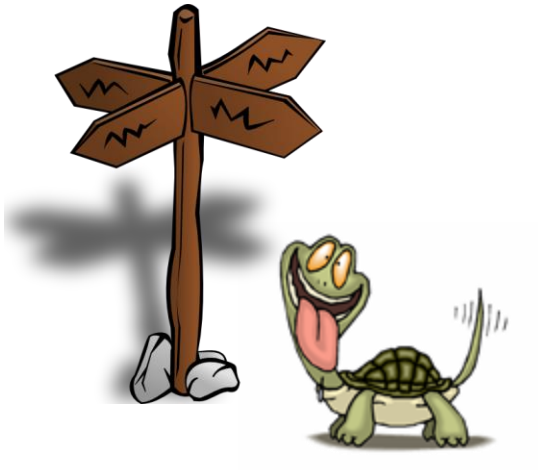
PAZ

2. prednáška



Premenné a podmienky

Naozajstné programovanie začína

Java

Na predošlej prednáške (1)

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

```
Turtle franklin = new Turtle();
```

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

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

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

```
public class SmartTurtle extends Turtle {


}
```

Na predošlej prednáške (2)

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

```
public void square(double size) {
                ... 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);
```

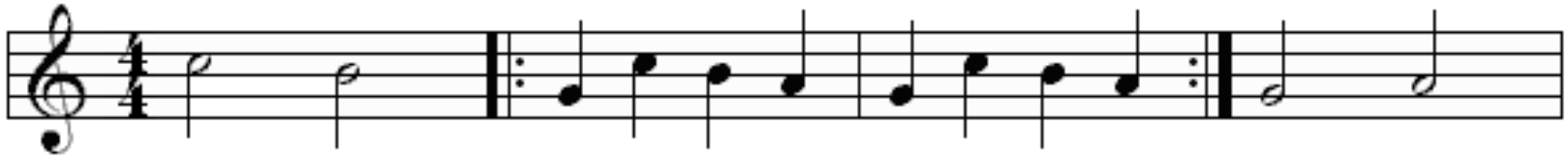
Na predošlej prednáške (3)

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

Koľko krát sa má niečo opakovať

```
for (int i=0; i<4; i++) {
   this.step(100);
   this.turn(90);
}
```

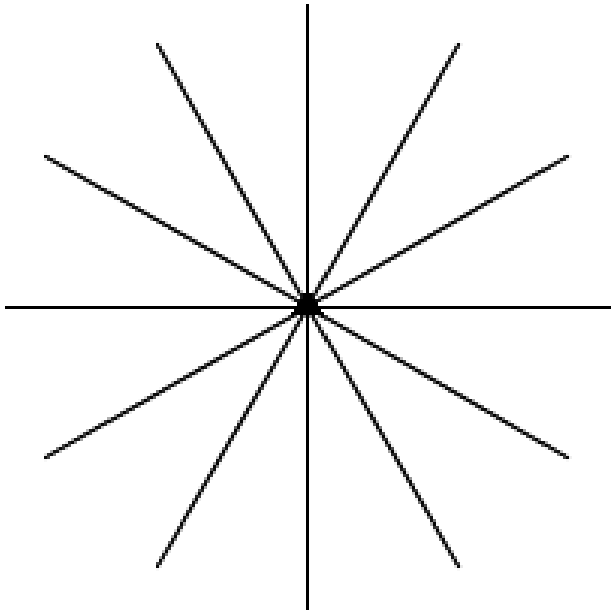
Príkazy, ktoré za majú opakovať



12-cípa hviezda

```
public void star(double size)
```

- návod:
  - 12 krát zopakuj:
    - sprav krok dĺžky *size*
    - sprav krok späť dĺžky *size*
    - otoč sa o *360 / 12 = 30* stupňov

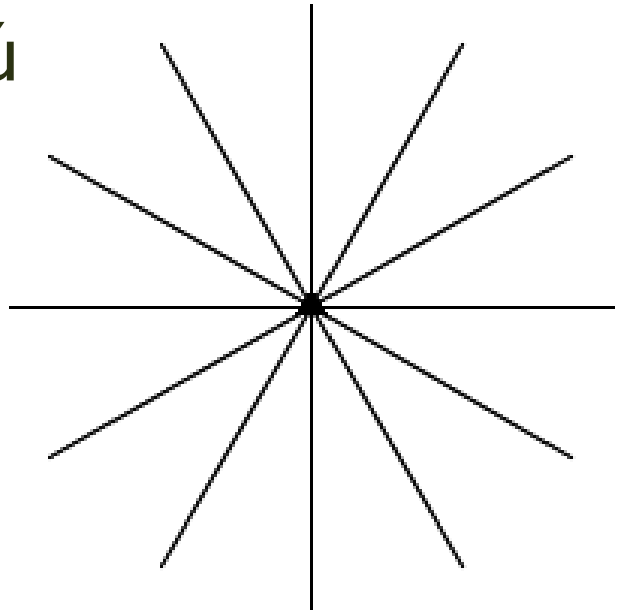


```
for (int i=0; i<12; i++) {
    this.step(size);
    this.step(-size);
    this.turn(30);
}
```

