



# *1. prednáška (21.9.2011)*

## *Úvod do Javy a JPAZu*

*Náš prvý program...*





# Prečo JAVA ?

- Java je **moderný** „mainstreamový“ programovací jazyk
- Java je **objektovo orientovaná**
- Java: *write once, run everywhere*
- Java programátori sú žiadaní
- Java sa neučí na stredných školách
- Java je „**C**“-čkoidný programovací jazyk



**JAVA is everywhere**



# Java včera, dnes a zajtra

- vznikla v rokoch **1991-1995**
  - James Gosling v Sun Microsystems
- dnes 3 „vetvy“ Javy:
  - **Java ME** - pre mobilné zariadenia
  - **Java SE** - pre „bežné“ použitie (tu sme aj my)
  - **Java EE** - pre podnikové a biznis aplikácie
- <http://www.oracle.com/technetwork/java/>
- aktuálna verzia: Java 7
- **Android** - postavený na Jave



**ORACLE®**





# Vývojové prostredie Eclipse

- programovať v Jave sa dá aj v Notepade, ale ...
- **Eclipse** (od IBM, dnes free SW)
  - je moderné **vývojové prostredie** nielen pre Javu (PHP, C, Python, Perl, Cobol, ...)
  - beží vo všetkých hlavných OS (Windows, Linux, ...)
  - je konkurentom **NetBeans** (študentský projekt z Matfyzu na UK v Prahe)
  - má obrovskú **podporu** a kopy dostupných **pluginov**





# Základné koncepty Eclipse

- **Workspace** (pracovný priestor)
  - miesto, kde vytvárame **projekty**
  - adresár na disku
- **Project** (projekt)
  - “kopa súborov, ktoré **patria k sebe**“
  - Java Project = „kopa“ tried (Classes), ktoré patria k sebe
  - podadresár vo Workspace
- Demo ...



# Prvý projekt a prvá trieda

- programovanie v Jave = vytváranie tried (Class)
- demo vytvorenia spustiteľnej triedy v Eclipse ...

- typická „spúšťacia“ trieda:

```
public class Spustac {  
    public static void main(String[] args) {  
  
    }  
}
```

Hovoríme  
„metóda main“

Priestor pre naše príkazy,  
ktoré sa postupne vykonajú  
po spustení triedy **v takom  
poradí**, v akom sú zapísané.



# Čo máme na disku

- súbory vznikajú v podadresároch projektu
- súbory s príponou *java*
  - zdrojový kód jednotlivých tried
  - treba posielat' pri riešení domácich úloh
- súbory s príponou *class*
  - vzniká pri kompilácii a tá sa robí pri každom uložení java súboru
  - „skompilovaný“ zdrojový kód
  - druh javackého „exe“ súboru





# JPAZ2 framework

- JPAZ2 framework

- umožní nám vidieť objekty
- umožní nám interakciu s objektami
- umožní nám „korytnačiu grafiku“
- v knižnici (súbore) `jpaz2.jar`



- ak ho chceme používať, musíme:

1. pripojiť `jpaz2.jar` k projektu (demo ...)
2. do prvého riadku „každej“ našej triedy napísať:

```
import sk.upjs.jpaz2.*;
```





# Prvá trieda s JPAZom

- vyskúšajme príkaz (v metóde „main“):

```
WinPane plocha = new WinPane ();
```

Vytvoríme „komunikačnú“ premennú s názvom **plocha** - cez ňu potom dokážeme komunikovať s objektom, s ktorým bola prepojená cez =

Vytvoríme nový objekt triedy *WinPane* (objekt sa nám zobrazí po spustení „spúšťača“)



# Object Inspector

- pridajme ďalšie príkazy:

```
ObjectInspector oi = new ObjectInspector();  
oi.inspect(plocha);
```

Cez premennú **oi** povieme *ObjectInspector*-u, že má špehovať objekt, s ktorým komunikujeme cez premennú **plocha**

Podobne ako objekt triedy *WinPane* vytvoríme aj objekt triedy *ObjectInspector* a premennú **oi**, cez ktorú s ním budeme komunikovať



