek 8

- Generikus metódusok
  - Lehet ilyet?

```
static void fromArrayToCollection(Object[] a,
Collection<?> c)
{
  for (Object o : a) { c.add(o); }
}
// compile error
```

Generikus konténerek 8

- Generikus metódusok
  - Lehet ilyet?

```
static void fromArrayToCollection(Object[] a,
Collection<?> c)
{
  for (Object o : a) { c.add(o); }
}
// compile error
```

  - Helyette:

Generikus konténerek 8

- Generikus metódusok
  - Lehet ilyet?

```
static void fromArrayToCollection(Object[] a,
Collection<?> c)
{
  for (Object o : a) { c.add(o); }
}
// compile error
```

  - Helyette:

```
static <T> void fromArrayToCollection(T[] a,
Collection<T> c)
{
  for (T o : a) { c.add(o); }
}
```

Generikus konténerek 9

- Generikus kód és régi kód keveredik…

```
public class Inventory
{
  public static void addAssembly(String name,
                                 Collection parts) {...}

  public static Assembly getAssembly(String name) {...}

}
public interface Assembly
{
  Collection getParts();
}
```

Generikus konténerek 9

- Generikus kód és régi kód keveredik…

```
public class Inventory
{
  publi
  publi
}
public
{
  Colle
}
```

```
public class Main
{
  public static void main(String[] args)
  {
    Collection<Part> c = new ArrayList<Part>();
    c.add(new Guillotine()); // implements Part
    c.add(new Blade()); // implements Part
    Inventory.addAssembly("thingee", c); // ?! (1)
    Collection<Part> k =
     Inventory.getAssembly("thingee").getParts();
      // ?! (2)
  }
}
```

Generikus konténerek 9

- Generikus kód és régi kód keveredik…

```
public class Inventory
{
  public static void addAssembly(String name,
                                 Collection parts) {...}

  public static Assembly getAssembly(String name) {...}

}
public interface Assembly
{
  Collection getParts();
}
```

Generikus konténerek 9

- Generikus kód és régi kód keveredik…

```
public class Inventory
{
  public static void addAssembly(String name,
                                 Collection parts) {...}

  public static Assembly getAssembly(String name) {...}

}
public interface Assembly
{
  Collection getParts();
}
```

Raw type

Generikus konténerek 9

- Generikus kód és régi kód keveredik…

```
public class Inventory
{
  public static void addAssembly(String name,
                                 Collection parts) {...}

  public static Assembly getAssembly(String name) {...}

}
public interface Assembly
{
  Collection getParts();
}
```

Raw type

Unchecked warning

Generikus konténerek 10

Generikus konténerek 10

```
public String loophole(Integer x)
{
  List<String> ys = new LinkedList<String>();
  List xs = ys;
  xs.add(x); // (1)
  return ys.iterator().next(); // (2)
}
// unchecked warning (1)
// … and ClassCastException!
// but on which call?
// (2)!
```

Generikus konténerek 10

- Egy utolsó apróság…

```
public String loophole(Integer x)
{
  List<String> ys = new LinkedList<String>();
  List xs = ys;
  xs.add(x); // (1)
  return ys.iterator().next(); // (2)
}
// unchecked warning (1)
// … and ClassCastException!
// but on which call?
// (2)!
```

Generikus konténerek 10

- Egy utolsó apróság…
  - Lehet ilyet?

```
public String loophole(Integer x)
{
  List<String> ys = new LinkedList<String>();
  List xs = ys;
  xs.add(x); // (1)
  return ys.iterator().next(); // (2)
}
// unchecked warning (1)
// … and ClassCastException!
// but on which call?
// (2)!
```

Generikus konténerek 10

- Egy utolsó apróság…
  - Lehet ilyet?

```
public String loophole(Integer x)
{
  List<String> ys = new LinkedList<String>();
  List xs = ys;
  xs.add(x); // (1)
  return ys.iterator().next(); // (2)
}
// unchecked warning (1)
// … and ClassCastException!
// but on which call?
// (2)!
```

Generikus konténerek 10

- Egy utolsó apróság…
  - Lehet ilyet?

```
public String loophole(Integer x)
{
  List<String> ys = new LinkedList<String>();
  List xs = ys;
  xs.add(x); // (1)
  return ys.iterator().next(); // (2)
}
// unchecked warning (1)
// … and ClassCastException!
// but on which call?
// (2)!
```

