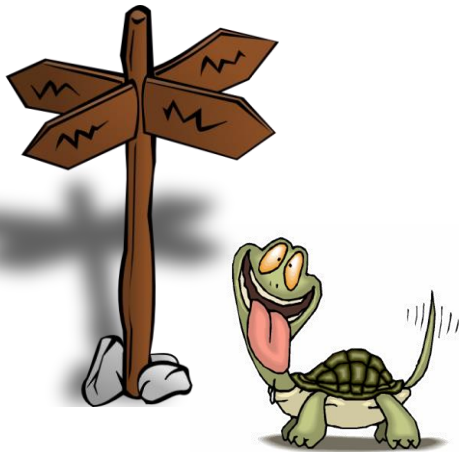
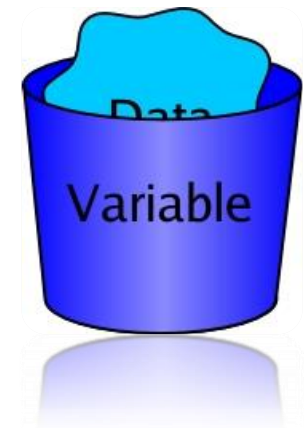




## 2. prednáška (28.9.2015)



# *Premenné a podmienky*



*Naozajstné programovanie  
začína*





# Na minulej prednáške (1)

- **Vytvorenie objektu** triedy a premennej (napr. jozko), cez ktorú s vytvoreným objektom komunikujeme:

```
Turtle jozko = new Turtle();
```

- **Volanie metód** nad objektmi („rozprávanie sa“ s objektom):

```
jozko.moveTo(30, 50);
```

- Vieme vytvárať nové triedy **vylepšovaním** existujúcich:

```
public class MojaTurtle extends Turtle {
```

```
}
```



## Na minulej prednáške (2)

- Vylepšovanie spočíva v **pridávaní** nami definovaných **metód** (aj s parametrami)

```
public void stvorec(double strana) {  
    ... naše príkazy ...  
}
```

- Objekty vylepšenej triedy majú všetky metódy a vlastnosti, ktoré mala pôvodná trieda + novodefinované
- V metódach vylepšených metód používame na oslovenie vykonávateľa („samého seba“) slovíčko **this**:

```
this.step(100);
```



# Na minulej prednáške (3)

- „magická **for**-mulka“ na opakovanie skupiny príkazov:

Koľko krát sa má niečo opakovať

```
for (int i=0; i<3; i++) {
```

```
    this.step(100);
```

```
    this.turn(120);
```

```
}
```

Príkazy, ktoré sa majú opakovať





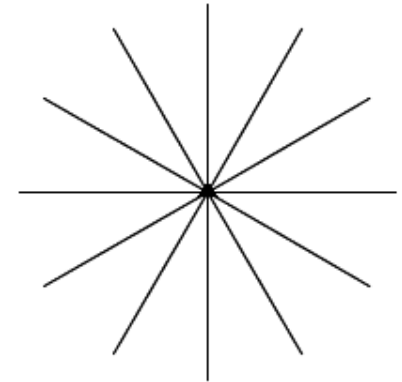
# Na minulej prednáške (4)

```
public void nVlocka(int n, double dlzkaLuca)
```

- Návod:

- n krát zopakuj:

- Sprav krok dĺžky *dlzkaLuca*
- Sprav krok späť dĺžky *dlzkaLuca*
- Otoč sa o  $360 / n$  stupňov (n-tina plného uhla)



```
for (int i=0; i<n; i++) {
    this.step(dlzkaLuca);
    this.step(-dlzkaLuca);
    this.turn(360 / n);
}
```

