A programozás alapjai 3.

Java bevezetés

Ez az oktatási segédanyag a Budapesti Műszaki és Gazdaságtudományi Egyetem oktatója által kidolgozott szerzői mű. Kifejezett felhasználási engedély nélküli felhasználása szerzői jogi jogsértésnek minősül.

Goldschmidt Balázs
balage@iit.bme.hu

1

Programozás alapjai

- 1: *Strukturált programozás*
  - Változók, vezérlési szerkezetek, függvények, adatstruktrúrák stb.
  - Nyelv: *C vagy Python*
- 2: *OO fogalmak*
  - Osztályok, egységbezárás, öröklés, polimorfizmus stb.
  - Nyelv: *C++*
- 3: *OO fejlesztés API-k használatával*
  - IO, kollekciók, többszálúság, GUI, egység-tesztelés stb.
  - Nyelv: *Java*

2

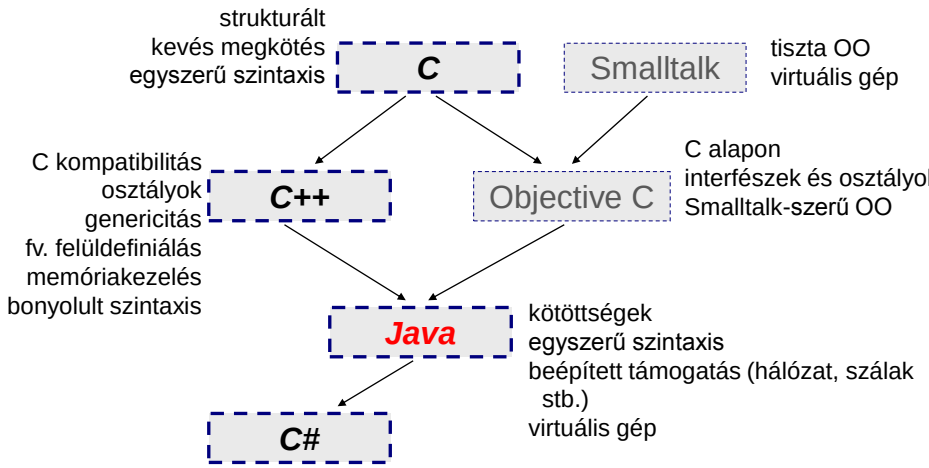
TIOBE Index (népszerűség, 12 havi átlag)

© 2019 TIOBE software BV

Programming Language	2019	2014	2009	2004	1999	1994
Java	1	2	1	1	14	-
C	2	1	2	2	1	1
Python	3	7	5	7	24	21
C++	4	4	3	3	2	2
Visual Basic .NET	5	9	-	-	-	-
C#	6	5	6	6	19	-
JavaScript	7	8	8	8	16	-
PHP	8	6	4	5	-	-
SQL	9	-	-	89	-	-
Objective-C	10	3	31	38	-	-