Generikus konténerek 10

- Egy utolsó apróság…
  - Lehet ilyet?

```
public String loophole(Integer x)
{
  List<String> ys = new LinkedList<String>();
  List xs = ys;
  xs.add(x); // (1)
  return ys.iterator().next(); // (2)
}
// unchecked warning (1)
// … and ClassCastException!
// but on which call?
// (2)!
```

Generikus konténerek 10

- Egy utolsó apróság…
  - Lehet ilyet?

```
public String loophole(Integer x)
{
  List<String> ys = new LinkedList<String>();
  List xs = ys;
  xs.add(x); // (1)
  return ys.iterator().next(); // (2)
}
// unchecked warning (1)
// … and ClassCastException!
// but on which call?
// (2)!
```

Generikus konténerek 10

- Egy utolsó apróság…
  - Lehet ilyet?

```
public String loophole(Integer x)
{
  List<String> ys = new LinkedList<String>();
  List xs = ys;
  xs.add(x); // (1)
  return ys.iterator().next(); // (2)
}
// unchecked warning (1)
// … and ClassCastException!
// but on which call?
// (2)!
```

Generikus konténerek 10

- Egy utolsó apróság…
  - Lehet ilyet?

```
public String loophole(Integer x)
{
  List<String> ys = new LinkedList<String>();
  List xs = ys;
  xs.add(x); // (1)
  return ys.iterator().next(); // (2)
}
// unchecked warning (1)
// … and ClassCastException!
// but on which call?
// (2)!
```

Generikus konténerek 10

- Egy utolsó apróság…
  - Lehet ilyet?

```
public String loophole(Integer x)
{
  List<String> ys = new LinkedList<String>();
  List xs = ys;
  xs.add(x); // (1)
  return ys.iterator().next(); // (2)
}
// unchecked warning (1)
// … and ClassCastException!
// but on which call?
// (2)!
```

Generikus konténerek 10

- Egy utolsó apróság…
  - Lehet ilyet?

```
public String loophole(Integer x)
{
  List<String> ys = new LinkedList<String>();
  List xs = ys;
  xs.add(x); // (1)
  return ys.iterator().next(); // (2)
}
// unchecked warning (1)
// … and ClassCastException!
// but on which call?
// (2)!
```

Generikus konténerek 10

- Egy utolsó apróság…
  - Lehet ilyet?

```
public String loophole(Integer x)
{
  List<String> ys = new LinkedList<String>();
  List xs = ys;
  xs.add(x); // (1)
  return ys.iterator().next(); // (2)
}
// unchecked warning (1)
// … and ClassCastException!
// but on which call?
// (2)!
```

Generikus konténerek 10

- Egy utolsó apróság…
  - Lehet ilyet?

```
public String loophole(Integer x)
{
  List<String> ys = new LinkedList<String>();
  List xs = ys;
  xs.add(x); // (1)
  return ys.iterator().next(); // (2)
}
// unchecked warning (1)
// … and ClassCastException!
// but on which call?
// (2)!
```

  - Miért?

Generikus konténerek 10

- Egy utolsó apróság…
  - Lehet ilyet?

```
public String loophole(Integer x)
{
  List<String> ys = new LinkedList<String>();
  List xs = ys;
  xs.add(x); // (1)
  return ys.iterator().next(); // (2)
}
// unchecked warning (1)
// … and ClassCastException!
// but on which call?
// (2)!
```

  - Miért?
    - **Erasure:** típusinformáció *törlődik* fordítás után, és castok jönnek be a megfelelő helyekre

Generikus konténerek 11

- Egy hasznos trükk
  - Mi történik akkor, ha „öröklött” kód raw type-ként használja a típusos kollekciónkat?
  - Avagy hogyan kerülhető ki az előző „késői” ClassCastException?
- Így:

Generikus konténerek 11

- Egy hasznos trükk
  - Mi történik akkor, ha „öröklött” kód raw type-ként használja a típusos kollekciónkat?
  - Avagy hogyan kerülhető ki az előző „késői” ClassCastException?
- Így:

```
public String loophole(Integer x)
{
  List<String> ys = Collections.checkedList(
            new LinkedList<String>(),
            String.class);
  List xs = ys;
  xs.add(x); // ClassCastException
  return ys.iterator().next();
}
```

