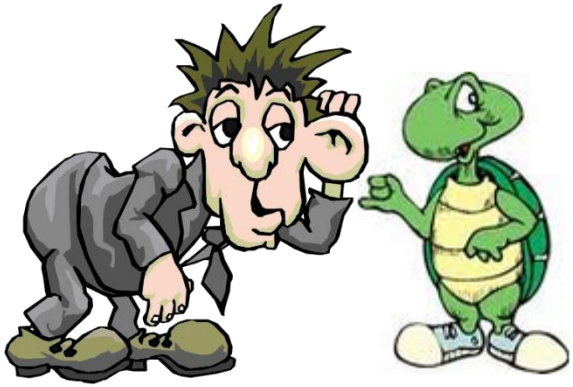
PAZ

3. Prednáška (5.10.2026)

Cykly, funkcie a referencie



Počúvajme, čo nám objekty hovoria

Java



Čo už vieme...

- **Vytvoriť objekt** nejakej triedy pomocou **new** a komunikovať s ním pomocou na to určenej „komunikačnej“ premennej
- **Vytvoriť vlastnú triedu** rozširujúcu triedu `Turtle` a popridávať do nej nové metódy
- Pracovať s **premennými primitívneho typu** na uloženie čísel (`int`, `byte`, `short`, `long`, `double`, `float`) a pravdivostnej hodnoty (`boolean`)
  - vieme zapisovať aritmetické a logické výrazy, rozumieme ich vyhodnocovaniu
- Poznáme **podmienkový príkaz** (`if-else`)

Zbesilá kockáčka (1)



- Štandardná náhodná pochôdzka:

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

- Chceme, aby sa korytnačka neotáčala akýmkoľvek smerom, ale iba **o násobok 90** (t.j. 0, 90, 180, 270, 360, ...)



Zbesilá kockáčka (2)

- Aké náhodné uhly otočenia chceme:
  - 0=0*90, 90=1*90, 180=2*90, 270=3*90, 360=4*90, ...

Potrebujeme generátor náhodných **celých** čísel - `Math.random()` dáva len reálne z intervalu <0, 1)

- Ako vygenerovať náhodné celé číslo medzi 0 až 3?

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

Do premennej na celé čísla nevieme uložiť reálne číslo!

Náhodné reálne číslo z <0, 4)



Pretypovanie

- Java umožňuje meniť hodnoty jedného typu na iný typ pomocou **pretypovania** hodnoty
- *Implicitné* („automatické“):
  - Java dokáže zmeniť celé číslo na reálne:

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

- *Explicitné* („musí chcieť programátor“)
  - Java na požiadanie dokáže zmeniť reálne číslo na celé **orezaním desatinnej časti**:

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

V `c` bude 10



Zbesilá kockáčka (3)

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

Na aký typ chceme zmeniť hodnotu (funguje aj s inými typmi ako **byte**, **long**, ...)

Akej hodnote chceme zmeniť typ

Pozor: Pri zmene z **double** na **int** sa hodnota oreže, nie zaokrúhli!

```
this.turn(angle);
```

Keďže `turn` čaká ako parameter **double**, tu sa udeje implicitné (automatické) pretypovanie z **int** na **double**.



Chceme viac matematiky!

- `Math.abs(number)` – absolútna hodnota čísla
- `Math.sqrt(number)` – odmocnina
- `Math.sin(angle)` – sínus uhla v radiánoch
- `Math.PI` – číslo π
- `Math.round(number)` – **zaokrúhlenie** reálneho čísla na celé

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

$$3\sqrt{2}$$



Štvorcová špirála (opäť)

- Naučme korytnačku metódu

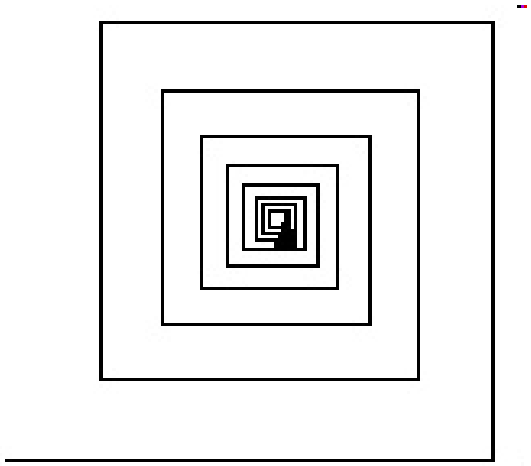
```
public void squareSpiral()
```

ktorá nakreslí **štvorcovú špirálu**. Počiatočný rozmer je 150 a každým krokom sa dĺžka kroku (strany) zníži o 10 %.