3

A Java családfája

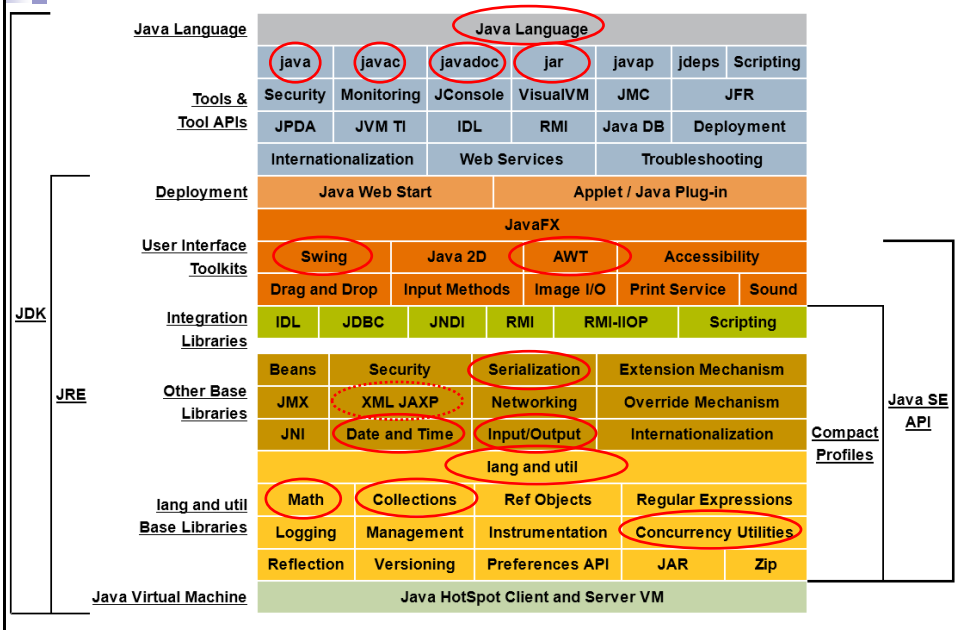


4

J2SE keretrendszer

- Java a C-hez hasonló
  - egyszerű szintaxis
  - hatalmas API támogatás
- Java programozás: legózás
  - előre elkészített építőkockákból dolgozunk
  - minden megoldás megtalálható
    - általában jobb minőségben, mintha mi csináltuk volna
  - az API ismerete a valódi Java tudás
  - a Java verziók API-ban és szintaxisban eltérhetnek
    - utolsó főverziószám: 14 (2020-03-10)

5



6

Java alapok

- (Majdnem) minden osztály vagy objektum
  - nincsenek globális függvények
  - alkalmazás struktúrája:
    - csomagok > osztályok > metódusok és változók > utasítások
- Kétfajta típus
  - primitív (int, double, boolean, …)
    - változók értéket tárolnak
  - objektum (String, Vector, …)
    - változók referenciát (memóriacímet) tárolnak

Java alapok 2

- Szintaxis C/C++-hoz nagyon hasonló
  - operátorok (+,-, >>, …)
  - vezérlési szerkezetek (for, while, switch)
  - metódushívás
- Azonban
  - *nincsenek pointerek*
  - *nincs goto*
  - *nincs operator overloading*
  - *külön* byte, char, *és* boolean *típusok vannak*

Java alapok 3

- A tömbök is objektumok
  - hossz tárolva (length) → futásidejű ellenőrzés

```
int a[] = new int[10];
//int[] a = new int[10]; // ez is jó
for (int i = 0; i < a.length; i++) {
      a[i] = i*2;
}
```

- Csak érték szerinti paraméterátadás
  - nincs pointeraritmetika
- Garbage collection (szemétgyűjtés)
  - nincs delete

Hello world

```
// C/C++

int main(int argc, char** argv) {
    printf("Hello world\n");
}
```

```
// Java (Hello.java)

public class Hello {
    static public void main(String[] args) {
        System.out.println("Hello world");
    }
}
```

Fordítás és futtatás

- Ökölszabály:
  - minden osztály külön forrásfájlban
    - `class Hello` → *Hello.java*
  - minden osztályhoz külön bytecode (class) fájlt generál a fordító
  - `> javac Hello.java` → *Hello.class*
- A JVM (Java virtual machine) a paraméterként megadott osztály *main* metódusát indítja el
  - `> java Hello`

Write Once, Run Anywhere

- C, C++ stb.:
  - *write once, compile everywhere*
  - *egyszer megírjuk, mindenhol fordítjuk*
- Java:
  - forráskód bytecode-dá fordul
  - a bytecode-ot futtatja a virtuális gép
  - ezzel elkerüljük, hogy bárhol újra kelljen fordítani
- write once, debug everywhere
  - (egyszer megírjuk, mindenhol debuggoljuk)
  - a jó tervezés továbbra is fontos
  - könnyen maradhat a kódban platformfüggő rész ☹

