1. prednáška (17.9.2018)

Úvod do Javy a JPAZu

Náš prvý program…

Prečo Java?

Ratings (%)

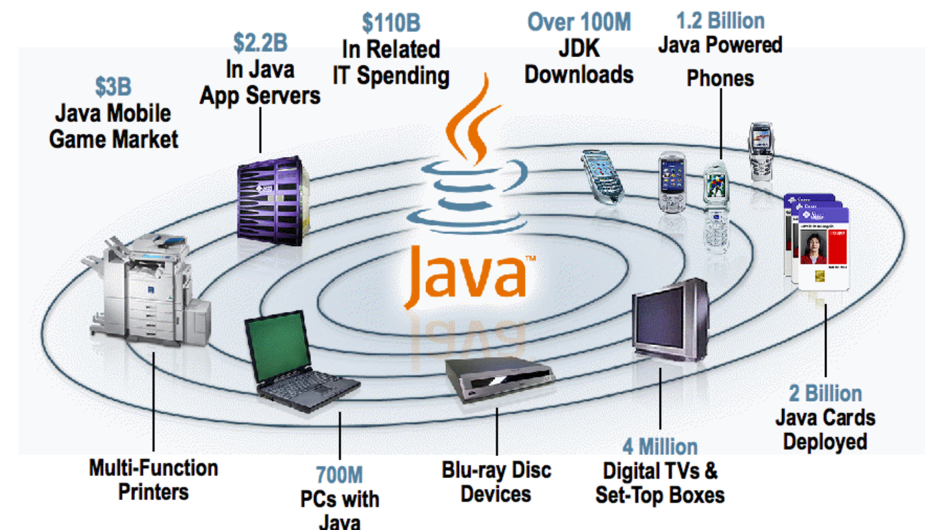
30
25
20
15
10
5
0

2002 2004 2006 2008 2010 2012 2014 2016 2018

Java
C
Python
C++
Visual Basic .NET
C#
PHP
JavaScript
SQL
Objective-C

Prečo JAVA?

- **moderný** „mainstreamový“ programovací jazyk
- **objektovo orientovaná**
- *write once, run everywhere*
- **„C“-čkoidný** programovací jazyk
- neučí sa na stredných školách?



Java včera, dnes a zajtra

- vznikla v rokoch **1991-1995**

  - James Gosling v Sun Microsystems
- dnes 4 „vetvy“ Javy:
  - **Java Card** – pre „smart“ karty (SIM, ...)
  - **Java ME** – pre mobilné zariadenia
  - **Java SE** – pre „bežné“ použitie (tu sme aj my)
  - **Java EE** – pre podnikové a biznis aplikácie ORACLE
- http://www.oracle.com/technetwork/java/
- aktuálna verzia: Java 10/Java 8 LTS
- **Android** – postavený na Jave

Vývojové prostredie Eclipse

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

Základné koncepty Eclipse

- **Workspace** (pracovný priestor)
  - miesto, kde vytvárame projekty
  - adresár na disku
- **Project** (projekt)
  - „kopa súborov, ktoré patria k sebe"
  - podadresár vo Workspace
- „Mavenovský" projekt
  - projekt s (nielen) **preddefinovanou štruktúrou**
  - bonus: prenositeľnosť medzi IDE, zjednodušené manažovanie a „buildovanie" projektu

Maven

- komplexný nástroj pre správu, riadenie a automatizáciou „buildov" aplikacií
- štandard vo svete Javy

- artefakt (artifact) – **základný prvok Mavenu -** niečo, čo je výsledkom projektu alebo je to použité projektom
- archetyp (archetype) – artefakt s predpripravenou šablónou projektu

JPAZ2 archetypy

- katalóg archetypov pre predmet PAZ1a:

  http://jpaz2.ics.upjs.sk/maven/archetype-catalog.xml
- **`jpaz2-archetype-novice`**
  - šablóna na týždne 1-3
- **`jpaz2-archetype-quickstart`**
  - šablóna na týždne 4-6
- **`jpaz2-archetype-launcher`**
  - šablóna na týždne 6-14
- **`jpaz2-archetype-theater`**
  - šablóna na záverečný projekt

