



Ciklusok, függvények és hivatkozások



*Hallgassuk, mit mondanak
az objektumok*





Amit már megtanultunk...

- Bizonyos osztályban **objektumokat létrehozni** a **new** segédszóval és beszélgetni velük egy „komunikációs” változó segítségével
- **Saját osztályt létrehozni** mely a Turtle osztályt bővíti és metódusokat írni benne
- **Primitív típusú változókkal** dolgozni, numerikus (int, byte, short, long, double, float) és logikai (boolean) értékeket elmenteni
 - Le tudunk írni számtani kifejezéseket és értjük a számításukat
- Ismerjük a **feltételes parancsot** (if-else)



Kerge Kocka (1)

- Általános véletlen séta:

```
public void veletlenSeta(int lepesSzam) {  
    for (int i=0; i<lepeszSzam; i++) {  
        this.turn(Math.random() * 360);  
        this.step(10);  
        JPAZUtilities.delay(30);  
    }  
}
```



- Azt akarjuk, hogy a teknős ne bármennyivel forduljon el, hanem csak a **90 többszöröseivel** (0, 90, 180, 270, 360, ...)





Kerge Kocka (2)

- Milyen szöggel fordulhat el?:

- $0=0*90$, $90=1*90$, $180=2*90$, $270=3*90$, $360=4*90$, ...

Szükségünk van egy **egész** számú véletlen generátorra – a `Math.random()` csak valós számokat ad a $[0, 1)$ intervallumból

- Hogyan generáljunk ki egy véletlen egész számot 0 és 3 között?

```
int szorzat = Math.random() * 4;
```

Egész számú változóba nem tudunk elmentni **nem egész** számot!

Véletlen valós szám $[0, 4)$



Típuskonverzió (casting)

- A Java lehetővé teszi az érték típusának változtatását más értékre **típuskonverzió** által
- *Implicit* („automatikus“):
 - A Java képes megváltoztatni egész számot valósra:

```
int c = 10;  
double r = c;
```

- *Explicit* („a programozónak kell akarnia“)
 - A Java kérésre képes valós számból egészet csinálni **a tizedesrész levágásával**:

A c-ben 10 lesz

```
double r = 10.8231;  
int c = (int) (r);
```



Kerge Kocka(3)

```
int szog = 90 * (int) (Math.random() * 4);
```

Milyen típusra akarjuk változtatni az értéket (más típussal is működik **byte**, **long**, ...)

Melyik értéknek akarjuk a típusát változtatni

Vigyázz: Ha a típus **double**-ről **int**-re, változik, az érték nem kerekítődik hanem levágódik!

```
this.turn(szog);
```

Mivel a `turn` paraméterként **double**-t vár, itt implicit (automatikus) konverzió történik **int**-ről **double**-re-



Több matematikát akarunk!

- `Math.abs (szam)` – A szám abszolút értéke
- `Math.sqrt (szam)` – gyökvonás
- `Math.sin (szog)` – a szög szinusza radiánban
- `Math.PI` – π szám
- `Math.round (szám)` – **kerekítés** valós számról egészre

```
double szam = 3 * Math.sqrt(2);  
szam = Math.sqrt(Math.sin(szam));
```

$$3\sqrt{2}$$



Négyzetes spirál (újra)

- Tanítsuk meg a teknősnek a következő metódust

```
public void negyzetesSpiral()
```

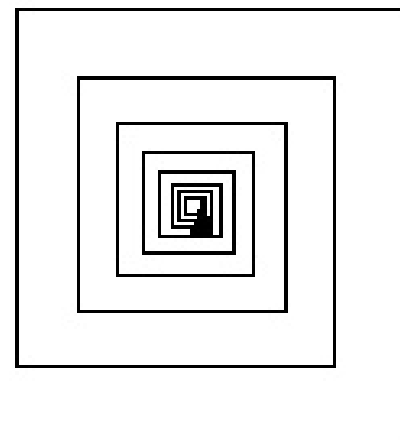
amely egy **négyzetes spirált** rajzol. A kezdeti méret 150 és minden lépés után a lépés/oldal hossza 10 %-kal csökken.

- Hány lépést tegyünk meg?