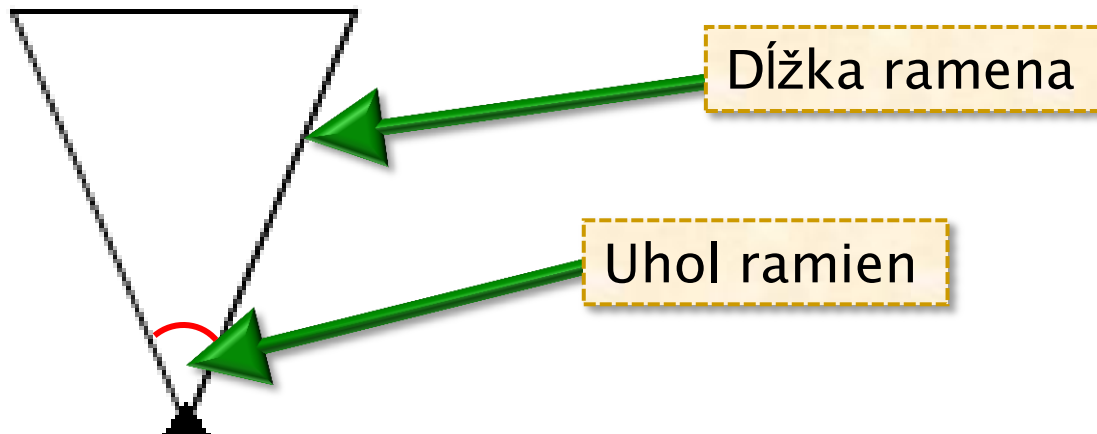
Parameter vieme použiť aj na riadenie počtu opakovaní cez „for“-mulku



# Rovnoramenný trojuholník

- Ako naučiť korytnačky namaľovať **rovnoramenný trojuholník** so zadanou dĺžkou ramena a zadaným uhlom, ktorý zvierajú ramená?

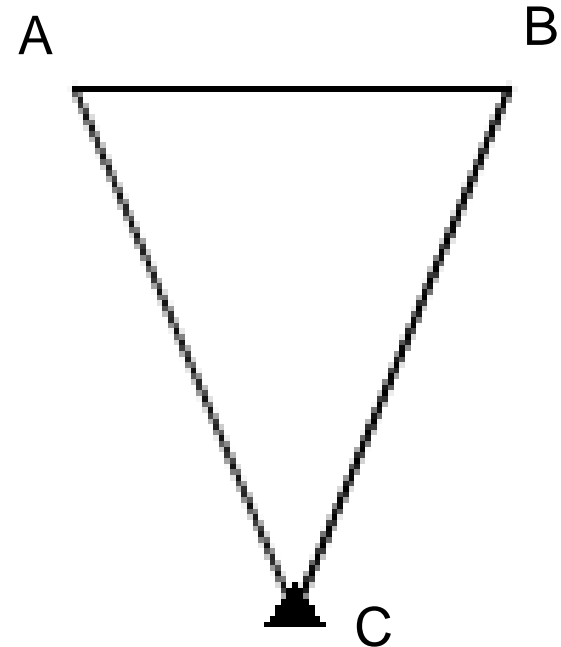
```
public void rovnoramenny(double rameno, double uhol)
```





# Rovnoramenný trojuholník

1. Otoč sa o **uhol/2** vľavo (v bode C)
2. Sprav krok dĺžky ramena (do bodu A)
3. Sprav krok späť (do bodu C)
4. Otoč o uhol **uhol** vpravo (v bode C)
5. Sprav krok dĺžky ramena (do bodu B)
6. Sprav krok do bodu A