Maven - artefakty

- identifikácia artefaktov:
  - **groupId** – jedinečná identifikácia skupiny artefaktov
    - pre nás: identifikácia autora + další info s bodkami
      príklad: `paz1a.janko.hrasko`
  - **artifactId** – identifikácia artefaktu v rámci skupiny
    - pre nás: názov projektu
  - ~~**version** – verzia artefaktu~~
  - ~~**packaging** – typ výstupu~~

Group Id:

Artifact Id:

Version: 0.0.1-SNAPSHOT

Prvý projekt a prvá trieda

- programovanie v Jave = vytváranie tried (Class)
- demo vytvorenia spustiteľnej triedy v Eclipse ...
- typická „spúšťacia“ trieda:

```
public class Launcher {
    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 posielať pri riešení domácich úloh

- zvyšok nás nateraz nezajíma...

JPAZ2 framework

- JPAZ2 framework
  - umožní nám vidieť objekty
  - umožní nám interakciu s objektami
  - umožní nám „korytnačiu grafiku“

Prvá trieda s JPAZom

- vyskúšajme príkaz (v metóde „main“):

```
AnimatedWinPane sandbox = new AnimatedWinPane();
```

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

Vytvoríme nový objekt triedy `AnimatedWinPane` (objekt sa nám zobrazí po spustení „spúšťača“)

Object Inspector

- pridajme ďalšie príkazy:

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

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

Podobne ako objekt triedy `AnimatedWinPane` 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 nejako zmení objekt)

Prvá korytnačka

- pridajme:

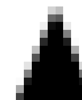
Vytvoríme objekt triedy `Turtle`, s ktorým budeme komunikovať cez premennú **franklin** („meno korytnačky")

```
Turtle franklin = new Turtle();
sandbox.add(franklin);
oi.inspect(franklin);
```

Pridáme vytvorenú korytnačku do pieskoviska (**sandbox**)

Povieme vytvorenému Object Inspectoru, aby „špehoval" vytvorenú korytnačku

Zjednodušený tvar korytnačiek



Pozorovanie korytnačky v OI

- korytnačka **„žije v pieskovisku/ploche“** (objekt triedy `AnimatedWinPane`) 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 **meniť**:
  - farbu (`penColor`)
  - natočenie (`direction`) - v uhloch, rastie v smere pohybu hodinových ručičiek, smer 0 je nahor

Pozícia a natočenie

turn = relatívne otočenie o zadaný uhol v smere hodinových ručičiek

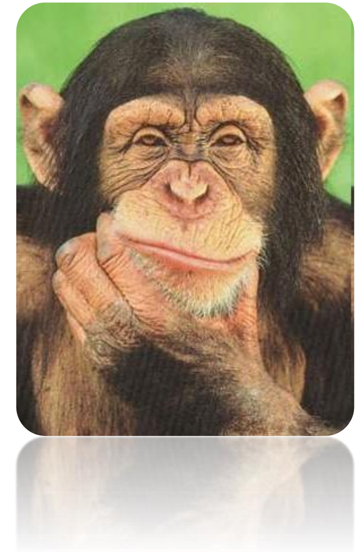
setDirection = absolútne natočenie zadaným smerom

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 štvorca (square) so stranou 100:
  - *spusti metódu step s parametrom 100*
  - *spusti metódu turn s parametrom 90*
  - *spusti metódu step s parametrom 100*
  - *spusti metódu turn s parametrom 90*
  - *spusti metódu step s parametrom 100*
  - *spusti metódu turn s parametrom 90*
  - *spusti metódu step s parametrom 100*
  - *spusti metódu turn s parametrom 90 (ak chceme, aby korytnačka bola nasmerovaná tak, ako bola na začiatku)*



- rovnostranný trojuholník (equilateral triangle)?

Späť k programovaniu

- už vieme:
  - **vytvoriť** objekty napísaním „magických" príkazov
  - „hrať" sa s objektami cez **ObjectInspector**
  - čo sú **vlastnosti** a čo **metódy**
  - poznáme čo robia niektoré metódy korytnačky
  - základ **korytnačej grafiky** a JPAZu:
    - pieskovisko (`AnimatedWinPane`) je domov pre korytnačky (`Turtle`)

Vytvorenie objektu v Jave

- 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“

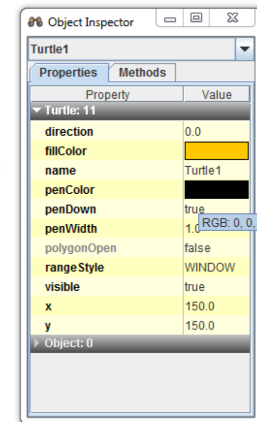
„náš program“

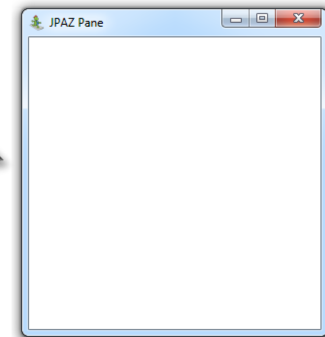
„svet objektov“

oi

franklin

sandbox





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

Experiment

- čo sa stane, ak dopíšeme ďalšie príkazy?