# Object Inspector a objekty

- **Object Inspector**
  - slúži na „špehovanie“ objektov
- cez Object Inspector vidíme, že objekty majú:
  - **vlastnosti** (properties)
  - **metódy** (methods)
- **vlastnosti** ukazujú „stav“ objektu a niektoré ide dokonca meniť (ich zmenou sa nejakým spôsobom zmení objekt)





# Prvá korytnačka

- pridajme:

Vytvoríme objekt triedy *Turtle*, s ktorým budeme komunikovať cez premennú **jozko** („meno korytnačky“)

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

```
plocha.add(jozko);
```

```
oi.inspect(jozko);
```

Pridáme vytvorenú korytnačku do plochy

Povieme vytvorenému Object Inspectoru, aby „špehoval“ vytvorenú korytnačku

Zjednodušený  
tvar  
korytnačiek





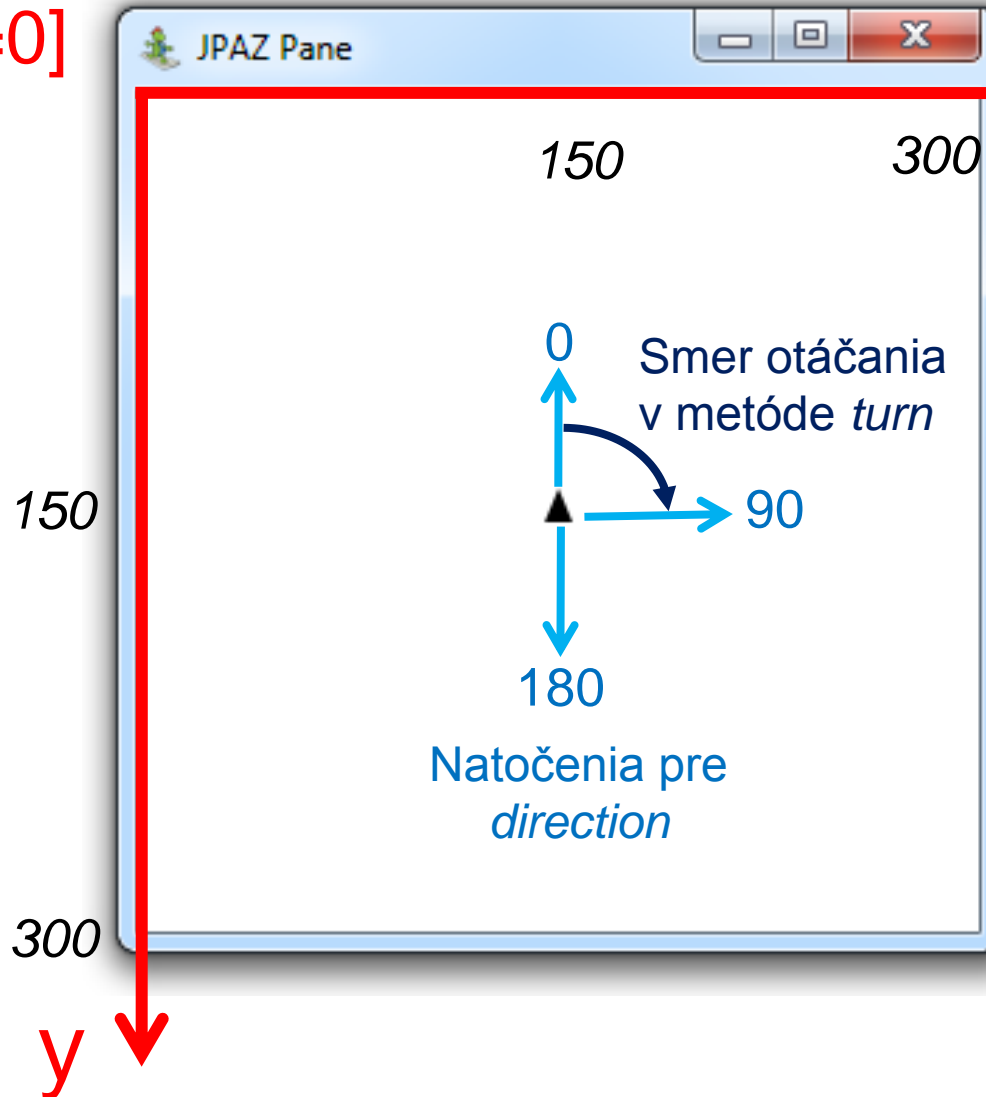
# Pozorovanie korytnačky v OI

- korytnačka „**žije v ploche**“ (objekt triedy *WinPane*) a miesto jej pobytu je určené súradnicami (X, Y)
- **súradnica** (0, 0) je v ľavom hornom rohu
- x-ová súradnica rastie zľava doprava
- y-ová súradnica rastie zhora nadol
- korytnačke ide **menit'**:
  - farbu (*penColor*)
  - natočenie (*direction*) - v uhloch, rastie v smere pohybu hodinových ručičiek, smer 0 je nahor