- Koľko krokov máme spraviť?

Možné riešenie:
Kreslíme, kým je krok „vidieť“ – kroky menšie ako 1 sa javia ako bod, t.j. nie je ich vidieť.

Koľko to je ale krokov?



Opakovanie riadené podmienkou

```
while (podmienka) {
    príkazy cyklu
}
```



- Kým je podmienka splnená, tak sa príkazy cyklu neustále opakujú ...
- Podmienka je nejaký logický výraz, ktorý sa testuje pred každým (opakovaným) vykonaním príkazov cyklu.

```
while (currentStepLength >= 1) {
    …
}
```

Keď je koniec v nedohľadne

- Riziko cyklov riadených podmienkou je to, že sa zmýlime a podmienka bude splnená stále ...
- ... stále splnená podmienka = nikdy nekončiaci (**nekonečný**) cyklus ...

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





For cyklus pod drobnohľadom

Vytvorí sa premenná `i` typu **`int`**, ktorá „žije“ len počas for-cyklu. Premenná `i` je inicializovaná na hodnotu 0.

Podmienka, ktorá **musí byť splnená**, aby sa príkazy cyklu vykonali.

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

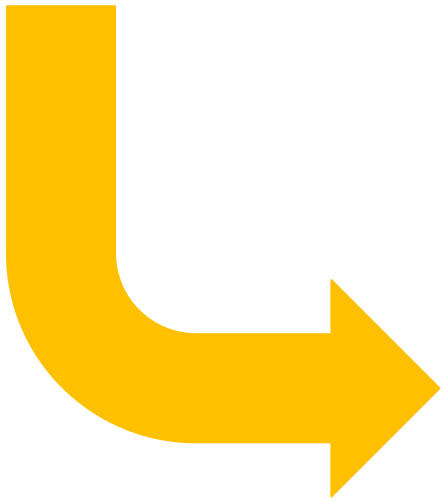
Premennú `i` zvykneme nazývať **riadiacou premennou cyklu.**

Čo sa má stať, **po každom vykonaní** príkazov cyklu (zvyčajne sa zvýši hodnota riadiacej premennej cyklu o 1)



For cyklus pod drobnohľadom

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



```
int i=0;
while (i<10){
  príkazy cyklu
  i++;
}
```

Pozor: toto nie je úplne presné!

Challenge: Prečo to nie je úplne presné? (Hint: prednáška 2)



Dôsledky

- Aktuálna hodnota premennej `i` sa **mení** pri každej iterácii (opakovaní) cyklu

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

- `i = 0, 1, 2, 3, 4, …, 9`
- Hodnotu premennej `i` vieme využiť v príkazoch cyklu (viac na cvičeniach)

Poznámka: s for-cyklami ide robiť aj iné programátorské „zverstvá“, ale my ich robiť nebudeme ...



Príkazy break a continue

- Príkazom **`break;`** vieme **okamžite ukončiť** vykonávanie celého cyklu
- Príkazom **`continue;`** vieme okamžite ukončiť vykonávanie aktuálnej iterácie príkazov cyklu

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

Ak je splnená podmienka, **ihneď** ukončíme vykonávanie cyklu.

`break` a **`continue`** sa vzťahujú na najbližší cyklus



do-while (len pre informáciu)

- do-while – cyklus s **podmienkou na konci** (v praxi sa využíva najmenej):

```
do {
    príkazy cyklu
} while (podmienka);
```



Podmienka sa testuje **až po vykonaní príkazov** v tele cyklu.

Dôsledok: príkazy **do-while** cyklu sa vykonajú vždy aspoň raz.

Obľúbená štátnicová otázka:
Vieme sa bez niektorého z troch typov cyklu zaobísť? Prečo?

while vs. do-while





Metódy, ktoré počítajú ...