N-cípa hviezda (1)

- Chceme namaľovať jednoduchú

  n-cípu hviezdu



- **Parametre:**
  - `n` - počet lúčov
  - `rayLength` - dĺžka lúča
- **„Povolené hodnoty“** parametrov:
  - Počet lúčov - **celé číslo** (ako by vyzerala 3.8 cípa hviezda?)

`int`

  - Dĺžka lúča - **reálne číslo**

`double`

N-cípa hviezda (2)

```
public void nStar(int n, double rayLength)
```

- Návod:
  - `n` krát zopakuj:
    - Sprav krok dĺžky `rayLength`
    - Sprav krok späť dĺžky `rayLength`
    - Otoč sa o *360 / n* stupňov (n-tina plného uhla)

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

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

N-cípa hviezda (3)

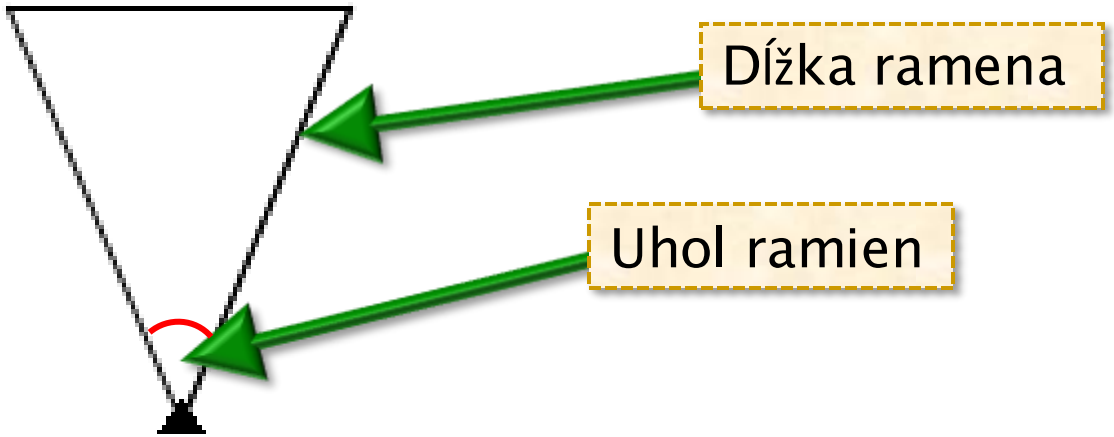
- Typy „povolených“ hodnôt:
  - `double` – reálne číslo (3.14, 2.71, 3.0, -14, -4.5)
  - `int` – celé číslo (4, 1000, -40, 90)

- Experiment:
  - funguje jednoduchá hviezda pre každé n?

Rovnoramenný trojuholník

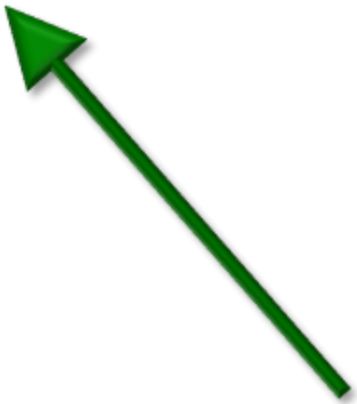
- Ako naučiť korytnačky namaľovať **rovnoramenný trojuholník** (isosceles triangle) so zadanou **dĺžkou ramena** a zadaným **uhlom**, ktorý zvierajú ramená?