# Pozícia a natočenie

$[x=0, y=0]$



***turn*** = relatívne otočenie o zadaný uhol v smere hodinových ručičiek

***setDirection*** = absolútne natočenie zadaným smerom



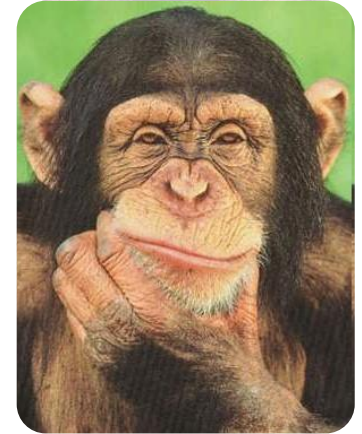
# Pozorovanie metód cez OI

- **metódy sú príkazy pre objekty:**
  - *center* - korytnačka sa presunie do stredu plochy
  - *step* - korytnačka sa posunie o zadanú dĺžku
  - *turn* - korytnačka sa otočí o zadaný uhol
- cez metódy sa „**rozprávame**“ s objektmi
- niektoré metódy majú **parametre** (parameters), ktorými sa bližšie upresňuje, ako sa má príkaz vykonať
- niektoré metódy **odpovedajú** hodnotou (result)
- niektoré metódy **sú podobné vlastnostiam** (vlastnosť x a metódy setX a getX)



# Zmysluplné „klikanie“ príkazov

- Nakreslenie trojuholníka so stranou 100:
  - *spusti metódu step s parametrom 100*
  - *spusti metódu turn s parametrom 120*
  - *spusti metódu step s parametrom 100*
  - *spusti metódu turn s parametrom 120*
  - *spusti metódu step s parametrom 100*
  - *spusti metódu turn s parametrom 120 (ak chceme, aby korytnačka bola nasmerovaná tak, ako bola na začiatku)*
  
- A čo štvorec?







# Spät' k programovaniu

- Už vieme:
  - vytvorit' objekty napísaním „magických“ príkazov
  - „hrať“ sa s objektami cez **Object Inspector**
  - čo sú **vlastnosti** a čo **metódy**
  - poznáme čo robia niektoré metódy korytnačky
  - základ **korytnačej grafiky** a JPAZu:
    - plocha (*WinPane*) je domov pre korytnačky (*Turtle*)
- D.Ú: preskúmajte metódy plochy (*WinPane*)



# Vytvorenie objektu v Java

- Ako teda vytvárame objekty (intuícia)?

```
Trieda komunikator = new Trieda ();
```

- Čo sa stane?
  1. Vo **svete objektov vznikne** („narodí sa“) objekt triedy *Trieda*
  2. V programe (vo svete nášho Java programu) **vznikne premenná komunikator** a nastaví sa tak, aby sa cez ňu dalo **komunikovať s objektom**, ktorý vznikol v bode 1
- Odborná terminológia: premenná **referencuje** objekt



# Program vs. „svet objektov“

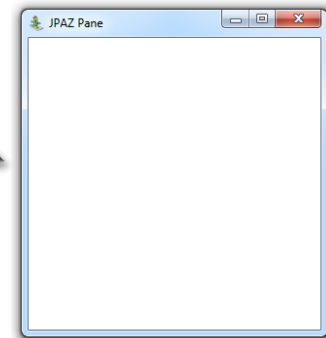
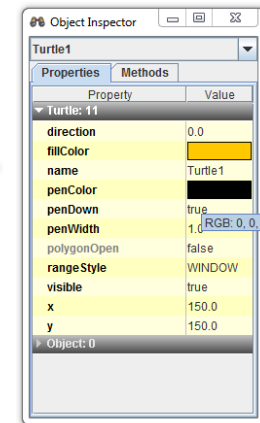
„náš program“

oi

jozko

plocha

„svet objektov“



Cez „komunikačné“ premenné v programe komunikujeme s objektmi, ktoré sme vytvorili cez **new** vo „svete objektov“.