Egyéb generikus nyelvi lehetőségek

Egyéb generikus nyelvi lehetőségek

```
ThreadLocal<Context> threadLocal =
    new ThreadLocal<Context>();

void doSomethingInContext(Context c) {
    Context previous = threadLocal.get();
    threadLocal.set(c);
    try { doSomething(); }
    finally { threadLocal.set(previous); }
  }
```

Egyéb generikus nyelvi lehetőségek

Egyéb generikus nyelvi lehetőségek

- „csomagoló” osztályok (**`ThreadLocal`**)
- java.lang.Class type tokens
  - **`String.class → Class<String>`**
  - Például:

Egyéb generikus nyelvi lehetőségek

- „csomagoló” osztályok (**`ThreadLocal`**)
- java.lang.Class type tokens
  - **`String.class → Class<String>`**
  - Például:

```
public static <T> Collection<T> select
(Class<T>c, String sqlStatement)
{
 Collection<T> result = new ArrayList<T>();
 /* run sql query using jdbc */
 for ( /* iterate over jdbc results */ )
 {
   T item = c.newInstance();
   /* use reflection and set all of item's
    fields from sql results */
   result.add(item);
 }
 return result;
}
```

Foreach

```
void cancelAll(Collection<TimerTask> c)
{
  Iterator<TimerTask> i = c.iterator();
  while (i.hasNext())
  {
    i.next().cancel();
  }
}
```

Foreach

- A régi módszer

```
void cancelAll(Collection<TimerTask> c)
{
  Iterator<TimerTask> i = c.iterator();
  while (i.hasNext())
  {
    i.next().cancel();
  }
}
```

Foreach

- A régi módszer

```
void cancelAll(Collection<TimerTask> c)
{
  Iterator<TimerTask> i = c.iterator();
  while (i.hasNext())
  {
    i.next().cancel();
  }
}
```

Foreach

- A régi módszer

```
void cancelAll(Collection<TimerTask> c)
{
  Iterator<TimerTask> i = c.iterator();
  while (i.hasNext())
  {
    i.next().cancel();
  }
}
```

Foreach

- A régi módszer

```
void cancelAll(Collection<TimerTask> c)
{
  Iterator<TimerTask> i = c.iterator();
  while (i.hasNext())
  {
    i.next().cancel();
  }
}
```

Foreach

- A régi módszer

```
void cancelAll(Collection<TimerTask> c)
{
  Iterator<TimerTask> i = c.iterator();
  while (i.hasNext())
  {
    i.next().cancel();
  }
}
```

Foreach

- A régi módszer

```
void cancelAll(Collection<TimerTask> c)
{
  Iterator<TimerTask> i = c.iterator();
  while (i.hasNext())
  {
    i.next().cancel();
  }
}
```

- Ehelyett

```
void cancelAll(Collection<TimerTask> c)
{
  for (TimerTask t : c)
        t.cancel();
}
```

Foreach 2

- Beágyazott ciklusok…

- Helyesen:

```
for (Suit s: suits)
  for (Rank r: ranks)
    sortedDeck.add(new Card(s, r));
```

Foreach 2

- Beágyazott ciklusok…

```
for (Iterator i = suits.iterator(); i.hasNext(); )
  for (Iterator j = ranks.iterator(); j.hasNext(); )
    sortedDeck.add(new Card(i.next(), j.next()));
// NoSuchElementException
```

- Helyesen:

```
for (Suit s: suits)
  for (Rank r: ranks)
    sortedDeck.add(new Card(s, r));
```

Foreach 3

Foreach 3

```
int sum(int[] a)
{
 int result = 0;
 for (int i : a)
  result += i;
 return result;
}
```

Foreach 3

- Tömbökkel is működik
- Mikor **nem** használhatjuk?
  - Filterezés
  - Több kollekció párhuzamos olvasása

```
int sum(int[] a)
{
 int result = 0;
 for (int i : a)
  result += i;
 return result;
}
```

Dobozolás (Autoboxing)

```
Set s = new HashSet();
s.add(1); s.add(true);
```

Dobozolás (Autoboxing)

- Lehet ilyet?
- Most már lehet.
  - Dobozolás: primitív típusok és objektum megfelelőik közötti automatikus oda-vissza konverzió
- Mire jó?
  - „felesleges” kódtól szabadít meg
  - De milyen áron?

```
Set s = new HashSet();
s.add(1); s.add(true);
```

Dobozolás 2

- Óvatosan használjuk:
  - Integer, Boolean lehet **`null`** → kidobozolásnál jöhet **`NullPointerException`**
  - **`==`** jelentése:
    - Érték szerinti összehasonlítás primitívekre
    - Referencia egyenlőség szerinti összehasonlítás objektumokra
- Mikor **ne** használjuk?
  - Ha sebesség kell.

