



5. prednáška (20.10.2025)



Polia



***Korytnačky na farme a
korytnačí futbal***



Čo už vieme...

- Poznáme **podmienkový príkaz** (if-else)
- Poznáme **cykly**: **for**, **while**, **do-while**
 - **break** a **continue** na ich prerušenie
- Metódy vracajúce hodnoty a príkaz **return**
- Komentáre, debugovanie
- Premenné v metódach (lokálne premenné)
 - primitívny typ (8 typov) vs. referenčný typ
- Čo je to **referencia**, na čo slúži a ako sa používajú premenné referenčného typu



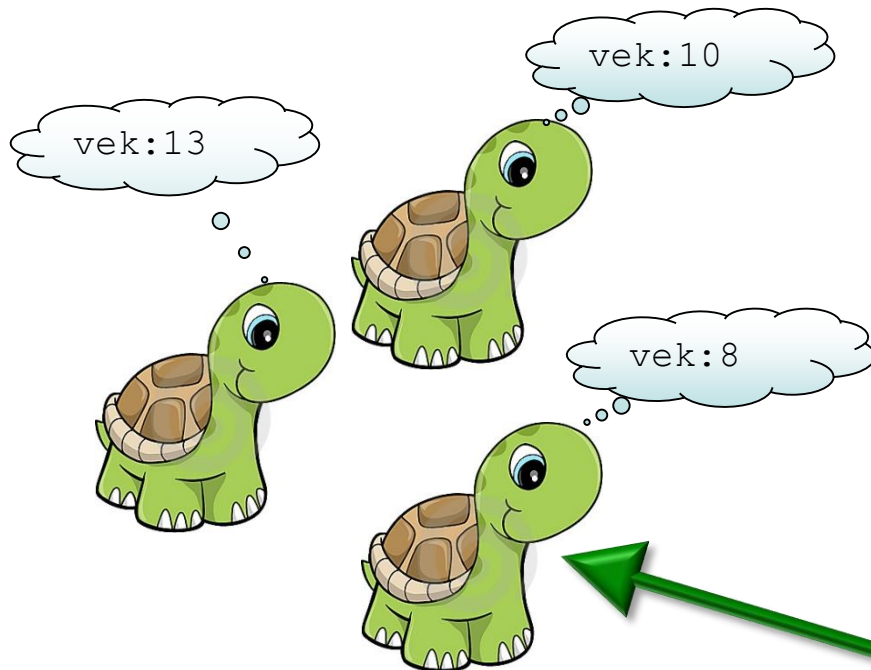
Čo už vieme...

- **Trieda** je „vzor“ správania sa pre všetky objekty danej triedy - „**šablóna**“
- Ako sa **vytvára objekt triedy** pomocou **new**
 - vieme, že objekt môže mať viacero konštruktorov líšiacich sa parametrami (`WinPane`, `String`, ...)
 - vieme, že `String` na uchovávanie postupnosti znakov je „extra“ trieda... („skratky“ pre programátorov)
- **Vytvoriť vlastnú triedu** rozširujúcu triedu *Turtle* alebo *WinPane* a popridávať do nej nové metódy
- Ako pridať do všetkých objektov nejakej triedy „trvalú pamäť“ - **inštančné premenné**

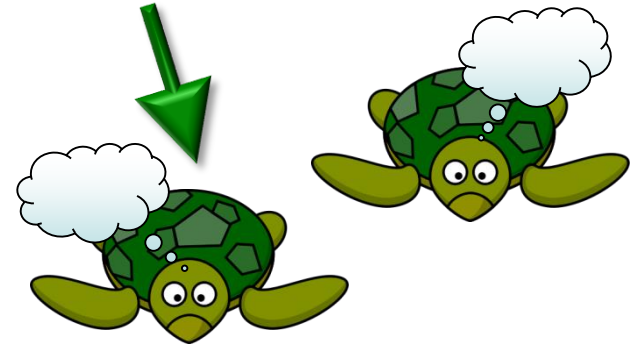


Inštančné premenné

```
public class KorytnackaSPamatou extends Turtle {  
    private int vek;  
  
    ...  
}
```



new Turtle();



new KorytnackaSPamatou();



Čo už vieme...

- Ak do triedy rozširujúcej triedu `WinPane` pridáme metódy „so správnym menom a parametrami“, vykonajú sa pri myšacích aktivitách



```
protected void onMouseClicked(int x, int y,  
                                MouseEvent detail)
```

```
protected void onMousePressed(int x, int y,  
                                MouseEvent detail)
```