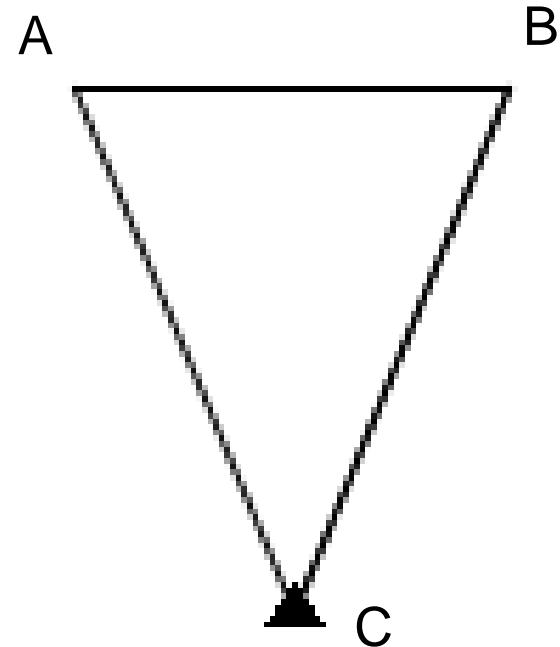


**Ako spravíme krok do bodu A,  
ak nepoznáme jeho súradnice?**



# Rovnoramenný trojuholník

1. Otoč sa o **uhol/2** vľavo (v bode C)
2. Sprav krok dĺžky ramena (do bodu A)
  - som v bode A, nejakú si zapamätám svoju x-ovú a y-ovú súradnicu
  - `this.getX()`
  - `this.getY()`
3. Sprav krok späť (do bodu C)
4. Otoč o uhol **uhol** vpravo (v bode C)
5. Sprav krok dĺžky ramena (do bodu B)
6. Sprav krok do bodu A
  - spravím `this.moveTo(?, ?)` na súradnice, ktoré som si zapamätal na konci operácie (2)



Potrebujeme niečo na zapamätanie hodnoty tak, aby sa k tejto hodnote dalo dostať neskôr.





# Ako si pamätat?





# Ako si pamätať ...

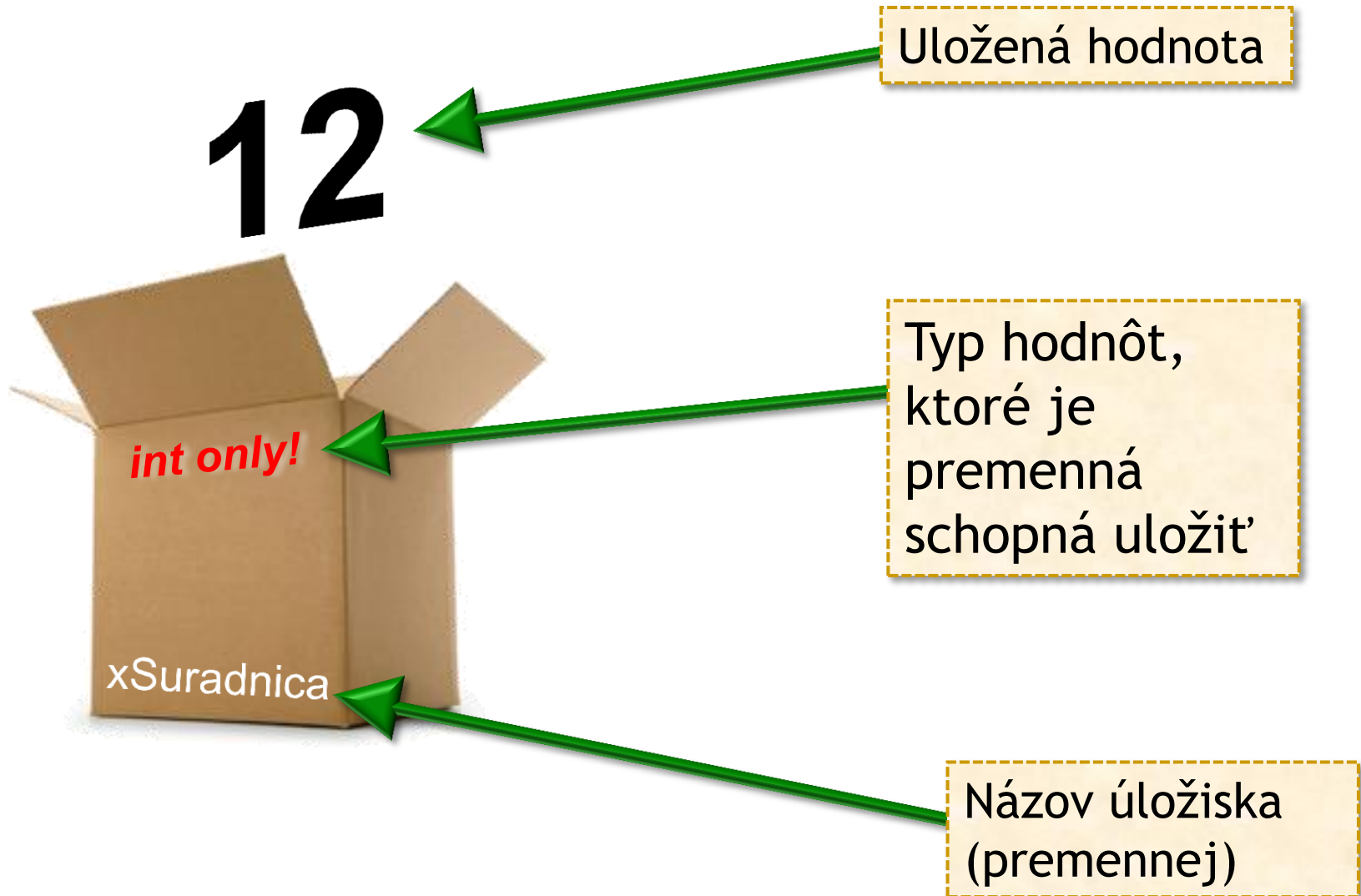
- na zapamätanie hodnôt vieme vytvoriť a použiť **premenné**...
- premenná:
  - má **meno** (označenie, pomenovanie)
  - má **typ** - povolené hodnoty, ktoré do nej vieme uložiť (tak ako parametre)