Enumerációk

```
public static final int ENUM_LIT1 = 0;
public static final int ENUM_LIT2 = 1;
public static final int ENUM_LIT3 = 2;
```

Enumerációk

- Enumeráció „szerűség” Java-ban eddig:

```
public static final int ENUM_LIT1 = 0;
public static final int ENUM_LIT2 = 1;
public static final int ENUM_LIT3 = 2;
```

- Mi ezzel a baj?
  - Nem biztonságos (**`int`**)
  - Nem illeszkedik a Java névtérbe (**`ENUM_`**)
  - Körülményes a módosítás (próbáljunk újat felvenni két érték közé…)
  - Debug célú **`print()`** eredménye egy szám.

Enumerációk 2

- Mostantól:

```
enum Season { WINTER, SPRING, SUMMER, FALL }
```

- Tudja a „hagyományos” enum funkciókat
  - **`Switch()`**-be tehető
- Mennyivel több ez C(++ / #) enumnál?
  - Ez egy valódi osztály! (*enum type*)
  - Lehetnek metódusai, tagváltozói
  - Megvalósíthat interfészeket, öröklődhet
  - Értelmes **`.toString()`**, stb. megvalósításokat tartalmaz
  - **`Comparable, Serializable`**
  - Statikus **`.values()`** metódus az összes érték lekéréséhez

Enumerációk 3

- További trükkök…

```
public enum Operation
{
  PLUS { double eval(double x, double y) { return x + y; } },
  MINUS { double eval(double x, double y) { return x - y; } },
  TIMES { double eval(double x, double y) { return x * y; } },
  DIVIDE { double eval(double x, double y) { return x / y; } };

  // Do arithmetic op represented by this constant
  abstract double eval(double x, double y);
} // constant-specific methods
```

Enumerációk 4

- További trükkök…

```
for (Day d : EnumSet.range(Day.MONDAY, Day.FRIDAY))
        System.out.println(d);
// use java.util.EnumSet for fast set operations on enums,
// including:.allOf, .noneOf, .complementOf

// use .of instead of traditional bit flags:
EnumSet.of(Style.BOLD, Style.ITALIC);



// use java.util.EnumMap for fast enum-keyed maps
private static Map<Suit, Map<Rank, Card>> table =
    new EnumMap<Suit, Map<Rank, Card>>(Suit.class);
```

Változó argumentumszámú metódusok

Változó argumentumszámú metódusok

```
String format(String pattern, Object... arguments);
// use varargs only in the final argument!
// treat arguments as an Object[]
```

Változó argumentumszámú metódusok

- **`System.out.printf("%d %d", 1,2);`**

```
String format(String pattern, Object... arguments);
// use varargs only in the final argument!
// treat arguments as an Object[]
```

Változó argumentumszámú metódusok

- **`System.out.printf("%d %d", 1,2);`**
- Hogyan kell ilyet deklarálni?

```
String format(String pattern, Object... arguments);
// use varargs only in the final argument!
// treat arguments as an Object[]
```

Változó argumentumszámú metódusok

- **`System.out.printf("%d %d", 1,2);`**
- Hogyan kell ilyet deklarálni?

```
String format(String pattern, Object... arguments);
// use varargs only in the final argument!
// treat arguments as an Object[]
```

Változó argumentumszámú metódusok

- **`System.out.printf("%d %d", 1,2);`**
- Hogyan kell ilyet deklarálni?

```
String format(String pattern, Object... arguments);
// use varargs only in the final argument!
// treat arguments as an Object[]
```

Változó argumentumszámú metódusok

- **`System.out.printf("%d %d", 1,2);`**
- Hogyan kell ilyet deklarálni?

```
String format(String pattern, Object... arguments);
// use varargs only in the final argument!
// treat arguments as an Object[]
```

- És használni?