Lehetséges megoldás:

Addig rajzolunk, amíg „látni” – az 1-nél kisebb képések pontnak tűnnek, nem látni őket.

Hány lépés ez?





Feltételes ismétlés

```
while (feltétel) {  
    a ciklus parancsai  
}
```



- Amíg a **feltétel igaz**, a ciklus parancsai állandóan ismétlődnek ...
- A feltétel egy logikai kifejezés amely **tesztelése minden** ismétlés előtt végbemegy.

```
while (aktualisLepes >= 1) {  
    ...  
}
```



Beláthatatlan vég...

- A feltételes ismétlések **veszélye**, hogy hibát csinálunk és a feltétel mindig igaz lesz ...
- ... Mindig igaz feltétel = sose végződő (**végtelen**) ciklus ...

```
int k = 100;  
while (k > 10) {  
    this.moveTo(k, 30);  
}
```





For ciklus nagyító alatt

Létrejön az **int** típusú *i* változó, mely csak a for ciklus során él. Az *i* változó 0-val van inicializálva.

A feltétel, melynek **igaznak kell lennie**, hogy a parancsok végre lehessenek hajtva.

```
for (int i=0; i<10; i++) {  
    a ciklus parancsai  
}
```

Az *i* változót a **ciklus irányítóváltozójának** szoktuk nevezni.

Mi történjen, **minden alkalommal** miután lefutottak a ciklus parancsai (általában az irányítóváltozó értéke 1-gyel növekszik)

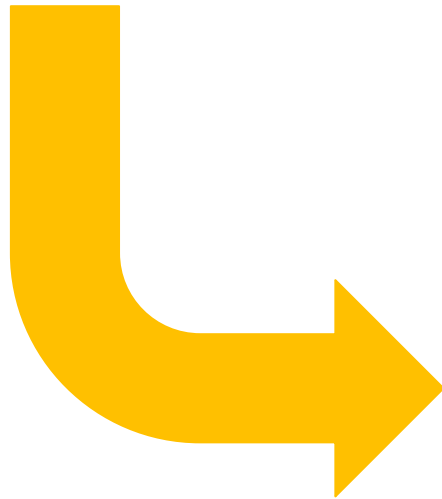


For ciklus nagyító alatt

```
for (int i=0; i<10; i++) {
```

a ciklus parancsai

```
}
```



```
int i=0;
```

```
while (i<10) {
```

a ciklus parancsai

```
i++;
```

```
}
```

Vigyázat: nem teljesen pontos!



Következmények

- Az i változó értéke minden iteráció (ismétlés) után **változik**

```
for (int i=0; i<10; i++) {  
    a ciklus parancsai  
}
```

- $i = 0, 1, 2, 3, 4, \dots, 9$
- Az i változó értékét használhatjuk a ciklus parancsaiban



A *break* és *continue* parancsok

- A **break**; paranccsal be tudjuk **azonnal fejezni** az egész ciklus végrehajtását
- A **continue**; paranccsal be tudjuk azonnal fejezni a ciklus aktuális iterációját

```
while (true) {  
    valamilyen parancsok  
    if (this.distanceTo(100, 100) > 200) {  
        break;  
    }  
    valamilyen parancsok  
}
```



Ha a feltétel igaz, **azonal** leállítjuk a ciklus végrehajtását.

break és **continue** a legközelebbi ciklusra vonatkoznak



do-while (csak érdekesség)

- do-while - hátultesztelő ciklus (a gyakorlatban nagyon ritkán használják):

```
do {
```

A ciklus parancsai

```
} while (feltétel);
```

A feltétel tesztelése **csak a parancsok végrehajtása után** történik.

Gyakori kérdés meghallgatásokon:
Tudunk boldogulni a 3 ciklus valamelyike nélkül? Miért?

Következmény: a do-while parancsai legalább egyszer **mindig** lefutnak.