**Pozor:** premenné v matematike sú čosi iné.

**pomenované úložisko jednej hodnoty daného typu**



# Premenné - predstava





# Čo potrebujeme vedieť?

- Základne otázky o premenných ...
  - Ako vytvoriť premennú?
  - Ako pomenovať premennú?
  - Ako hodnoty môže premenná uchovávať?
  - Ako uložiť do premennej nejakú hodnotu?
  - Ako z premennej v nej uloženu hodnotu prečítať?
  - Kedy premenná končí svoju životnú púť?





# Vytvorenie premennej

- Príkaz na vytvorenie premennej:

```
double xPoziciaA;
```



Typ povolených  
hodnôt, ktoré možno  
uložiť v premennej

Názov premennej

- Vytvorená premenná je **neinicializovaná**, t.j. nie je v nej uložená žiadna hodnota.



# Nastavenie hodnoty premennej

- Príkaz na nastavenie hodnoty premennej:

```
xPoziciaA = 30.5;
```

Názov premennej,  
ktorej priradíme  
hodnotu

Hodnota, ktorú chceme do  
premennej uložiť (priradiť)

- Uložiť môžeme aj výsledok volania metódy:

```
xPoziciaA = this.getX();
```

- Uloženie hodnoty do neinicializovanej premennej túto premennú **inicializuje**.



# *dva v jednom*

- Vytvorenie premennej spolu s inicializáciou (prvotným nastavením hodnoty)

```
double xPoziciaA = 30.5;
```

Skratená verzia pre:

```
double xPoziciaA;  
xPoziciaA = 30.5;
```

- **Rada:**

- Je dobrým zvykom premennú hneď pri vytvorení aj inicializovať.
- S **neinicializovanou** premennou **nemožno pracovať**, Java toto prísne kontroluje!

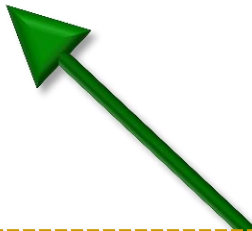


# Čítanie hodnoty premennej

- Názov premennej **zastupuje hodnotu** v danom okamihu v nej uloženú (jej priradenú)!

```
double dlzkaKroku = 30;
```

```
korytnacka.step(dlzkaKroku);
```



Korytnačka sa posunie o toľko, aká hodnota je **aktuálne** uložená v premennej *dlzkaKroku*, t.j. o 30





# Terminológia

- **Deklarácia premennej** - príkaz vytvorenia premennej
- **Príkaz priradenia** - príkaz na uloženie novej hodnoty do premennej (znak =)
- **Typ premennej** - typ povolených hodnôt, ktoré môžu byť uložené v premennej
- **Lokálna premenná** - každá premenná vytvorená (deklarovaná) vo vnútri metódy
- **Literál** - konkrétna hodnota (napr. 30, 2.4, 1) použitá v príkazoch programu



# Konečne trojuholník ...

```

public void rovnoramenny(double rameno, double uhol) {
    this.turn(-uhol/2);
    this.step(rameno);
    double xPoziciaA = this.getX();
    double yPoziciaA = this.getY();
    this.step(-rameno);
    this.turn(uhol);
    this.step(rameno);
    this.moveTo(xPoziciaA, yPoziciaA);
}

```

Do premenných *xPoziciaA* a *yPoziciaA* sme uložili súradnice aktuálnej pozície korytnačky (bodu A)

Využijeme hodnoty uložené v premenných ako hodnoty parametrov pri volaní metódu.