```
franklin.step(100);
```

Výsledok:

korytnačku vieme ovládať aj z programu !!!

Spúšťanie metód „z Javy“

```
franklin.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“

```
franklin.step(100);
```

- odborná terminológia: voláme metódu `step` objektu referencovaného premennou `franklin`

- 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“

Programujme prvé programy!

- budeme písať príkazy, ktoré namaľujú:
  - štvorec, obdĺžnik, trojuholník v kombinácii 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,` ...

Vo dvojici je život krajší...

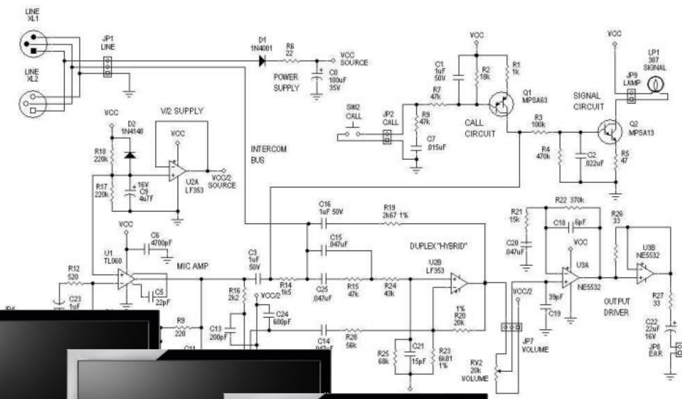
- náš cieľ: pridať do plochy ďalšiu korytnačku a nechať ju „špehovať" Object Inspectorom...

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

- pozorovanie:
  - obe korytnačky majú rovnaké metódy a vlastnosti
  - aktuálne hodnoty vlastností sú rôzne

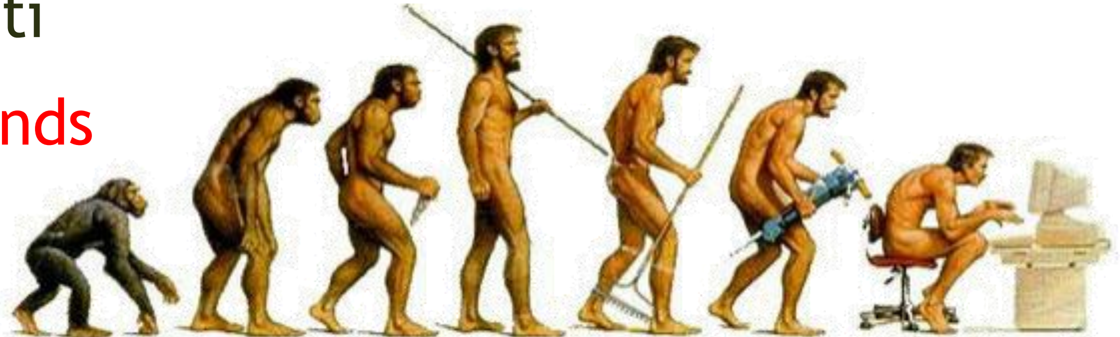
Čo je to trieda?