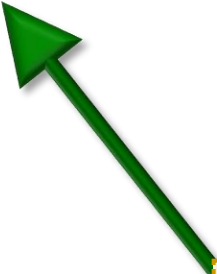
Metódusok, melyek számolnak

- Sok teknős metódus **visszaad** (kiszámol) ...
valamilyen értéket
 - *getX()* - visszaadja a teknős aktuális x koordinátáját
 - *directionTowards(double, double)* - visszaadja valamilyen pont irányát a teknőshöz viszonyítva
 - *distanceTo(double, double)* - visszaadja a teknős távolságát egy adott ponttól
- Vizsgáljunk meg néhány metódust az Object Inspector segítségével ...



Hogyan adjon vissza értéket a metódus?

```
public double negyzetesSpiral() {  
  
}
```



Az érték típusa, melyet a metódus visszaad

● Lépések:

- A **void** helyett az érték típusát írjuk (**int**, **double**, **float**, **byte**, ...), melyet a metódus visszaad
- **void** = visszaadok egy semmit („csak megcsinálom amit kell“)



A visszaadás parancsa

- Az érték visszaadására a **return** parancs szolgál:

return megtett;



Az érték, melyet
visszaadunk
(kiszámolunk) a
metódus során.

A **return** parancsban lévő értéknek muszáj kompatibilisnek lennie a deklarált visszaadandó értékkel!

return befejezi a metódus végrehajtását!



Mire kell figyelni

- A **return** parancs **befejezi** a metódus **minden további parancsának** végrehajtását. 
- Minden metódus amely nem **void**-ot ad vissza, **muszáj** minden végrehajtását a **return** **paranccsal befejeznie** (a Java ezt ellenőrzi).
- A visszaadott értéknek muszáj **kompatibilisnek lennie** a definiált visszaadandó értékkel
 - Az a metódus, mely úgy van definiálva, hogy *int* értéket ad vissza, nem adhat vissza valós számot (*double* értéket)
- A **return;** parancsot használhatjuk **void**-ot visszaadó metódusban is a metódus végrehajtásának azonnali megszakítására.



Teknős - tudósként

- Tanítsuk meg a teknőset érdekes számítások elvégzésére:
 - Talaljuk meg egy adott szám egész számú osztóinak számát
 - az **A** szám osztható a **B** számmal ha $A \% B == 0$
 - Találjuk meg két szám legnagyobb közös osztóját
 - Prímszám-e az adott szám
 - Számoljuk össze az adott szám számjegyeit
 - ...





Egész számok - trükkök

● Utolsó számjegy:

- $123 \% 10 = 3$, $34 \% 10 = 4$, $12345 \% 10 = 5$
- A `c` változó utolsó számjegye: `c % 10`

● Az utolsó számjegy levágása:

- $123 / 10 = 12$, $34 / 10 = 3$, $12345 / 10 = 1234$
- Szám létrehozása, mely az utolsó számjegy eltávolításával jön létre: `c / 10` (osztás egész számmal)

● Utolsó számjegy hozzáadása:

- $123 \leftarrow 8 = 123 * 10 + 8 = 1230 + 8 = 1238$
- `szam = szam * 10 + szamjegy;`



Számjegyek összege

- A 294 számjegyösszege $294 = 2 + 9 + 4 = 15$

Kiválasztjuk az utolsó számjegyet

```
public int szamjegyOssze(int szam) {  
    int eredmeny = 0;  
    while (szam > 0) {  
        int szamjegy = szam % 10;  
        eredmeny = eredmeny + szamjegy;  
        szam = szam / 10;  
    }  
    return eredmeny;  
}
```

A számjegyet hozzáadjuk az eredményhez

Létrehozunk egy új számot, amely az előző rövidítettje lesz.



Hogy mások is megértsék!

- A programokat úgy kell írni, hogy érthetőek és áttekinthetőek legyenek ... nemcsak a Java számára, hanem **ez emberek számára is**

```
// kiválasztjuk az utolsó számjegyet
```

```
int szamjegy = szam % 10;
```

```
// hozzáadjuk a számjegyet az  
összeghez
```

```
eredmeny = eredmeny + szamjegy;
```

```
// lerovíditjuk a számot
```

```
szam = szam / 10;
```



Kommentárok

- Minden jel a `//` után egészen a sor végéig kommentárnak van tekintve
- Minden jel a `/*` és a `*/` között kommentárnak van tekintve
- Kommentárt bárhová írhatunk ...

Kommenteljétek a programaitokat !!!