Java alkalmazások futtatása

- Legegyszerűbb mód
  - parancssorból vagy batch-fájlból
- Jar file
  - speciális ZIP fájl, mindent tartalmaz
  - jó beállítások esetén klikkelésre elindul
- Alkalmazásszerver (Jakarta EE)
  - speciális futtatókörnyezet
  - nem önálló alkalmazásokhoz
  - sok gyári komponenssel és beállítással

Alaptípusok, operátorok, utasítások

Primitív típusok és változók

- Primitív típusok
  - boolean
  - char (16bit unicode)
  - byte, short, int, long (8, 16, 32, 64 bites előjeles egészek)
  - float, double (32 és 64 bit valósok)
- Változók deklarációja és definíciója
  - hasonlóan, mint C és C++

```
int a = 13;
double d = f = 3.14;
```


Összetett típusok

- Tömbök, Stringek stb. mind összetettek
- Változó csak referenciát tárol
  - mint C/C++ pointer
  - C/C++ *NULL* Javaban *null*
- Értékadás a referenciát módosítja
  - referált objektum nem változik

```
String s = "12345"; // String literál
s = "hello"; // korábbi referencia elveszik,
             // de az "12345" String a
             //memóriában marad
String q = null;
```

Tömbök

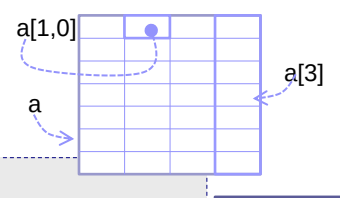
- Egyszerű tömb definiálása

```
int a[] = new int[13]; // 13 elemű, int-ek vannak benne
double[] d = new double[20]; // 20 db double
```

- Tömbök merevek
  - méretük konstans
    - létrehozáskor rögzül
    - *length* attribútummal lekérdezhető
  - indexelés 0-tól
  - alapértelmezetten nullára inicializálódnak az elemek
    - referencia esetén *null*

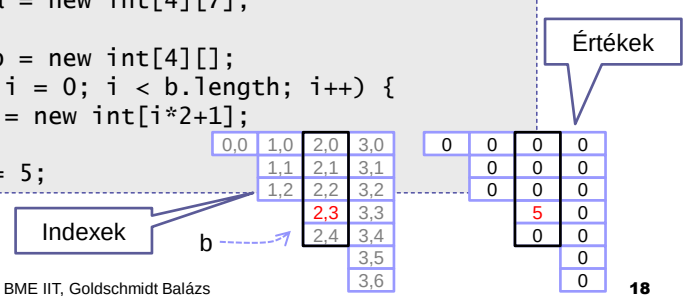
Többdimenziós tömbök

- Többdimenziós tömb
  - tömbök tömbje
  - mindig referencia a tömb is
  - eltérhet az altömbök mérete




```
int[][] a = new int[4][7];

int[][] b = new int[4][];
for (int i = 0; i < b.length; i++) {
    b[i] = new int[i*2+1];
}
b[2][3] = 5;
```



Operátorok

- Ugyanazok az operátorok, mint C/C++ esetén
  - ugyanazok a precedencia és asszociációs szabályok
  - logikai műveletek logikai értéket adnak
    - nincs keverve a logikai érték az aritmetikaival
- Hiányzó műveletek (Java-ban nincs)
  - **`delete`**, **`->`**
- Új vagy módosított műveletek
  - **`>>`** (előjel megmarad)
  - **`>>>`** (balról 0 kerül be)
  - nem-lusta kiértékeléshez: **`&`**, **`|`**, **`^`**

Utasítások

- Hasonlóan a C/C++-hoz
  - if-else, while, do-while, for, switch-case
    - **`if`**, **`while`**, **`for`** (2. kifejezésben) logikai értéket vár
    - (Java 7: **`case`** Stringeken is működik)
  - continue, break, return
    - első kettő címkével is használható

```
int i = 1;
loop: while (i < 100) {
    for (int k = i; k < 300; k++) {
        if (k == i*2) break loop;
    }
}
```