- **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 („vyšľachtiť“) vylepšený **nový druh** korytnačiek, ktorý bude chytrejší (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 `SmartTurtle` (cez Eclipse), ktorá vylepšuje (rozširuje - extends) triedu `Turtle` (superclass v Eclipse)

```
public class SmartTurtle extends Turtle {

}
```

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

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

Evolúcia vo svete JPAZ (3)

- Postup (demo):
  2. Doplníme novo naučený príkaz ...

```
public class SmartTurtle extends Turtle {

    public void triangle() {

    }

}
```

„Magické slovíčka“ (vysvetlíme neskôr)

Názov metódy

Priestor pre už naučené príkazy, ktoré namaľujú trojuholník

Evolúcia vo svete JPAZ (4)

- 3. Doplníme príkazy:

```
public class SmartTurtle extends Turtle {
    public void triangle() {
        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 `SmartTurtle`, ktorý som bol požiadaný vykonať metódu `triangle`.

Evolúcia vo svete JPAZ (5)

- Čo sme spravili?
  - vytvorili sme novú triedu „chytrejších" korytnačiek s menom `SmartTurtle`, ktoré poznajú navyše metódu *triangle*
- Zmeňme

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

na

```
SmartTurtle franklin = new SmartTurtle();
```

- 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:

```
franklin.triangle();
```

ale nie

```
cecil.triangle();
```

lebo cez premennú `cecil` vieme komunikovať len s korytnačkami triedy `Turtle` - tie nepoznajú metódu `triangle`

Nerobme veci dvakrát ...

- naučme objekty triedy `SmartTurtle` ďalšiu metódu so záhadným kódom:

```
public void mystery() {
    this.triangle();
    this.turn(120);
    this.triangle();
    this.turn(120);
    this.triangle();
    this.turn(120);
}
```

Môžeme volať aj tie metódy, ktoré sme objekty triedy `SmartTurtle` doučili.

Metódy s parametrom (1)

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

```
public void triangle(double size) {
    this.step(size);
    this.turn(120);
    this.step(size);
    this.turn(120);
    this.step(size);
    this.turn(120);
}
```

Meno parametra.

„Magické slovíčko" hovoriace, že parameter musí byť číslo.

Metódy s parametrom (2)

- metóda môže mať aj viac parametrov:

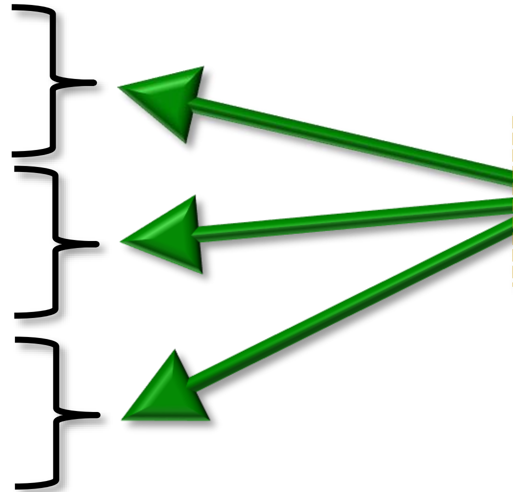
```
public void rectangle(double width, double height) {
   ... príkazy na nakreslenie obdlžníka ...
}
```

- jednotlivé parametre **oddeľujeme čiarkou**
- **paremeter** 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

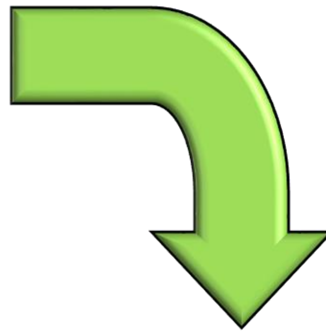
```
public void triangle(double side) {
    this.step(side);
    this.turn(120);
    this.step(side);
    this.turn(120);
    this.step(side);
    this.turn(120);
}
```



3x úplne rovnaká postupnosť príkazov

Trojuholník ...