Kiíratás „Eclipse-be“

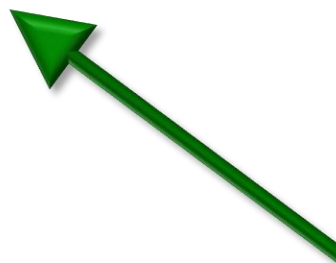
- Néha jó dolog metódusokat Object Inspector nélkül használni...

- Írni tudunk „Eclipse-be“ is:

```
System.out.println(Math.PI*8);
```

```
System.out.println(franklin.szamjegyOsszeg(293));
```

```
System.out.println(„Hello World“);
```



Ak chceme vypísať nejaký text, dáme ho do úvodzoviek " "

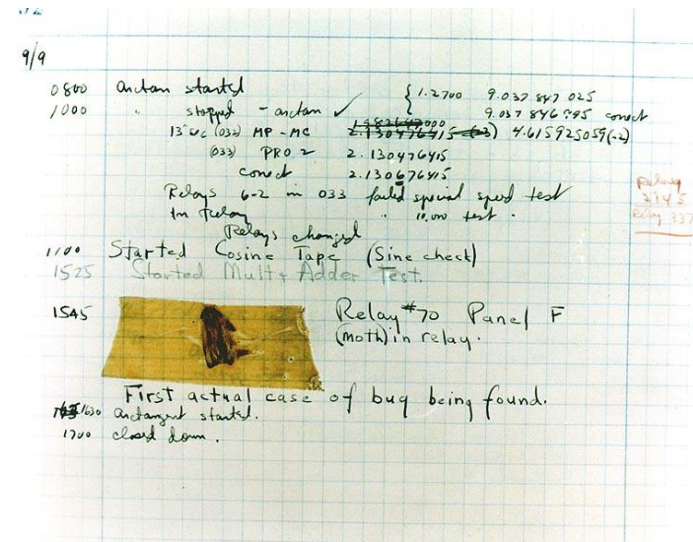


Debugolás

- Debugolás = bogártalanítás, hibakeresés



- Az ember hibázó lény, a számítógép szerencsére nem ...
- **A debugolás** alkalmazása **elengedhetetlen** a legtöbb program fejlesztése során ...
- Szemléltetés ...





Változók Javaban (újra)

- A Java változóinak két típusa:
 - **primitív típus**
 - *int, byte, short, long, double, float, boolean és char*
 - A char-on kívül mindent ismerünk
 - életcél: **raktározni egy egyszerű értéket**
 - **referencia típus**
 - életcél: **objektumra hivatkozni** (lehetővé tenni az objektumokkal való kommunikációt)

WinPane pane;



A `pane` változó képes hivatkozni a `WinPane` osztály objektumaira.



Referencia (hivatkozás)

Referencia típusú változók

```
ObjectInspector oi;
```

oi

```
Turtle franklin;
```

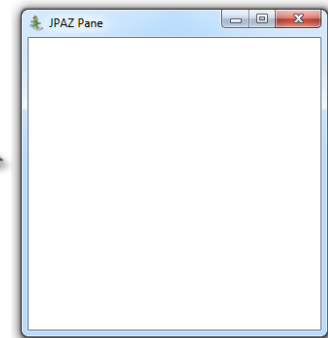
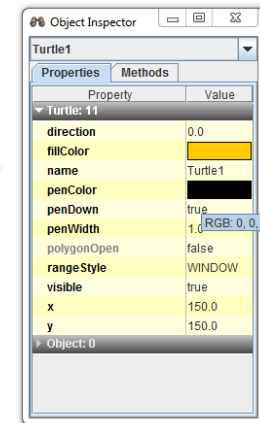
franklin

```
WinPane pane;
```

pane

A referencia típusú változó **a hivatkozást tartalmazza** (linket, kapcsolatot) **egy** bizonyos **objektumra** ...

„objektumok világa“





Referencia típusú változó deklarálása

WinPane pane;

Milyen osztály
objektumaira
hivatkozhat az éppen
születő változó

A változó neve

- A referencia típusú változó nem hivatkozhat bármilyen objektumra!
- Mint ahogy a már ismert változók mindegyikét, a referencia típusú változót is használat előtt **inicializálni kell**.