  - nincs ***`goto`***

Objektumok, osztályok és interfészek

Osztályok

- Emlékeztet a C++-ra
  - kisebb-nagyobb eltérésekkel
- Eltérések
  - láthatóság osztályoknál is (csomag szint)
  - láthatóság tagonként külön (metódus, attribútum)
  - attribútumok kezdőértékkel (0, null stb.)
  - csak *inline* metódusok
  - minden metódus virtuális
    - *private* metódusok kivételével
  - nincs operator felüldefiniálás

Osztályok 2

- Eltérések (folyt.)
  - nincs másoló és move konstruktor
    - referencia értéke adódik át (paraméter, return)
  - nincs inicializáló lista
  - nincsenek default paraméterek
  - nincs sem többszörös, sem virtuális öröklés
  - *this* konstruktorhívásként is használható konstr.-ban
  - nincs destruktor, helyette van *finalize()*
  - Java referencia a C++ pointer, nem a C++ referencia

Osztálypélda

```
public class Something {
   int a; // package láthatóság
   private double d;
   protected long l;
   public String s;

   public Something(int a) {
       this.a = a;
   }

   public Something() {
       this(10);
       l = 24l;
   }
   // ...
```

Osztálypélda folyt.

```
    // ...
    public void finalize() {
        ...
    }

    private void increment(int i) {
        a += i;
    }
    public long add(int i) {
        increment(i);
        l += i;
        return l;
    }
}
```


25

Osztálypélda folyt.

```
// valahol az osztályban....

public static void main(String[] args) {

    // zárójel kötelező konstruktorok esetén
    // s csak referenciát tárol
    // NINCS "*" operátor!
    Something s = new Something(5);

    // mezőkiválasztás "." operátorral
    // NINCS "->" operátor!
    long f = s.add(34);

}
```


26

Hozzáférési korlátozások

- private
  - mint a C++: csak ugyanabból az osztályból
- *package* (nincs jele, “default-access”)
  - nincs C++ megfelelője
  - hozzáférés ugyanabból a csomagból
- protected
  - mint a C++: leszármazottban vagy azonos csomagból
- public:
  - mint a C++: bárhonnan

További módosítók

- static
  - mint a C++: *osztályszintű metódus vagy attribútum*
- final
  - nincs a C++-ban:
    - metódusnál: leszármazott nem definiálhatja felül
    - változóknál: mint a C++ *const,* egyszer kaphat értéket
- abstract
  - csak metódusok és osztályok esetén
    - metódusok esetén, mint C++ pure virtual
      - nem absztrakt leszármazottban definiálva kell legyen
  - van abstract metódusa, osztály is abstract kell legyen
    - abstract osztály nem példányosítható

Statikus tagok

- Statikus tagok, mint C++ esetén
  - statikus metódus csak statikus tagokat ér el közvetlenül
  - statikus tag elérhető nem statikus metódusból is
- Statikus változók inicializálása

```
class A {
   static long l = 13; // inline
   static long k;
   static { // statikus inicializáló blokk
       k = 15; // osztály betöltésekor fut le
   }
}
```


String: egy különös osztály

- Szokásos string műveleteket nyújtja
  - `length()`, `equals()`, `startsWith()`
  - `substring()`, `trim()`, `split()`, `concat()`
  - `toUpperCase()`, `toLowerCase()`, `replace()`
  - `charAt()`, `indexOf()`, `lastIndexOf()`
  - `valueOf()`
  - ...
- Az egyetlen, akin működik a + és +=
  - összefűzés, nem hatékony
- Immutábilis *(immutable)*
  - objektum tartalma sosem változik

Öröklés

- Szintaxis eltér a C++-tól
  - *extends*

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

  - a *super()* metódussal hívhatjuk az ős konstruktorát
- Szemantika is eltér a C++-tól
  - minden metódus virtuális
  - nincs többszörös öröklés (max. egy közvetlen ős)
  - legfelső ősosztály: *Object*
  - konstruktorok inicializálása nem a C++-ban megszokott