# Experiment

- Čo sa stane, ak dopíšeme ďalšie príkazy?

```
jozko.center();
```

```
jozko.step(100);
```



**Výsledok:**

**korytnačku vieme ovládať aj z programu !!!**



# Spúšťanie metód „z Javy“

```
jozko.step(100);
```

## KTO

Meno premennej, cez ktorú komunikujeme s objektom (ktorá obsluhuje objekt, či ktorá **referencuje** objekt), ktorého metódu chceme vykonať.

## ČO

Meno metódy, ktorú chceme vykonať.

## UPRESNENIE

Parametre metódy medzi zátvorkami. Ak je parametrov viac, oddeľujeme ich čiarkou.

```
korytnacka.stamp();
korytnacka.moveTo(30, 50);
```



# Spúšťanie metód „z Javy“

```
jozko.step(100);
```

- odborná terminológia: voláme metódu *step* objektu referencovaného premennou *jozko*
- na zapamätanie (najčastejšie chyby!):
  - pred a za **bodkou** nesmú byť **medzery**
  - za každým príkazom v Jave sa píše **bodkočiarka** (až na jednu výnimku - bude neskôr)
  - pravidlá **slušného formátovania** (viac na teoretickom cvičení) alebo **CTRL+SHIFT+F** v Eclipse
  - v Jave na **veľkosti písmen** záleží: „Ahoj nie je aHoj“





# Programujeme prvé programy!

- budeme písať príkazy, ktoré namaľujú:
  - štvorec, obdĺžnik, trojuholník v kombinácií so zmenami farby ...
- novinky:
  - *JPAZUtilities.delay(100)* - zastaví vykonávanie programu na 100 ms
  - *Color.RED*, *Color.BLACK*, ... - hodnoty farieb pre korytnačí príkaz *setPenColor*
  - zápis reálnych čísel (*double* v OI): 2.3, 4.82, ...
  - *int* v OI: len celé čísla 20, -60, 130, ...



# V dvojici je život krajší ...

- Náš cieľ: pridať do plochy ďalšiu korytnačku a nechať ju „špehovať“ Object Inspectorom ...

```
Turtle katka = new Turtle();  
plocha.add(katka);  
oi.inspect(katka);
```

- Pozorovanie:
  - Obe korytnačky majú **rovnaké** metódy a vlastnosti
  - Aktuálne **hodnoty vlastností** sú rôzne