Változó argumentumszámú metódusok

- **`System.out.printf("%d %d", 1,2);`**
- Hogyan kell ilyet deklarálni?

```
String format(String pattern, Object... arguments);
// use varargs only in the final argument!
// treat arguments as an Object[]
```

- És használni?

```
format("%d %d",1,2);
format("%d %d",new Object[]{1,2});
// pass arguments as an Object[] OR
// as a sequence of arguments
```

Annotációk (metaadatok)

- Annotáció = a kód egy meghatározott részletéhez „ragasztott” címke
- Mire jó?
  - Dokumentáció
  - **Származtatott (generált) kód (fájlok) előállításának vezérlésére – segítésére**
    - Jó példa: (web) deployment descriptor

Annotációk 2

Annotációk 2

- Nincs már erre eszköz? (JavaDoc?)
- Dehogynem: (JavaDoc ☺)
  - Az annotációk először itt jelentek meg

Annotációk 2

- Nincs már erre eszköz? (JavaDoc?)
- Dehogynem: (JavaDoc ☺)
  - Az annotációk először itt jelentek meg
    - @author, @deprecated, @return, @param, @throws, @see, …
  - Mi ezzel a „baj”?

Annotációk 2

- Nincs már erre eszköz? (JavaDoc?)
- Dehogynem: (JavaDoc ☺)
  - Az annotációk először itt jelentek meg
    - @author, @deprecated, @return, @param, @throws, @see, …
  - Mi ezzel a „baj”?
    - *Komment*ben vannak → elsősorban dokumentációs célra alkalmas
- Miben tudnak a Java 5 annotációk többet?
  - Részei nemcsak a forrásnak, hanem a class fájloknak is
  - Reflectionnel kiolvashatók
  - speciális nyelvi elemek, bővíthető APIval

Annotációk 2

- Nincs már erre eszköz? (JavaDoc?)
- Dehogynem: (JavaDoc ☺)
  - Az annotációk először itt jelentek meg
    - @author, @deprecated, @return, @param, @throws, @see, …
  - Mi ezzel a „baj”?
    - *Komment*ben vannak → elsősorban dokumentációs célra alkalmas
- Miben tudnak a Java 5 annotációk többet?
  - Részei nemcsak a forrásnak, hanem a class fájloknak is
  - Reflectionnel kiolvashatók
  - speciális nyelvi elemek, bővíthető APIval
  - **`apt`**: Annotation Processing Tool

Annotációk 2

- Nincs már erre eszköz? (JavaDoc?)
- Dehogynem: (JavaDoc ☺)
  - Az annotációk először itt jelentek meg
    - @author, @deprecated, @return, @param, @throws, @see, …
  - Mi ezzel a „baj”?
    - *Komment*ben vannak → elsősorban dokumentációs célra alkalmas
- **Fontos!**

  Az annotációk NEM közvetlenül módosítják a program viselkedését, mint a többi nyelvi elem.

  Sokkal inkább a programot feldolgozó és futtató környezet viselkedését befolyásolhatják.

Annotációk 3

- Hogyan deklaráljunk egy címkét?

```
/**
* Describes the Request-For-Enhancement(RFE) that led
* to the presence of the annotated API element.
*/
public @interface RequestForEnhancement
{
  int id();
  String synopsis();
  String engineer() default "[unassigned]";
  String date() default "[unimplemented]";
}
```

Annotációk 3

- Hogyan deklaráljunk egy címkét?

```
/**
* Describes the Request-For-Enhancement(RFE) that led
* to the presence of the annotated API element.
*/
public @interface RequestFo
{
   int id();
   String synopsis();
   String engineer() default
   String date() default "[t
}
```

Minden metódus a címke egy-egy *elem*ét jelöli.

Nem szabad paramétereket használni, illetve `throws` klózt.

Lehetséges visszatérési értékek: primitívek, String, Class, enumerációk, *annotációk*, és ezek tömbjei.