Öröklés példa

```
class A {
   int k;
   public A() { k = 13; }
   public A(int i) { k = i; }
   public void foo() { System.out.println("A"); }
   public void bar() { foo(); }
}

class B extends A {
   public B() {}
   public B(int j) { super(j); }
   public void foo() { System.out.println("B"); }
}
```

Konstruktor feladatai

- Objektum szerekezetének felépítése
  - attribútumok 0-ra inicializálása
  - virtuális fv-tábla inicializálása
- Ősosztályt inicializáló műveletek
  - ...
- Osztály inicializálása
  - explicit attribútum-inicializálás
  - inicializáló blokk *(névtelen blokk)* futtatása
  - konstruktor törzsének futtatása

Konstruktor feladatai

```
class A {
   int k,l;
   { k = 20; } // inic. blokk
   public A() { l = 13; }
   public void foo() { System.out.println("A"); }
}

class B extends A {
   public B() {}
   public void foo() { System.out.println("B"); }
}
```

`Object` ősosztály

- Legfelső ősosztály
- Metódusai
  - `boolean equals(Object o)`
    - tartalom-alapú összehasonlítás *(alapból referenciával)*

      `a == b` vs. `a.equals(b)`
  - `int hashCode()`
    - hash érték számítása hatékony tároláshoz
  - `void finalize()`
    - hasonlít a C++ destruktorhoz, GC hívja meg
    - inkább ne használjuk

`Object` ősosztály 2

- Metódusok folyt.
  - `String toString()`
    - visszaadja az objektum String reprezentációját
    - elsősorban teszteléshez
    - ahol Stringet várunk, meghívódik

      `"my car: " + myCar + ";"`
  - `Object clone()`
    - visszaadja az objektum másolatát (részletek később)

Interfészek

- Implementáció nélküli osztályok
  - minden interfész saját forrásfájlba kerül
- Metódusok csak fejléccel
  - nincs implementáció (definíció)
  - implicit public hozzáférés
- Lehetnek attribútumaik (konstans értékekhez)

```
interface A {
   void foo();
   int bar(String s);
   public static final int maxLength = 100;
}
```


Interfészek 2

- Interfészek között lehet többszörös öröklés
  - csak ha az attribútumok és metódusok nem ütköznek
- Osztályok bárhány interfészt megvalósíthatnak
  - *implements* kulcsszó

```
class A extends B implements C, D {}
```

- Nem kell minden metódust megvalósítani
  - de akkor az osztály *abstract* kell legyen

Interfész példa

```
interface A {
   void foo();
   int bar(String s);
}

abstract class B implements A {
   ...
   public void foo() { System.out.println("B"); }
   abstract public int bar(String s);
}
class C implements A {
   ...
   public void foo() { System.out.println("B"); }
   public int bar(String s) { return s.length();}
}
```


Csomagok

Csomagok

- Hierarchikus névterek létrehozásához
  - hasonlóan a C++ *namespace*-ekhez
- A csomagok hierarchiája meg kell feleljen a mappahierarchiának
  - azonos névvel, azonos hierarchiában
- Osztályok és interfészek
  - forráskódban definiálni kell a befoglaló csomagot
    - `package foo.bar.baz;`
  - forrásfájlt a megfelelő mappába kell tenni
    - pl. `C:\Users\joe\javakodok\foo\bar\baz`

Csomagok és osztályok nevei

- Teljes név
  - `foo.bar.baz.MyClass`
- Nevek importálása
  - az előatagok elhagyásához
  - osztályok, interfészek, tagok
  - hasonlóan a *using namespace X* megoldáshoz
  - definiálja, hogy hol kell keresni az azonosítókat
  - ha ütközés lenne, a teljes nevet kell alkalmazni
    - pl. *List* része a *java.util* és a *java.awt* csomagnak is
  - *static import* használható tagok importálására

```
import foo.bar.baz.*;
import mypack.MyClass;
```