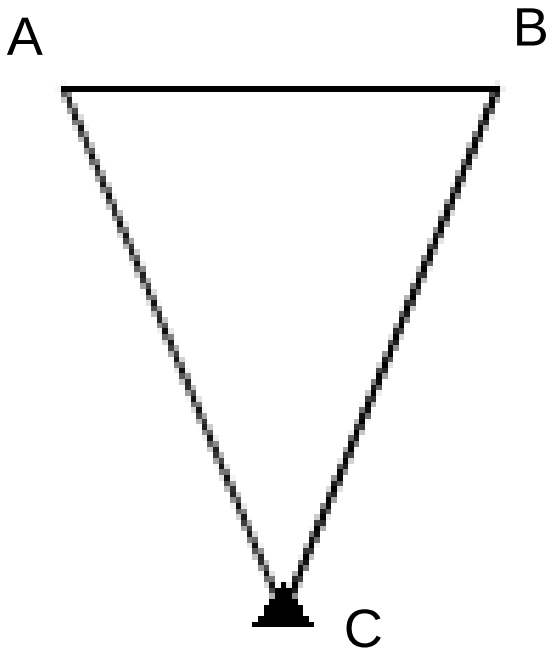
```
public void isosceles(double legLength, double angle)
```



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**

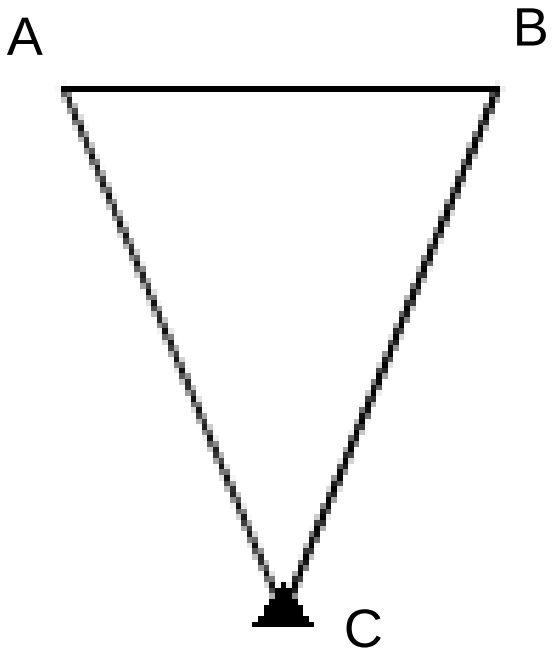




**Ako spravíme krok do bodu A,
ak nepoznáme jeho súradnice?**

Rovnoramenný trojuholník

1. Otoč sa o uhol/2 vľavo (v bode **C**)
2. Sprav krok dĺžky ramena (do bodu **A**)
   - som v bode A, nejako 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 pamätať?

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

Uložená hodnota

Typ hodnôt, ktoré je premenná schopná uložiť

Názov úložiska (premennej)

Čo potrebujeme vedieť?

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

Vytvorenie premennej

- Príkaz na vytvorenie premennej:

```
double xCoordinate;
```

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:

```
xCoordinate = 30.5;
```

Názov premennej, ktorej priraďujeme hodnotu

Hodnota, ktorú chceme do premennej uložiť (priradiť)

- Uložiť môžeme aj výsledok volania metódy:

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

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

dva v jednom

- Vytvorenie premennej spolu s inicializáciou (prvotným nastavením hodnoty)

```
double xCoordinate = 30.5;
```

Skratená verzia pre:

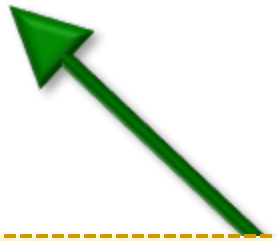
```
double xCoordinate;
xCoordinate = 30.5;
```

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

Čítanie hodnoty premennej

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

```
double stepLength = 30;
turtle.step(stepLength);
```