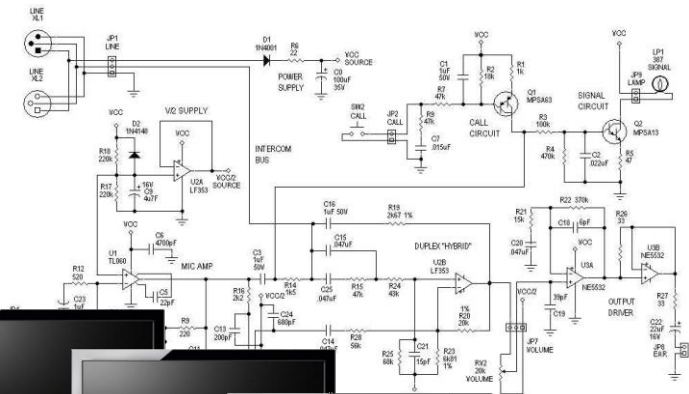






# Čo je to trieda?

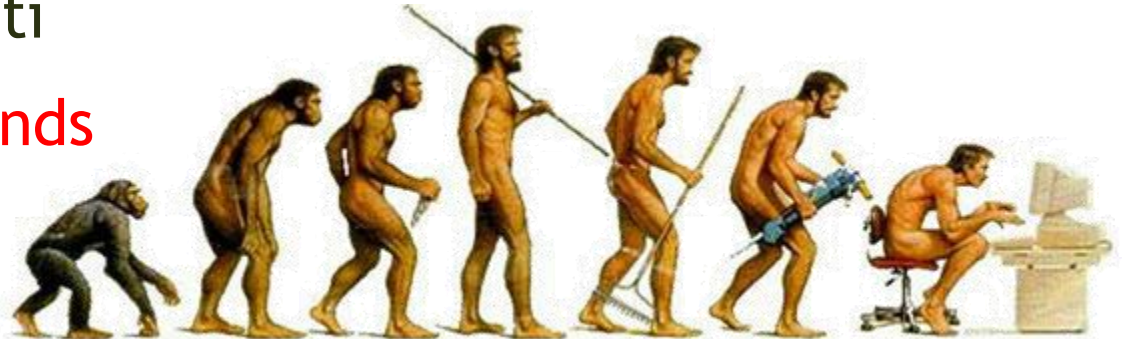
- Trieda je **šablóna** (vzor), ktorý **predpisuje** aké **metódy** má trieda a čo sa stane pri ich spustení.
- Trieda je „genetická informácia“, ktorú dostáva objekt danej triedy pri svojom „narodení“ (vytvorení vo „svete objektov“)





# Evolúcia vo svete JPAZ (1)

- Tvorstvo prežije iba ak sa **učí nové veci** ...
  - Ako vytvoriť („vyšľachtit'“) vylepšený **nový druh** korytnačiek, ktorý bude chytřejší (napr. bude poznať viac metód)?
- Naučiť nové znamená:
  - poznať všetko staré (nezabudnúť, čo sa už vedelo) a navyše poznať aj nové veci
  - rozšíriť existujúce schopnosti (z triedy *Turtle*) o nové schopnosti
  - rozšíriť = **extends**





# Evolúcia vo svete JPAZ (2)

- Postup (demo):

1. Vytvoríme novú triedu MojaTurtle (cez Eclipse), ktorá vylepšuje (rozširuje - extends) triedu Turtle (superclass v Eclipse)

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

Priestor pre pridanie („naučenie“) nových príkazov (metód)

„Vytvárame šablónu pre objekty triedy **MojaTurtle** rozšírením šablóny (pridanie nových vecí) pre objekty triedy **Turtle**.“



# Evolúcia vo svete JPAZ (3)

- Postup (demo):

2. Doplníme novo naučený príkaz ...

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

„Magické  
slovička“  
(vysvetlíme neskôr)

Názov metódy

```
public void trojuholnik() {
```

← Priestor pre už naučené príkazy,  
ktoré namaľujú trojuholník

```
}
```

```
}
```



# Evolúcia vo svete JPAZ (4)

- 3. Doplníme príkazy:

```
public class MojaTurtle extends Turtle {
    public void trojuholnik() {
        this.step(100);
        this.turn(120);
        this.step(100);
        this.turn(120);
        this.step(100);
        this.turn(120);
    }
}
```

```
this = „ja“
„ja“ spravím step(100)
„ja“ spravím turn(120)
```

**this** = ja, objekt triedy MojaTurtle, ktorý som bol požiadaný vykonať metódu trojuholnik.



# Evolúcia vo svete JPAZ (5)

- Čo sme spravili?
  - vytvorili sme novú triedu „chytřejších“ korytnačiek s menom MojaTurtle, ktoré poznajú navyše metódu *trojuholnik*
- Zmeňme

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

na

```
MojaTurtle jozko = new MojaTurtle();
```
- Pozorujme, čo sa stane v OI ...



# Evolúcia vo svete JPAZ (6)

- pozorovanie z OI:
  - každý objekt **má len** také metódy, aké mu **predpisuje** príslušnosť k triede
- v Java programe môžeme písať:

```
jozko.trojuholnik();
```

ale nie

```
katka.trojuholnik();
```

lebo cez premennú *katka* vieme komunikovať len s korytnačkami triedy *Turtle* - tie nepoznajú metódu *trojuholnik*



# Nerobme veci dvakrát ...

- naučme objekty triedy *MojaTurtle* ďalšiu metódu so záhadným kódom:

```
public void zahada () {  
    this.trojuholnik();  
    this.turn(120);  
    this.trojuholnik();  
    this.turn(120);  
    this.trojuholnik();  
    this.turn(120);  
}
```

Môžeme volať aj tie metódy, ktoré sme objekty triedy *MojaTurtle* doučili.