Modulok

- Csomagok csoportosítására
  - Java 9 óta
    - 95 modult vezettek be
  - pl. *java.base* modulban *java.lang*, *java.io*, *java.util* stb.
  - [modulok] > csomagok > osztályok > …
- Előnyei
  - erősebb kötődés az összetartozó csomagok között
  - tisztább függőségi viszonyok
    - egységbezárás, láthatóság
  - skálázhatóság és optimalizáció

Kivételkezelés

Kivételkezelés C++-ban

- Hagyományos kivételkezelés

```
try {
  ...
} catch (E e) { // E típusú kivétel
} catch (...) { // minden egyéb
  ...
  throw;
}
```

- Nincs előre definiált kivétel ősosztály
  - STL bevezette az *exception* osztályt

Kivételkezelés Java-ban

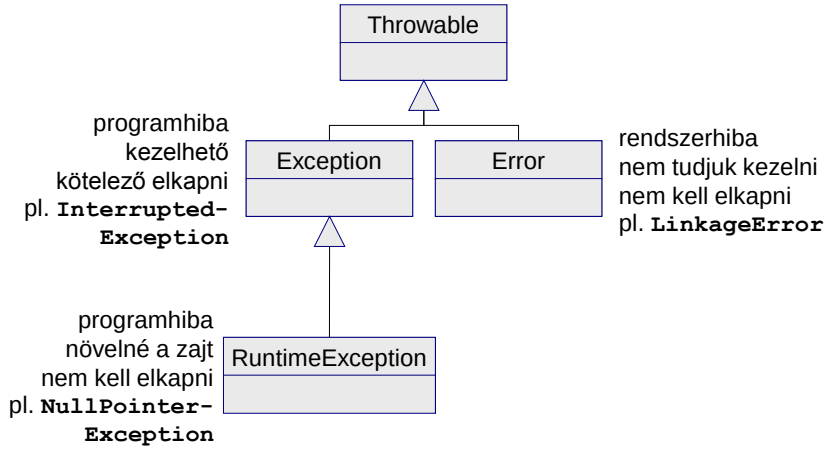
- C++ megoldásán alapul
- de létezik egy közös kivétel-ős
  - **Throwable**
- minden kivételt, ami dobódhat, meg kell adni
  - **throws**
- ősosztályok különféle kivételekhez
  - **Exception**, **RuntimeException**, **Error**
- záró blokk: **finally**
- kitekintés: *try with resources* (Java7)

Kivételkezelés **Java**-ban

```
void foo(String s) throws IOException {
        …
}
```

```
FileInputStream fis = new FileInputStream("a");
try {
        ...
        foo(s);
        ...
} catch (IOException e) {
        ...
        throw e;
} finally {
        fis.close();
}
```


Kivételek hierachiája



Kaszkád kivételek

```
public AST parseFile(File f) throws ParseException {
       AST ast = new AST();
       try {
              …
       } catch (IOException e) {
              ParseException pe = new ParseException();
              pe.initCause(e);
              throw pe;
       }
       …
       return ast;
}
```

```
try { ast = parseFile(f); }
catch (ParseException pe) { pe.printStackTrace(); }
```


Kaszkád kivételek

```
ParseException
        at Test2.parseFile(Test2.java:19)
        ...

Caused by: IOException
        at Test2.nextString(Test2.java:12)
        ...
        at Test2.parseFile(Test2.java:17)
```

- a **`Throwable`** osztályban a *cause* (hibaok) állítható
- az ok-okozati lánc látszik a *stack-trace*-ben

Csomagoló-osztályok

Csomagoló-osztályok

- Minden primitív típushoz tartozik egy osztály
  - Integer, Byte, Boolean stb.
  - mert néha objektum kell, nem primitív érték
  - immutábilis
- Konstruktorok többféle paraméterrel
  - `Integer(int i)`, `Integer(String s)`
- Konstansok
  - MAX_VALUE, MIN_VALUE, SIZE
  - NaN, NEGATIVE_INFINITY stb.