```
public void triangle(double size) {
   this.step(size);
   this.turn(120);
   this.step(size);
   this.turn(120);
   this.step(size);
   this.turn(120);
}
```



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

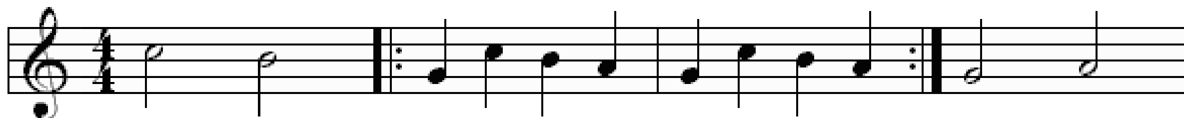
Ako opakovať?

- „magická formula" na opakovanie skupiny príkazov:

Koľko krát sa má niečo opakovať

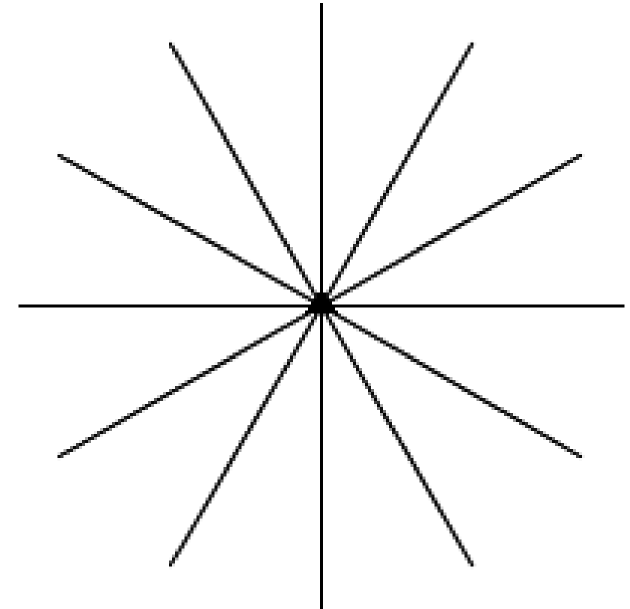
```
for (int i=0; i<3; i++) {
    this.step(100);
    this.turn(120);
}
```

Príkazy, ktoré za majú opakovať



Jednoduchá hviezda (1)

- chceme namaľovať hviezdu s 12-timi cípmi
- parameter:
  - *size* – dĺžka lúča
- návod:
  - 12 krát zopakuj:
    - sprav krok dĺžky *size*
    - sprav krok späť dĺžky *size*
    - otoč sa o *360 / 12* = *30* stupňov



Jednoduchá hviezda (2)

```
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);
}
```

Sumarizácia (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 {


}
```

Sumarizácia (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);
```

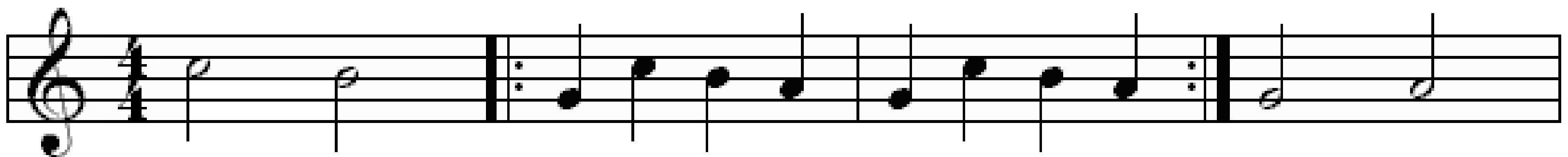
Sumarizácia (3)

- „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



to be continued …

ak nie sú otázky...

Ďakujem za pozornosť !