# Iné príklady ...

- Korytnačka sa po skončení vykonávania príkazov metódy vráti tam, kde bola na začiatku ...
- Korytnačka je po skončení vykonávania príkazov metódy v stave, v akom bola pred vykonaním metódy ...
- Metóda na nakreslenie čiary z  $(x_1, y_1)$  do  $(x_2, y_2)$   

```
public void ciara(  
    double x1, double y1,  
    double x2, double y2)
```



# O svete napravo od =

- V príkaze priradenia napravo od = môžeme písať ľubovoľné **aritmetické výrazy** ...
- Pred priradením hodnoty sa výraz napravo od = najprv vyhodnotí („vyčísli“) a táto hodnota sa uloží (priradí do premennej) ...

```
double cislo = 10;
```

```
double mocnina = cislo * cislo;
```

```
mocnina = cislo * cislo * cislo;
```



# Čo sa deje?

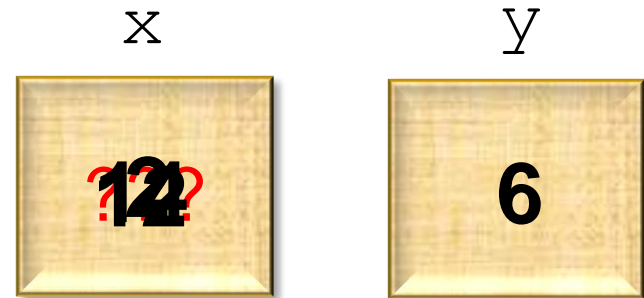
→ **int** x;

x = 2;

**int** y = 6;

x = 2 \* y;    x ◀ 2 \* 6;

x = x + 2;    x ◀ 12 + 2;



- Vždy sa najprv vyhodnotí výraz napravo od =
  - volania metód sa nahradia výsledkami volaní
  - mená premenných sa nahradia aktuálnymi hodnotami
  - numerický výraz sa vypočíta a výsledok sa uloží do premennej



# O svete napravo od =

- Ak priradíme hodnotu do premennej typu `double` (`int`), napravo od `=` môžeme napísať ľubovoľný **aritmetický výraz** vypočítajúci nejaké reálne číslo (celé číslo) - **kompatibilnú hodnotu**.
- **Poznámka:** aritmetické výrazy môžeme písať aj na miestach, kde v Jave zadávame hodnoty parametrov volanej metódy

```
this.setDirection(2*natocenie);
```

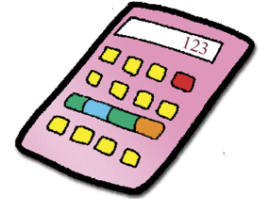


Pred volaním metódy sa aritmetický výraz najprv vyhodnotí.



# Aritmetický výraz

- V aritmetickom výraze môžeme použiť:



- symboly **matematických operácií**:

- \* (násobenie), + (sčítanie), - (odčítanie), / (delenie),
- % (zvyšok po delení):  $10 \% 3$  je 1,  $6 \% 8$  je 6,  $12 \% 4$  je 0

- okrúhle zátvorky ( )
- mená premenných zastupujúce v nich uložené hodnoty
- číselné literály (t.j. konkrétne čísla)
- hodnoty, ktoré sú výsledkom volaní metód:

```
double posun =
    3 * this.distanceTo(100, 200);
```



# Špirála

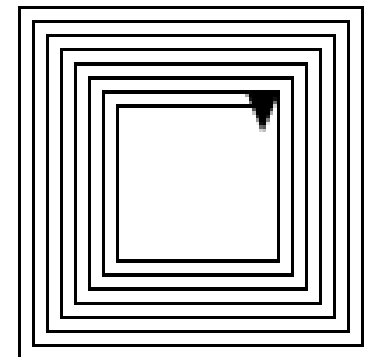
- Naučme korytnačku metódu:

```
public void stvorcovaSpirala(int pocetStran)
```

- nakreslí **štvorcovú špirálu** so zadaným počtom strán
- prvá strana má dĺžku 150
- každým krokom sa dĺžka strany (kroku) zníži o 2



150, 148,  
146, 144, ...