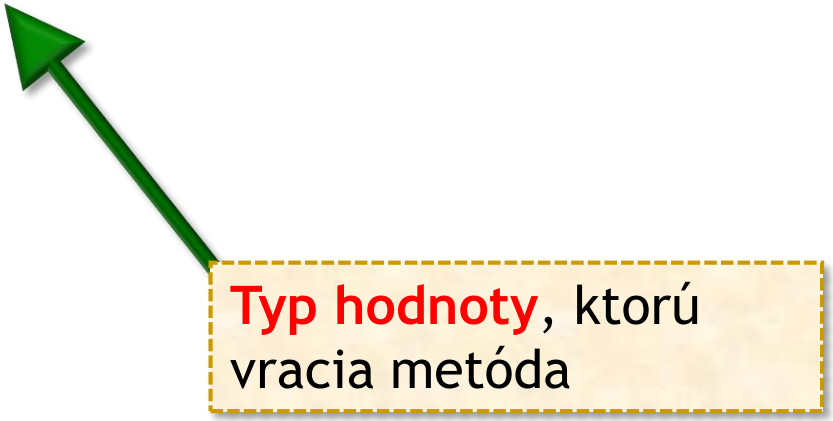
- Mnoho korytnačích metód **vráti** (vypočíta) nejakú hodnotu
  - *getX()* – vráti aktuálnu x-ovú súradnicu korytnačky
  - *directionTowards(double, double)* – vráti smer k danému bodu
  - *distanceTo(double, double)* – vráti vzdialenosť korytnačky k danému bodu
- Skúmajme ďalšie metódy cez Object Inspector ...



Ako naučiť metódy vrátiť hodnotu?

```
public double squareSpiral() {


}
```



- Postup:
  - Namiesto **`void`** napíšeme typ hodnoty (**`int`**, **`double`**, **`float`**, **`byte`**, …), ktorú vracia metóda
  - **`void`** = vraciam nič („len spravím to, čo treba“)



Príkaz vrátenia hodnoty

- Na vrátenie hodnoty použijeme príkaz **return**:

```
return odometer;
```



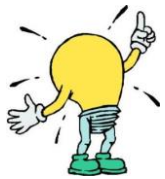
Hodnota, ktorú chceme **vrátiť** (vypočítať) z aktuálneho vykonania metódy

Hodnota v príkaze **`return`** musí byť kompatibilná s deklarovaným typom návratovej hodnoty!

`return` ukončuje vykonávanie metódy!



Na čo treba pamätať



- Vykonanie príkazu **`return`** **ukončuje ďalšie vykonávanie** príkazov metódy.
- Každá metóda, ktorá nevracia **`void`**, **musí** každé svoje vykonanie **skončiť príkazom** **`return`** (Java to prísne kontroluje).
- Vrátená hodnota musí byť **kompatibilná** s definovaným typom návratovej hodnoty
  - metóda definovaná ako metóda vracajúca *int* nemôže vrátiť reálne číslo (*double* hodnotu)
- Príkaz **`return;`** možno použiť v metódach vracajúcich **`void`** na okamžité ukončenie metódy.

Korytnačka - vedec

- Naučíme korytnačky **zaujímavé počty:**
  - nájsť počet deliteľov zadaného celého čísla
    - číslo **A** je deliteľné číslom **B** ak `A % B == 0`
  - nájsť najväčšieho spoločného deliteľa dvoch čísel
  - zistiť, či je číslo prvočíslo
  - spočítať počet cifier zadaného celého čísla
  - ...



Čo je to číslo?



70 0x46

LXX

七十

Číslo != Zápis čísla



Práca s celými číslami - finty

- **Posledná cifra:**
  - `123 % 10 = 3, 34 % 10 = 4, 12345 % 10 = 5`
  - Posledná cifra čísla v premennej c: `c % 10`
- **Skrátenie čísla o poslednú cifru:**
  - `123 / 10 = 12, 34 / 10 = 3, 12345 / 10 = 1234`
  - Vytvorenie čísla, ktoré vznikne odstránením poslednej cifry: `c / 10` (celočíselné delenie)
- **Pridanie poslednej cifry:**
  - `123 ◄ 8 = 123 * 10 + 8 = 1230 + 8 = 1238`
  - `cislo = cislo * 10 + cifra;`



Ciferný súčet

- **Ciferný súčet čísla 294 = 2 + 9 + 4 = 15**

```
public int digitSum(int number) {
   int result = 0;
   while (number > 0) {
       int digit = number % 10;
       result = result + digit;
       number = number / 10;
   }
   return result;
}
```

Vyberieme poslednú cifru

Cifru pripočítame k výsledku

Vyrobíme nové číslo, ktoré bude skrátené o poslednú cifru



Nech aj ľudia chápu!