Korytnačka sa posunie o toľko, aká hodnota je **aktuálne** uložená v premennej `stepLength`, 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 isosceles(double legLength, double angle) {
   this.turn(-angle/2);
   this.step(legLength);
   double xCoord = this.getX();
   double yCoord = this.getY();
   this.step(-legLength);
   this.turn(angle);
   this.step(legLength);
   this.moveTo(xCoord, yCoord);
}
```

Do premenných `xCoord` a `yCoord` sme uložili súradnice aktuálnej pozície korytnačky (bodu A)

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

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 (x1, y1) do (x2, y2)

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

O svete napravo od =

- V príkaze priradenia napravo od = môžeme písať napr. 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 = x + 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 priraďujeme 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:

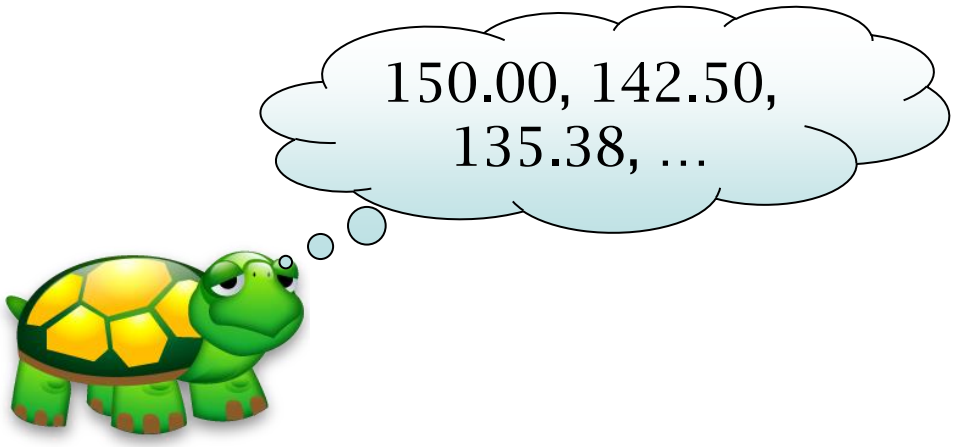
```
double posun =
      3 * this.distanceTo(100, 200);
```

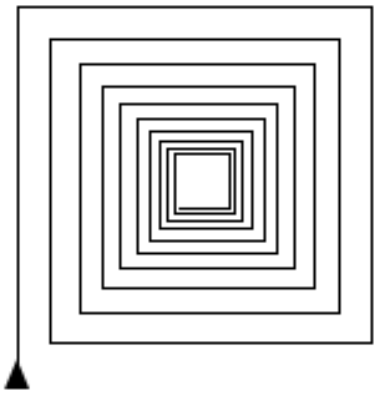
Špirála

- Naučme korytnačku metódu:

```
public void squareSpiral(int lineCount)
```

  - nakreslí **štvorcovú špirálu** so zadaným počtom strán
  - prvá strana (čiara) má dĺžku 150
  - každým krokom sa dĺžka strany (kroku) zníži o 5%





O nefungujúcej hviezde ...

- Prečo nefunguje jednoduchá hviezda pre každú hodnotu n?
- Odpoveď: aj **hodnoty majú svoj typ** ...
  - 360 je celé číslo (`int`)
  - 360.0 je reálne číslo (`double`)
  - matematická 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 » double*
  - *int + int » 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 xCoord = this.getX();
  this.step(10);
  this.turn(10);
  this.setX(xCoord + 8);
}
```

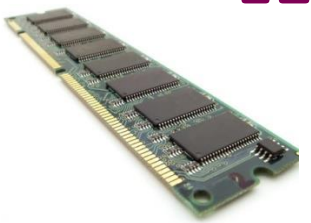
Premenná `xCoord` sa vytvorí a zanikne celkom 20 krát

Zátvorka, na ktorej premenná `xCoord` 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)
- **Reálne čísla:**
  - `double` – lepšia presnosť, zaberá viac pamäte
  - `float` – menšia presnosť, zaberá menej pamäte

rôzne typy = rôzne **rozsahy** povolených hodnôt a zabratá pamäť



primitívny = jednoduchý

Náhodou o náhodách ...

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

```
Math.random()
```

$\langle 0, 1)$

- 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

- Náhodné číslo z intervalu $\langle a, b)$:

```
a + Math.random()*(b-a)
```

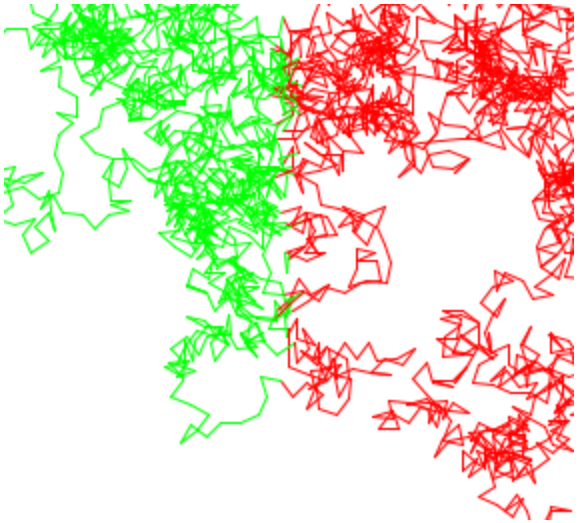
Zbesilá korytnačka

- Naprogramujme spomalený náhodný pohyb korytnačky ...

```
public void randomWalk(int stepCount) {
      for (int i=0; i<stepCount; 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:
  - *(vyraz1)* && *(vyraz2)* ... platí *vyraz1* a *vyraz2* $\wedge$
  - *(vyraz1)* || *(vyraz2)* ... platí *vyraz1* alebo *vyraz2* $\vee$
  - *!(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 ďalej 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 **ťažko 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 !