# O nefungujúcej hviezdke ...

- Prečo nefunguje jednoduchá hviezdka pre každú hodnotu  $n$ ?
- Odpoveď: aj **hodnoty majú svoj typ** ...
  - 360 je celé číslo (**int**)
  - 360.0 je reálne číslo (**double**)
  - matematické operácie závisí od typu operandov:
    - celočíselné delenie:  $5/2=2$  nie  $2.5$
    - **int** / **int** » **int** (vždy, keď oba operandy / sú **int**-y)
    - klasické delenie:  $5.0/2=2.5$
  - oprava: **this**.turn(360.0 / n);



# Pravidlá o typoch

- Ak hodnoty operandov sú **celé čísla**, výsledkom operácie je **celé číslo** (desatinná časť výsledku je odrezaná)
- Ak **jeden** z operandov je **reálne číslo**, výsledkom operácie je **reálne číslo**
- Do premennej pre celočíselné hodnoty (typu `int`) môžeme uložiť len hodnotu, ktorá je celé číslo !
- Príklad:
  - $int + double \gg double$
  - $int + int \gg int$



# Kedy premenná zaniká? (1)

- Prislúchajúce kučeravé zátvorky `}` definujú **blok príkazov**
- V bloku príkazov môžu byť iné bloky príkazov:
  - v bloku príkazov celej metódy je podblok príkazov *for* opakovania
- **Premenná zaniká, keď sa skončí vykonávanie toho bloku príkazov, v ktorom vznikla!**
- **Rozsah platnosti (scope) premennej** - od miesta vzniku (deklarácie) premennej po uzatváraciu kučeravú zátvorku toho bloku, v ktorej bola deklarovaná.



# Kedy premenná zaniká? (2)

```
for (int i=0; i<20; i++) {  
    double xPozicia = this.getX();  
    this.step(10);  
    this.turn(10);  
    this.setX(xPozicia + 8);  
}
```

Premenná  
xPozicia sa  
vytvorí a zanikne  
celkom 20 krát





Zátvorka, na  
ktorej premenná  
xPozicia  
zaniká





# Typy primitívnych premenných

## ● Celočíselné hodnoty:

- **byte** (-128 až 127) 
- **short** (-32 768 až 32 767) 
- **int** (-2 147 483 648 až 2 147 483 647) 
- **long** (-9 223 372 036 854 775 808 až 9 223 372 036 854 775 807) 

rôzne typy = rôzne rozsahy povolených hodnôt a zabratá pamäť

## ● Reálne čísla:

- **double** - lepšia presnosť, zaberá viac pamäte
- **float** - menšia presnosť, zaberá menej pamäte



primitívny = jednoduchý



# Náhodou o náhodách ...

- Java poskytuje funkciu, ktorá vygeneruje **náhodné reálne číslo** medzi 0 a 1:

`Math.random()`  
 $< 0, 1)$



- Príklady:

- náhodné natočenie korytnačky:

```
korytnacka.turn(Math.random() * 360);
```

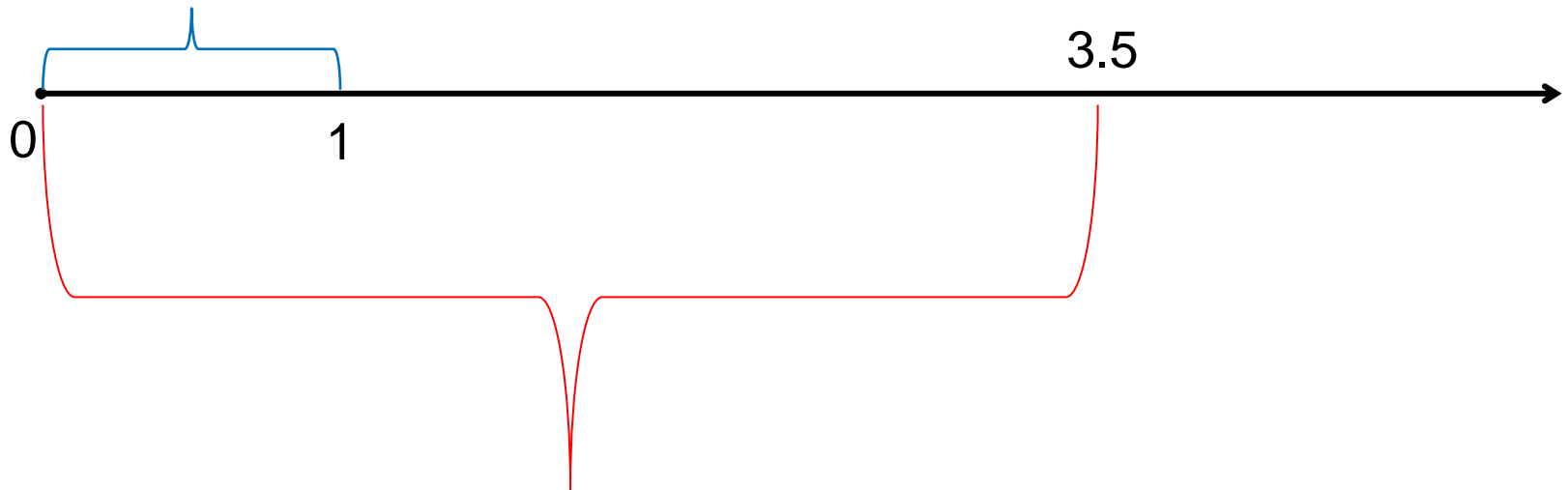
- náhodná dĺžka kroku korytnačky medzi 10 a 25:

```
korytnacka.step(10 + Math.random() * 15);
```