- Programy treba písať tak, aby boli čitateľné a prehľadné ... nielen pre Javu ale **aj pre ľudí**

```
// vyberieme poslednu cifru
int digit = number % 10;
// pridame cifru do suctu
result = result + digit;
// skratime cislo o poslednu cifru
number = number / 10;
```



Komentáre

- Všetky znaky za dvojicou znakov `//` až do konca riadku sú považované za komentár
- Všetky znaky medzi dvojicou znakov `/*` a `*/` sú považované za komentár
- Komentáre môžeme písať kdekoľvek ...

Konvencia v Jave – komentár je na riadku **pred príkazom**

Komentujte svoje programy !!!



Vypisovanie „do IntelliJ“

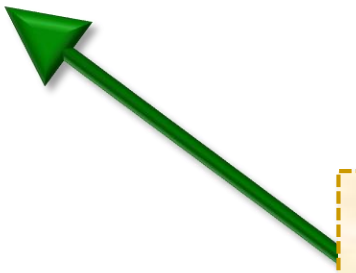
- Niekedy sa nám hodí vyskúšať metódy aj bez použitia Object Inspectora...

- Vypisovať vieme aj „do IntelliJ“:

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

System.out.println(albert.digitSum(293));

System.out.println("Ahoj svet");
```



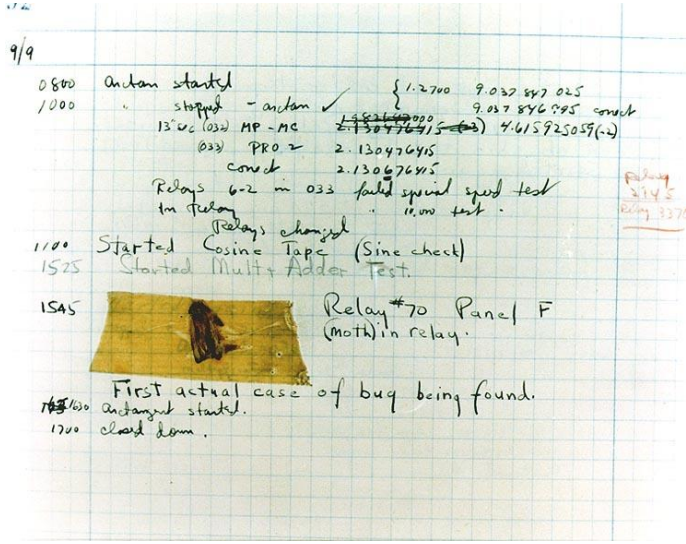
Ak chceme vypísať nejaký text, dáme ho do úvodzoviek ""

Debugovanie

- **Debugovanie** = ladenie, trasovanie, krokovanie







- Človek je tvor omylný, počítač naštastie nie ...
- Zvládnutie **debugovania je nevyhnutné** pre vývoj väčších programov ...
- Ukážka ...



Premenné v Jave (opäť)

- Dva typy premenných v Jave:
  - **primitívneho typu**
    - *int*, *byte*, *short*, *long*, *double*, *float*, *boolean a char*
    - okrem char poznáme všetky ...
    - životná misia: **uchovávať jednoduchú hodnotu**
  - **referenčného typu**
    - životná misia: **referencovať objekty** (umožniť komunikáciu s objektami)

```
WinPane sandbox;
```



Premenná `sandbox` schopná referencovať objekty triedy `WinPane`

Referencia

Referenčné premenné

„svet objektov“

```
ObjectInspector oi;
```

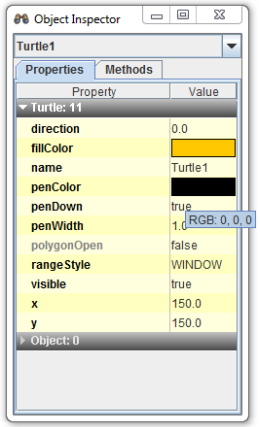
oi

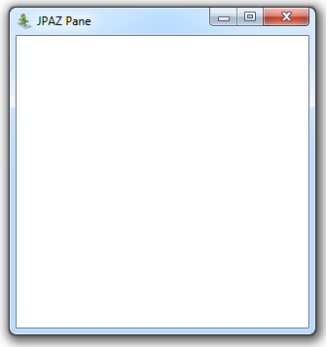
```
Turtle franklin;
```

franklin