# Metódy s parametrom (1)

- aj naše doučené metódy môžu mať **parametre** ...
- parameter zastupuje hodnotu, s ktorou sa metóda volá...

```
public void trojuholnik(double strana) {  
    this.step(strana);  
    this.turn(120);  
    this.step(strana);  
    this.turn(120);  
    this.step(strana);  
    this.turn(120);  
}
```

Meno parametra.

„Magické slovíčko“  
hovoriace, že  
parameter **musí byť**  
**číslo**.



# Metódy s parametrom (2)

- metóda môže mať aj viac parametrov:

```
public void obdlznik(double sirka, double vyska) {  
    ... príkazy na namaľovanie obdĺžníka ...  
}
```

- jednotlivé parametre **oddeľujeme čiarkou**
- parameter je vždy popísaný dvojicou:
  - „magické slovíčko“ definujúce **povolené hodnoty**
    - **double** – povolená hodnota je ľubovoľné reálne číslo
  - **názov** parametra, pod ktorým je hodnota parametra dostupná v metóde



# Opakovanie je ...

- ... matkou múdrosti a základ programovania

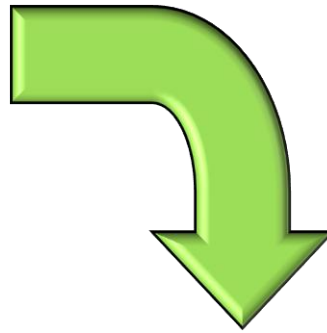
```
public void trojuholnik(double strana) {  
    this.step(strana);  
    this.turn(120);  
    this.step(strana);  
    this.turn(120);  
    this.step(strana);  
    this.turn(120);  
}
```

**3x** úplne rovnaká  
postupnosť príkazov



# Trojuholník ...

```
public void trojuholnik(double strana) {  
    this.step(strana);  
    this.turn(120);  
    this.step(strana);  
    this.turn(120);  
    this.step(strana);  
    this.turn(120);  
}
```



```
public void trojuholnik(double strana) {  
    for (int i=0; i<3; i++) {  
        this.step(strana);  
        this.turn(120);  
    }  
}
```



# Ako opakovať?

- „magická formula“ na opakovanie skupiny príkazov:

Koľko krát sa má niečo opakovať

```
for (int i=0; i<3; i++) {  
    this.step(100);  
    this.turn(120);  
}
```

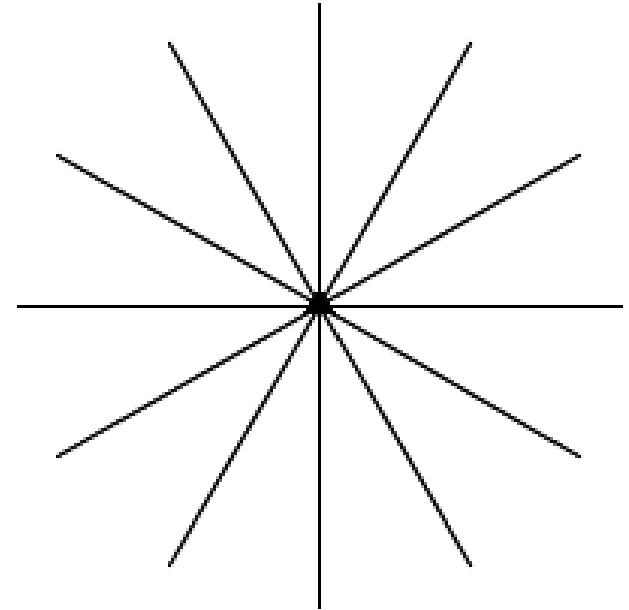
Príkazy, ktoré sa majú opakovať





# Vločka (1)

- chceme namaľovať vločku s 12-timi ramenami
- parameter:
  - *dlzkaLuca* - dĺžka lúča
- návod:
  - 12 krát zopakuj:
    - sprav krok dĺžky *dlzkaLuca*
    - sprav krok späť dĺžky *dlzkaLuca*
    - otoč sa o  $360 / 12 = 30$  stupňov





# Vločka (2)

```
public void vločka(double dlzkaLuca)
```

- návod:

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

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



*to be continued ...*

**ak nie sú otázky...**

**Ďakujem za pozornosť !**