# Náhodné čísla

`Math.random()`



`Math.random() * 3.5`

- Náhodné číslo z intervalu  $<a, b)$ :

$$a + \text{Math.random()} * (b - a)$$



# Zbesilá korytnačka

- Naprogramujeme spomalený náhodný pohyb korytnačky ...

```
public void nahodnaPochodzka(int pocetKrokov) {  
    for (int i=0; i<pocetKrokov; i++) {  
        this.setDirection(Math.random() * 360);  
        this.step(10);  
        JPAZUtilities.delay(30);  
    }  
}
```

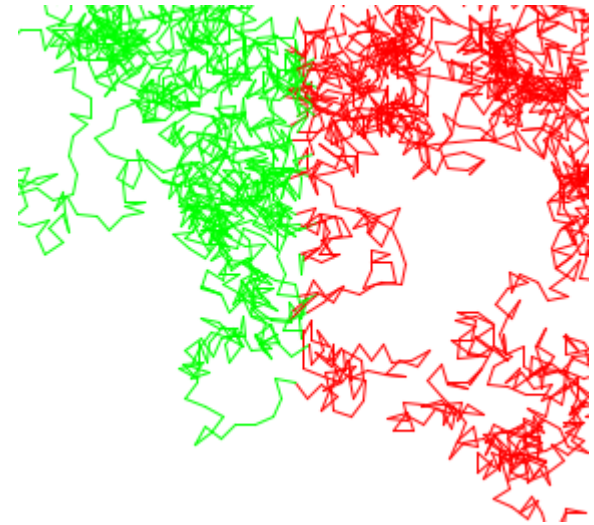






# Zbesilá korytnačka a farba

- Chceme, aby korytnačka menila farbu kresliaceho pera podľa toho, kde sa práve nachádza:
  - **Ak** je jej x-ová súradnica menšia ako 150, **tak** kreslí zelenou farbou, **inak** kreslí červenou farbou.





# Podmienkový príkaz (1)

Podmienka

Príkazy na vykonanie,  
ak podmienka platí

```
if (this.getX() < 150) {  
    this.setPenColor(Color.green);  
} else {  
    this.setPenColor(Color.red);  
}
```

Príkazy na vykonanie,  
ak podmienka neplatí



# Podmienkový príkaz (2)

```
if (podmienka) {  
    príkazy ak podmienka platí  
} else {  
    príkazy ak podmienka neplatí  
}
```





# Logický výraz (1)

- **Podmienka** sa zadáva vždy ako **logický výraz**
- Logický výraz je výraz, ktorého výsledkom je **pravdivostná hodnota**: pravda/nepravda
- Logický výraz sa skladá z:
  - Porovnávacích operátorov pre číselné hodnoty
    - Ostrá nerovnosť  $>$ ,  $<$
    - Neostrá nerovnosť  $>=$ ,  $<=$
    - **Rovnosť**  $==$
    - Nerovnosť  $!=$
  - Logických spojok (a, alebo, ...)