Csomagoló-osztályok

- Transzformáló metódusok
  - `rotateLeft, reverse, signum, ...`
- Konvertáló metódusok
  - `intValue(), longValue(), parseInt(String s), valueOf(...), ...`
- Egyéb segédfüggvények
  - `Double.isNaN(double d), ...`

Csomagoló-osztályok és boxing

- Csomagoló ↔ primitív konverzió automatikusan

```
int a = 2;
Integer b = 3;
a = a + b;
Integer d = 3+3;  // int -> Integer
System.out.println(d*3); // Integer -> int
```

Csomagoló-osztályok és boxing 2

- A konverzió nem hatékony

```
public static void main(String args[]) {
       Integer i1 = Integer.parseInt(args[0]);// boxing
       Integer i2 = Integer.parseInt(args[0]);// boxing

       i1.equals(i2);       // igaz, nincs boxing
       i1 == i2;            // csak ha -128<i1≤127
       i1 <= i2;            // igaz, boxing

       Integer i3 = i1;
       i1++;                // növel, boxing
       i1 == i3;            // hamis (i1 megváltozott,
                            // új referenciája lett)
       i1 > 120;            // boxing
}
```


Változó hosszú paraméterlista

- Java 5 előtt: tömböt kell átadni
  - kényelmetlen

```
void foo(String s, Object[] oa) { for (Object o : oa) …; }
```

- Java 5 után: *varargs*
  - tömb és paraméter-sorozat is adható
  - csak a paraméterek végén

```
void foo(String s, Object… oa) { for (Object o : oa) …; }
```

```
Object[] oo = {"a", "b", "c"};
foo("X", oo);
foo("X", "j", "k", "l", "m", "n");
```

Annotációk

- Forráskód extra információval való bővítése

```
public @interface Copyright {
    String value() default "2008 Me";
}
@Copyright("2008 Bytemongers Limited")
public class ÁrvíztűrőTükörfúrógép { ... }
```

  - Deklaráció: *@interface*
  - Metódusok definiálják az annotáció tartalmát
    - visszatérési érték primitív, *String, Class, enum*
  - Lehet default értékük is

Annotáció használata

- Metaadat leírására valók
  - fordítóprogram számára
  - kódgenerátor számára (pl. DB-hez)
  - egyéb célból
- Pl: metódus felüldefiniálás

```
@Override
public int read() throws IOException {
      return super.read();
}
```

Memóriakezelés

Memóriakezelés

- C: memória-gondok
  - pointerek + aritmetika
  - void*
  - malloc/calloc/realloc/free
- C++ próbálta kezelni, nem sikerült
  - másoló konstruktor
  - virtuális destruktor
  - értékadás operátor
  - new/delete

```
a[3] ≡ *(a+3) ≡ *(3+a) ≡ 3[a]
```

```
class C : A, virtual B {
  int l; Complex c;
public:
  C(Complex k, int i)
    : A(i), c(k), l(i)
      { l++; }
};
```

Memóriakezelés 2

- Java beépített szemétgyűjtéssel (Garbage Collector, GC)
  - **`new`**: heap-en allokál
  - **`delete`**: nincs, a GC szabadít fel, ha szükséges
- GC felszabadítja a nem referált objektumokat
  - **`void finalize()`** meghívódik
  - a programból nem elérhető objektumokat gyűjti be
- GC explicit meghívása:
  - **`System.gc()`** or **`Runtime.gc()`**

Kódolási stílus

Azonosítók

- Változók, attribútumok és metódusok
  - **`camelCase`**, kezdőbetű kicsi, szavak kezdete nagy
    - **`getSecondBiggestNumber()`**
    - **`int importantVariable;`**
- Osztály- és interfésznevek
  - **`CamelCase`**, kezdőbetű nagy, szavak kezdete nagy
    - **`StringBuffer`**
- Csomagok nevei
  - végig kisbetűs
    - **`java.util`**

Zárójelezés

- Zárójelek pozíciója
  - sor végén nyitjuk

```
while (true) {
```

  - új sorban zárjuk, kötések egy sorban (pl. *else*)

```
if (a<b) {
  ...
} else {
  ...
}
```

Köszönöm a figyelmet!