Annotációk 4

- Hogyan használjunk egy címkét?

```
@RequestForEnhancement(
  id = 2868724, synopsis = "Enable time-travel",
  engineer = "Mr. Peabody", date = "4/1/3007"
)
public static void travelThroughTime(Date destination)
{ ... }
@Preliminary // marker annotation, no elements
public class TimeTravel{ ... }
// annotation with a single element
public @interface Copyright
{
String value(); // should be named value
}
@Copyright("2002 Yoyodyne Propulsion Systems")
public class OscillationOverthruster { ... }
```

Annotációk 5

- Meta-annotációk
  - az annotációk annotációi

```
import java.lang.annotation.*;

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Test { }
```

Annotációk 5

- Meta-annotációk
  - az annotációk annotációi

```
import java.lang.annotation.*;

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Test { }
```

Jelzi, hogy ezt az annotációt meg kell tartani futási időben is, hogy reflektíven ki tudjuk olvasni.

Annotációk 5

- Meta-annotációk
  - az annotációk annotációi

```
import java.lang.annotation.*;

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Test { }
```

Jelzi, hogy ezt az annotációt meg kell tartani futási időben is, hogy reflektíven ki tudjuk olvasni.

Jelzi, hogy ezt az annotációt csak metódusok címkézésére használhatjuk.

Annotációk 6

- Teszt keretrendszer

```
import java.lang.annotation.*;
import java.lang.reflect.*;
public class RunTests
{
 public static void main(String[] args) throws Exception
 {
  for (Method m : Class.forName(args[0]).getMethods())
  {
   if (m.isAnnotationPresent(Test.class))
   {
    try { m.invoke(null); }
    catch (Throwable ex)
    {
     System.out.printf("%s failed: %s %n", m, ex.getCause());
    }
}}}}
```

Annotációk 6

- Teszt keretrendszer

Végigmegyünk a tesztosztályból példányosított objektum metódusain, reflexióval.

```
import java.lang.annotation.*;
import java.lang.reflect.*;
public class RunTests
{
 public static void main(String[] args) throws Exception
 {
  for (Method m : Class.forName(args[0]).getMethods())
  {
   if (m.isAnnotationPresent(Test.class))
   {
    try { m.invoke(null); }
    catch (Throwable ex)
    {
     System.out.printf("%s failed: %s %n", m, ex.getCause());
    }
}}}}
```

Annotációk 6

- Teszt keretrendszer

```
import java.lang.annotation.*;
import java.lang.reflect.*;
public class RunTests
{
 public static void main(String[] args) throws Exception
 {
  for (Method m : Class.forName(args[0]).getMethods())
  {
   if (m.isAnnotationPresent(Test.class))
   {
    try { m.invoke(null); }
    catch (Throwable ex)
    {
     System.out.printf("%s failed: %s %n
    }
}}}}
```

Végigmegyünk a tesztosztályból példányosított objektum metódusain, reflexióval.

Ha @Test címkével jelzett metódust találunk, meghívjuk.

Annotációk 7

- Annotation Processing Tool (**`apt`**)
  - Célja: automatizáljuk a címkézett programok fordítását
  - Fordítási időben készít egy read-only nézetet a programunkról,
  - Amit saját osztályokkal feldolgozhatunk (további kód generálása, egyéb fájlok létrehozása).

Annotációk 8

- Annotation Processing Tool (**`apt`**)
  - Read-only nézet:
    - Java programok absztrakciója, a Mirror API alapján (**`com.sun.mirror.declaration`**)
  - Ezt annotation processorok dolgozzák fel (**`com.sun.mirror.apt`**), a *visitor* minta alapján
  - Ha új forráskód jött létre, rekurzívan folytatódik a feldolgozás

Kitekintés: Java 6

- Collections keretrendszer új elemei
  - **`Deque`** (két irányból módosítható várakozási sor)
  - **`NavigableSet`** (megkereshetjük a keresési kulcshoz adott szempontból legközelebbi találatot)
  - **`NavigableMap`** (hasonlóan)

Kitekintés 2

- IO
  - **`java.io.Console (.printf (), .readLine(), .readPassword() )`**
  - **`System.console()`**
- Networking
  - Beépített pehelysúlyú HTTP(S) szerver
    - Beágyazott webserverek készítéséhez (~servlet szerű absztrakció)

Kitekintés 3

- Scripting
  - Egyszerű script fordítókat ültethetünk rá a Java alkalmazásunkra
  - Keverhető a Java és JavaScript forráskód
- JConsole
  - A JMX API-n keresztül monitorozhatjuk a VM-einkben futó Java alkalmazásokat
- …