A hivatkozás metaforája (1)

- Minden ember Szlovákiában **egyértelműen** **identifikálható a születési száma által**, melyet születésénél kap meg...
- Minden objektum a „Java világában” **identifikálható a „java-s születési számával”**
 - Némely objektum elárulja a „java-s születési számát” a **toString** metódus által



„java-s születési szám”



Object: WinPane (sk.upjs.jpaz2.WinPane@2e257f1b)
Class: WinPane (sk.upjs.jpaz2.WinPane)



A hivatkozás metaforája (2)

int szám;

A `sam` változó egy számot képes raktározni

boolean ervenyesseg;

Az `ervenyesseg` változó egy logikai értéket képes raktározni

WinPane pane;

A `pane` változó egy WinPane osztály objektumának a „születési számát” képes raktározni, vagy egy speciális **null** értéket

2e257f1b





A referencia típusú változó értéke

- A referencia típusú változó raktározhat:
 - Hivatkozást gy objektumra („az objektum születési száma“)
 - **null** - speciális érték, mely azt mondja, hogy a változó **nem hivatkozik** semmilyen objektumra

```
pane = null;
```

A pane nem hivatkozik
semmilyen (WinPane
típusú) objektumra

```
if (pane != null) {
```

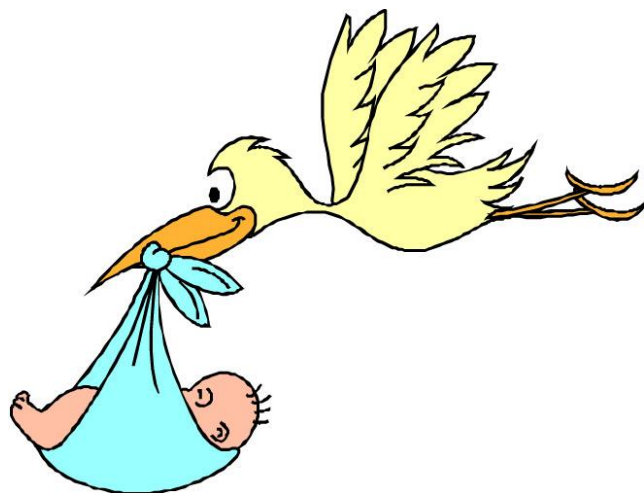
Ellenőrzés, hogy a
változó hivatkozik-e
valamilyen objektumra
(az értéke nem **null**)



Objektum létrehozása (1)

- Az objektumokat (az „objektumok világában”) a **new** parancs által hozzuk létre, ennek **eredménye** egy **hivatkozás** („születési szám”) a létrejövő objektumra.

```
new WinPane ();
```



Annak az osztálynak a neve, amelyiknek az objektumát hozzuk létre



Objektum létrehozása (2)

● Konstruktor

- speciális „dolog“, mely **létrehozza és inicializálja** az adott osztály objektumát
- konstruktor tartalmazhat **paraméterekt**, melyeket zárójelbe () írunk az objektum létrehozásánál
- Az osztálynak **több konstruktora is lehet** melyek a paraméterekben különbözhetnek

A konstruktor paraméterei = pontosítás, milyen objektumot akarunk létrehozni

```
new WinPane ( ) ;
```

```
new WinPane (400, 400) ;
```





A hivatkozás hozzárendelése

```
WinPane panel1;  
panel1 = new WinPane();  
WinPane pane2;  
pane2 = panel1;
```

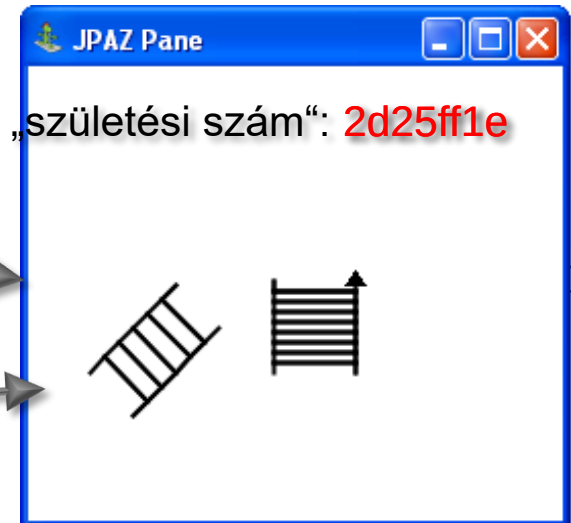
A végén a két változó
panel1 és pane2
ugyanarra az objektumra
fognak hivatkozni.