**Pozor, nie =**  
(častá chyba!)





# Logický výraz (2)

- Logické spojky:

- $(\text{vyraz1}) \ \&\& \ (\text{vyraz2}) \dots$  platí **vyraz1 a vyraz2**  $\wedge$
- $(\text{vyraz1}) \ || \ (\text{vyraz2}) \dots$  platí **vyraz1 alebo vyraz2**  $\vee$
- $!(\text{vyraz})$  platí práve vtedy ak neplatí vyraz  $\neg$

- Príklady:

```
x == 2
```

```
this.getX() > 100
```

```
(x >= 100) && (x <= 200)
```

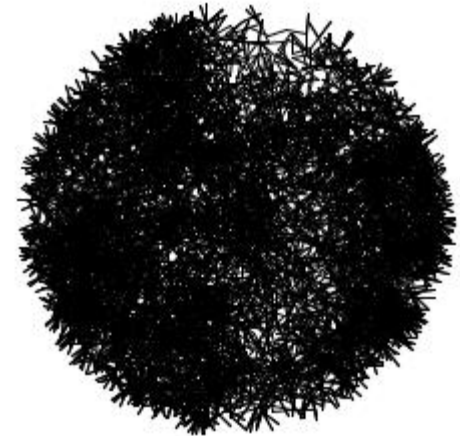
```
(x < 30) || (x > 100)
```

```
this.distanceTo(100, 100) > 300
```



# Zbesilá korytnačka na povrázku

- Vytvorme metódu realizujúcu zadaný počet náhodných krokov tak, aby korytnačka sa nikdy nevzdialila viac ako 100 od miesta kde začala ...
- Jeden krok:
  - Náhodne sa otoč
  - Sprav krok dĺžky 10
  - **Ak** si d'alej ako 100 od začiatku, **tak** sprav krok späť





# Podmienkový príkaz bez else

- Niekedy potrebujeme niečo spraviť, ak je splnená podmienka, ale **v opačnom prípade** nepotrebujeme spraviť **nič**
- Namiesto napísania prázdnej *else* vetvy ju môžeme pokojne vynechať:

```
if (podmienka) {  
    príkazy ak podmienka platí  
}
```



# Špecialitky ...

- Za kučeravou zátvorkou sa nikdy nepíše bodkočiarka!
- Na uloženie pravdivostnej hodnoty slúži typ **boolean** s literálmi **true** (pravda) a **false** (nepravda):

```
boolean blizko = (this.distanceTo(100, 100) < 30);
```

```
boolean somPravda = true;
```

```
boolean somNepravda = false;
```





# „Tajné skratky“

- Ak  $x$  je premenná celočíselného typu, tak:
  - $x++$ ; je to isté ako  $x = x + 1$ ;
  - $x--$ ; je to isté ako  $x = x - 1$ ;
  - $x += 5$ ; je to isté ako  $x = x + 5$ ;
  - $x *= y$ ; je to isté ako  $x = x * y$ ;
  - ... a ďalšie ...
- Okrem  $x++$  a  $x--$  ostatné skratky využívame s veľkou rozvahou! **Nie je umenie** napísať program, ktorý bude **t'ažko prečítať** a **pochopiť**.





# Čo ešte nevieme ...

- Parametre metódy nie sú nič iné, ako obyčajné premenné, ktorých hodnota je inicializovaná tou hodnotou, s akou sa metóda volá ...
- Pre **mená** premenných treba dodržiavať **pravidlá** slušného pomenovania:
  - nesmie začínať číslom, bez medzier
  - len malé písmená, veľké písmeno je použité pre prvé písmeno vo viac-slovných názvoch druhého a ďalšieho slova
    - mojaVelmiDlhaPremenna



# O čom to dnes bolo?

## ● Premenné

- pomenované **úložiská** uchovávajúce jednu hodnotu konkrétneho typu
- uchovávanú hodnotu môžeme zmeniť
- Aritmetické výrazy a „zradná“ operácia /
- Generovanie náhodných hodnôt
- **Podmienkové príkazy** (**if-else**, resp. **if**)
  - spôsob ako zabezpečiť, že sa príkaz vykoná len za určitých podmienok

*... nové stavebné prvky našich programov*



*to be continued ...*

**ak nie sú otázky...**

**Ďakujem za pozornosť !**