```
WinPane sandbox;
```

sandbox





Referenčná premenná **obsahuje referenciu** (odkaz, prepojenie) **na** nejaký **objekt** …



Deklarácia referenčnej premennej

```
WinPane sandbox;
```

Objekty akej triedy môžu byť referencované z vytváranej premennej

Názov premennej

- Referenčná premenná nemôže referencovať hocijaké objekty!
- Ako každá iná premenná, aj referenčná premenná musí byť pred prvým použitím najprv inicializovaná.

Metafora referencie (1)

- Každý človek na Slovensku je **jedinečne identifikovaný rodným číslom**, ktoré mu je pridelené pri narodení...
- Každý objekt vytvorený vo „svete Javy“ je identifikovaný „javackym rodným číslom“
  - Niektoré objekty prezradia svoje „javacke rodné číslo“ cez metódu **`toString`**



„javacke rodné číslo objektu“

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

Metafora referencie (2)

```
int cislo;
```

Premenná `cislo` je schopná uchovať jedno celé číslo

```
boolean platnost;
```

Premenná `platnost` je schopná uchovať pravdivostnú hodnotu

```
WinPane sandbox;
```

Premenná `sandbox` je schopná uchovať „rodné číslo“ jedného java-objektu triedy `WinPane` alebo špeciálnu hodnotu **`null`**

2e257f1b





Hodnoty referenčnej premennej

- V referenčnej premennej môže byť uložené:
  - referencia na objekt („rodné číslo objektu“)
  - **`null`** – špeciálna hodnota hovoriaca, že premenná neobsahuje referenciu na žiaden objekt

```
sandbox = null;
```

Premenná `sandbox` nereferencuje žiaden objekt (triedy WinPane).

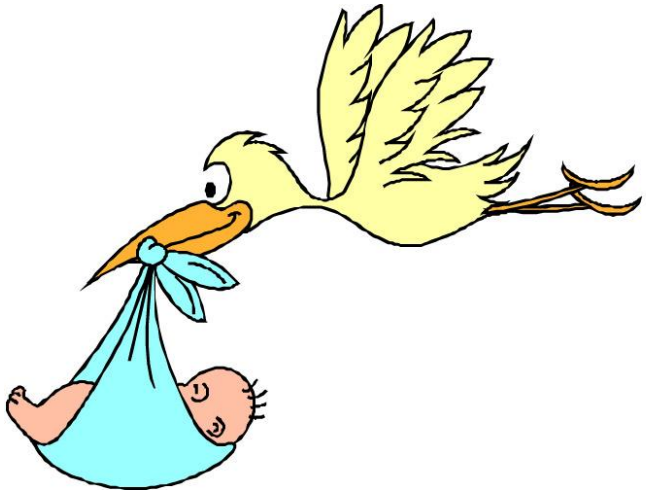
```
if (sandbox != null) {
```

Test, či referenčná premenná niečo referencuje (jej hodnota nie je **`null`**)

Vytvorenie objektu (1)

- Objekty (vo „svete objektov“) vytvárame príkazom **new**, **výsledkom** operácie **je referencia** („rodné číslo”) na vytvorený objekt.