panel1



pane2





A hivatkozás metaforája (3)

```
WinPane pane;
```

```
pane = new WinPane();
```

```
...
```

```
pane.clear();
```

Létrejön a `pane` változó, mely képes elmenteni egy `WinPane` típusú objektum „születési számát“

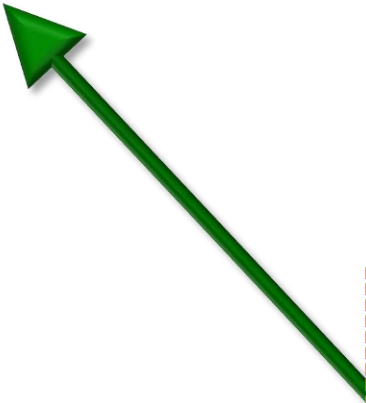
Létrehozzuk a `WinPane` osztály objektumát és a „születési számát“ elmentjük a `pane` változóba.

Annak az objektumnak, melynek a „születési száma“ a `pane` változóban van elmentve, azt mondjuk, hogy végezze el a `clear` metódust.



A hivatkozások egyenlősége

```
if (pane1 == pane2) {  
  
}
```



Teszteljük, hogy a referencia típusú változók *pane1* és *pane2* **ugyanarra** az objektumra hivatkoznak-e.

Metafora: teszteljük, hogy a *pane1* a *pane2* ugyanazt a „születési számot” tartalmazzák-e



Mire kell emlékezni

- **Nem igaz**, hogy az **inicializálatlan** referencia típusú változó **null**-t tartalmaz. A változó egyszerűen csak inicializálatlan (a hozzárendelésen kívül nem tudunk vele másat csinálni)
- Az objektumokat csak **létre tudjuk hozni**, nem tudjuk őket tönkretenni
- A Java illúziója: az objektumok örökké élnek (ill. amíg véget nem ér a program)
 - ...ellentétben a lokális változókkal
- Egy objektumra **hivatkozhat több** változó
- Ha az objektumra **nem létezik hivatkozás**, nem tudunk vele kommunikálni - az objektum elveszett.



Karakterek

- **Karakterek** elmentésére (primitív érték) a **char** típusú változó szolgál

```
char karakter = 'a';
```

- **Betűk jelei** (karakterek konkrét értékei) aposztrófba írjuk 'a', 'A', ',', ' ', ...
- A **char** típusú változóba a UNICODE kódolás első 65536 karakterének bármelyikét el tudjuk menteni (nyelvek speciális karakterei, ázsiai nyelvek jelei, ...) = majdnem bármi ami eszünkbe juthat
 - A speciális karakterek kódolásáról később





Amikor egy betű nem elég

- A gyakorlatban egy karakter elég kevés, nekünk karakterek sorozatát kell elmentenünk...
 - Karakterek sorozata alkot szavakat, mondatokat, ...
- **Karakterlánc** = karakterek sorozata
- Javában a karakterek sorozatát **objektumokban** tároljuk
 - Míg némelyik objektum rajzol, számol, mások értékeket raktároznak ...





Karakterláncok

- Karakterláncot a **String** osztály objektumában tárolhatjuk

```
String s = new String („Hello“);
```

Az `s` referencia típusú változó, amely hivatkozik a `String` osztály objektumára (raktározza a „születési számát“)

Létrehozzuk a **String** típusú objektumot, melynek „életcélja“ az 5-karakteres „Hello“ láncot raktározni.

Karakterlánc (konkrét érték), a karaktereket idézőjelbe írjuk `“ ”`.