```
new WinPane();
```



Názov triedy, ktorej objekt požadujeme vytvoriť



Vytvorenie objektu (2)

- **Konštruktor**
  - špeciálna „vec“, ktorá **vytvára** a **inicializuje** objekt danej triedy
  - konštruktor môže mať **parametre**, ktoré sa zadávajú do zátvoriek `()` pri vytváraní objektu
  - trieda môže mať **viacero konštruktorov** líšiacimi sa zoznamom parametrov

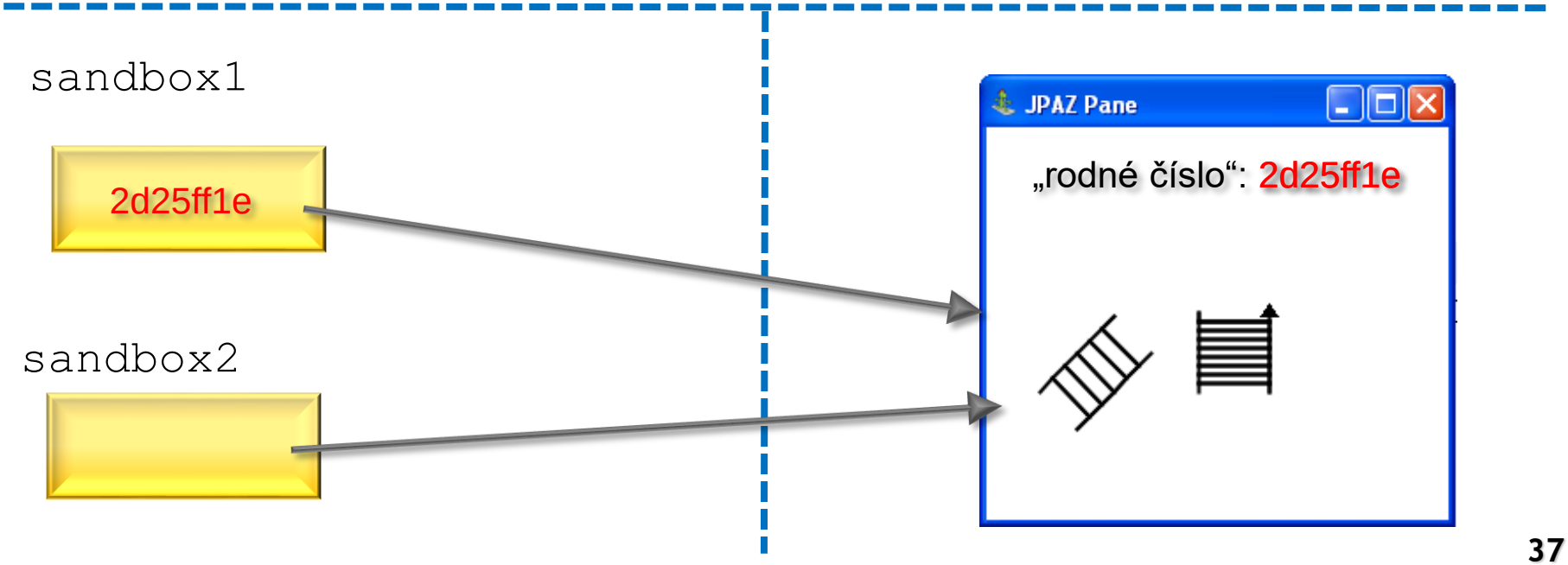
Parametre konštruktora = upresnenie, aký objekt chceme vytvoriť

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

Priradenie referencií

```
WinPane sandbox1;
sandbox1 = new WinPane();
WinPane sandbox2;
sandbox2 = sandbox1;
```

Na konci budú premenné `sandbox1` a `sandbox2` referencovať ten istý objekt.



Metafora referencie (3)

```
WinPane sanbox;

sandbox = new WinPane();

…

sandbox.clear();
```

Vytvorí sa premenná `sandbox` schopná uchovať „rodné číslo“ objektu triedy `WinPane`

Vytvoríme objekt triedy `WinPane` a jeho „rodné číslo“ uložíme do premennej `sandbox`.

Objektu, ktorého „rodné číslo“ je aktuálne uložené v premennej `sandbox`, povieme, aby vykonal metódu `clear`.



Rovnosť referencií

```
if (sandbox1 == sandbox2) {


}
```

Testujeme, či referenčné premenné `sandbox1` a `sandbox2` referencujú (ukazujú na) **ten istý** objekt.

Metafora: testujeme, či `sandbox1` a `sandbox2` uchovávajú rovnaké „rodné číslo“



Na čo treba pamätať

- Nie je pravda, že **neinicializovaná** premenná referenčného typu obsahuje **null**. Premenná je jednoducho neinicializovaná (okrem priradenia s ňou nemožno nič robiť)
- Objekty vieme **len vytvoriť**, nevieme ich zničiť
- Ilúzia vytvorená Javou: objekty žijú večne (resp. kým program neskončí)
  - ...narozdiel od lokálnych premenných
- Jeden objekt môže byť **referencovaný z viacerých** premenných
- Ak na objekt **neexistuje referencia**, nedokážeme s ním komunikovať – objekt je stratený.



O čom to dnes bolo?

- Pretypovanie (explicitné, implicitné)
  - generovanie náhodných **celých** čísel
- Cykly: `for`, `while`, `do-while`
  - `break` a `continue` na ich prerušenie
- Metódy vracajúce hodnoty a príkaz `return`
- Komentáre
- Debugovanie
- Referenčné premenné a konštruktory



to be continued …

ak nie sú otázky...

Ďakujem za pozornosť !