Vizsgáljuk meg a String osztályt

```
ObjectInspector oi = new ObjectInspector();  
String s = new String("Java");  
oi.inspect(s);
```

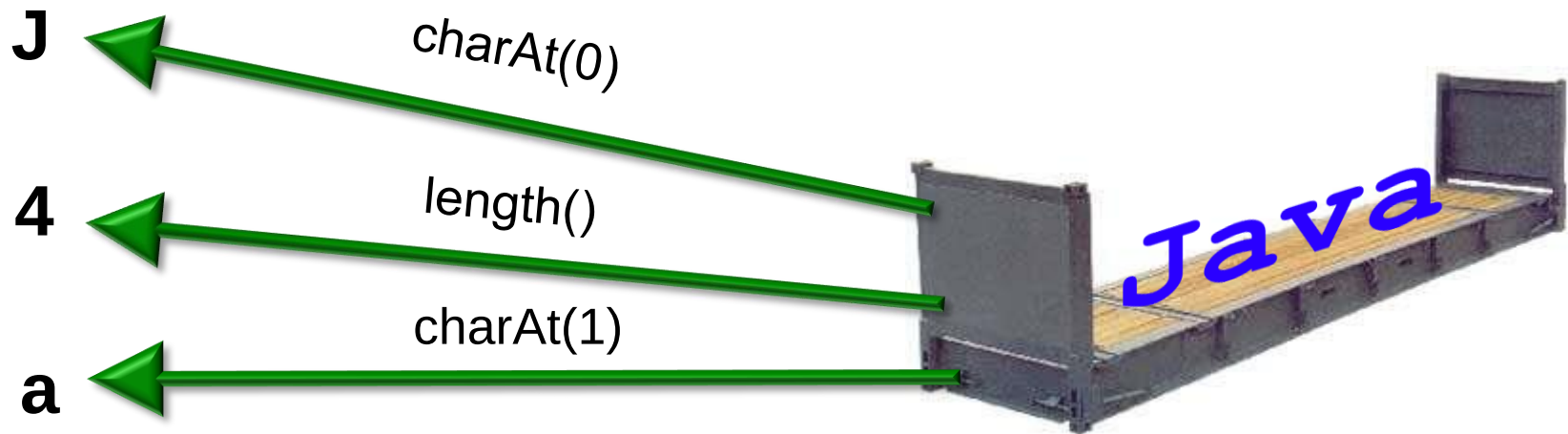
- Vizsgáljuk meg, mit csinálnak a String osztály objektumainak metódusai ...
 - Mit csinál a **charAt** és **length**?





A rövid megfigyelés eredménye

- `charAt(int)` - visszaad egy **karaktert** az *i*-edik **pozíción**, a karakterek számozása **0**-val kezdődik
- `length()` - visszaadja a **karakterlánc hosszát**





A betű előfordulásának száma (1)

- Tanítsuk meg a teknőst megszámolni az `'a'` betű előfordulását a szóban ...

- A `"Java"` szóban az `'a'` betű kétszer fordul elő

`public int` elofordulasA(String s)

Vissza akarjuk adni hányszor (egész szám) fordul elő az `a` betű az adott objektumban.

A paraméter egy hivatkozás egy String típusú objektumra („az objektum születési száma”), melyben az `a` betű számát számoljuk.



A betű előfordulásának száma (2)

```
public int elofordulasA(String s) {
```

```
    int szam = 0;
```

Felhasználjuk a `szam` változót mint számláló, hányszor válaszolt a `charAt` metódus `a` betűvel.

```
    for (int i=0; i<s.length(); i++) {
```

```
        if (s.charAt(i) == 'a') {
```

```
            szam++;
```

```
        }
```

```
    }
```

```
    return szam;
```

Kipróbáljuk a karakterlánc összes karakterét (`0, 1, ..., hossz-1`) és kérjük az adott index karakterét. Ha az `a`, akkor növeljük a számláló értékét 1-gyel.

```
}
```



Gyors ismételés

- Típuskonverzió (explicit, implicit)
 - véletlen **egész** számok generálása
- Ciklusok: **for**, **while**, **do-while**
 - **break** és **continue** a megszakításukra
- Értéket visszaadó metódusok és a **return** parancs
- Kommentárok
- Debugolás
- Referencia típusú változók és konstruktorok
- **char** típus és a `String` osztály