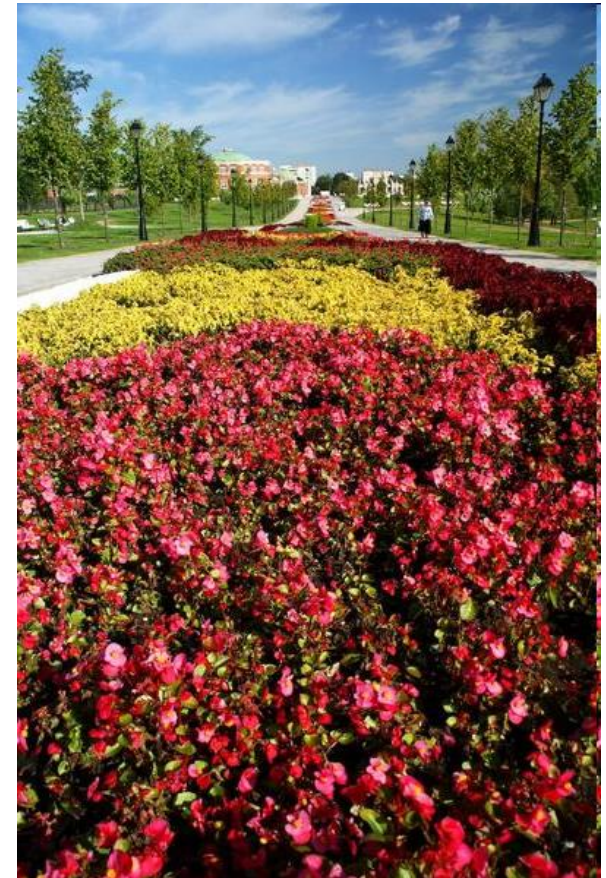
Myšacie udalosti

- **Udalosti**, ktoré môžeme obsluhovať (spracovať):
 - **onMousePressed** - pri zatlačení tlačidla myši
 - **onMouseReleased** - po uvoľnení tlačidla myši
 - **onMouseClicked** - po kliknutí (postupnosť: Pressed, Released, Clicked)
 - **onMouseMoved** - pri pohybe myši bez zatlačeného tlačidla myši
 - **onMouseDragged** - pri pohybe myši so zatlačeným tlačidlom myši



Projekt: kvetinová farma

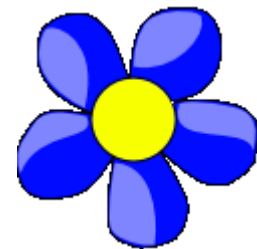
- Ideme vyrobiť:
 - Triedu **Farma**, rozširujúcu triedu *WinPane*
 - Každá farma:
 - má šírku 500 a výšku 300
 - má hnedé pozadie
 - je rozdelená na 5 záhonov rovnakej šírky
 - hranice záhonov sú čierne vertikálne čiary
 - Kliknutím do záhonu na danom mieste v záhone posadíme **kvet**





Čo chceme od farmy

- Aké metódy má mať „kvetinová“ farma:
 - **getPocetZahonov** - vráti počet záhonov
 - **vZahone** - vráti počet kvetín v i-tom záhone
 - **pocetKvetov** - vráti počet kvetín v celej záhradke
 - **sViacAko** - vráti počet záhonov, v ktorých je viac ako parametrom zadany počet kvetín
 - **vsadeNieco** - vráti, či je v každom záhone aspoň jeden kvietok
 - **najZáhon** - vráti poradové číslo (index) záhonu, v ktorom je najviac kvetín (ak je maximálnych záhonov viac, tak index ktoréhokoľvek z nich)
- Metódy skúmame cez ObjectInspector ...





Vytvorenie farmy

- Pri vytvorení objektu farmy spravíme:
 - nastavíme rozmery kresliacej plochy na 500 x 300
 - nastavíme farbu pozadia na hnedú
 - `new Color(255, 211, 155)`
 - nakreslíme hranice záhonov
 - spravíme si na to pomocnú metódu s počtom záhonov určeným parametrom, ktorú zavoláme...

Ako to spraviť, aby to všetko objekt triedy Farma **spravil „sám“ pri svojom vytvorení** (nie cez naše príkazy v „spúšť ači“)?



Pri zrode objektu ...

- Java nám ponúka možnosť, ako vykonať nejaké príkazy, hneď ako sa vytvorí objekt ...

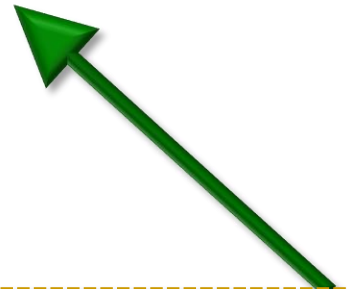
```
public class Farma extends WinPane {
```

```
    public Farma() {
```

```
        // inicializacne prikazy
```

```
    }
```

```
}
```



„inicializačná metóda“: má rovnaký názov ako trieda a nemá návratový typ



Programujeme...

- Pridáme metódy:

```
public void pridaJKvietok(int x, int y);
```

```
public int getPocetZahonov();
```



Zmena tvaru korytnačky

- Korytnačka vie zmeniť svoj tvar:

```
korytnacka.setShape(  
    new ImageTurtleShape("/kvietok.png"));
```



Názov súboru s obrázkom
vloženým do projektu

- Cez metódu `stamp` vie korytnačka otlačiť svoj tvar do kresliacej plochy





Programujeme...

- Pridáme metódy:

```
public void pridaJKvietok(int x, int y);  
public int getPocetZahonov();
```

- Ako zistíme počet kvetov v konkrétnom záhone?

- vytvoríme inštančné premenné, v ktorých si budeme pamätať počet kvetín v jednotlivých záhonoch:

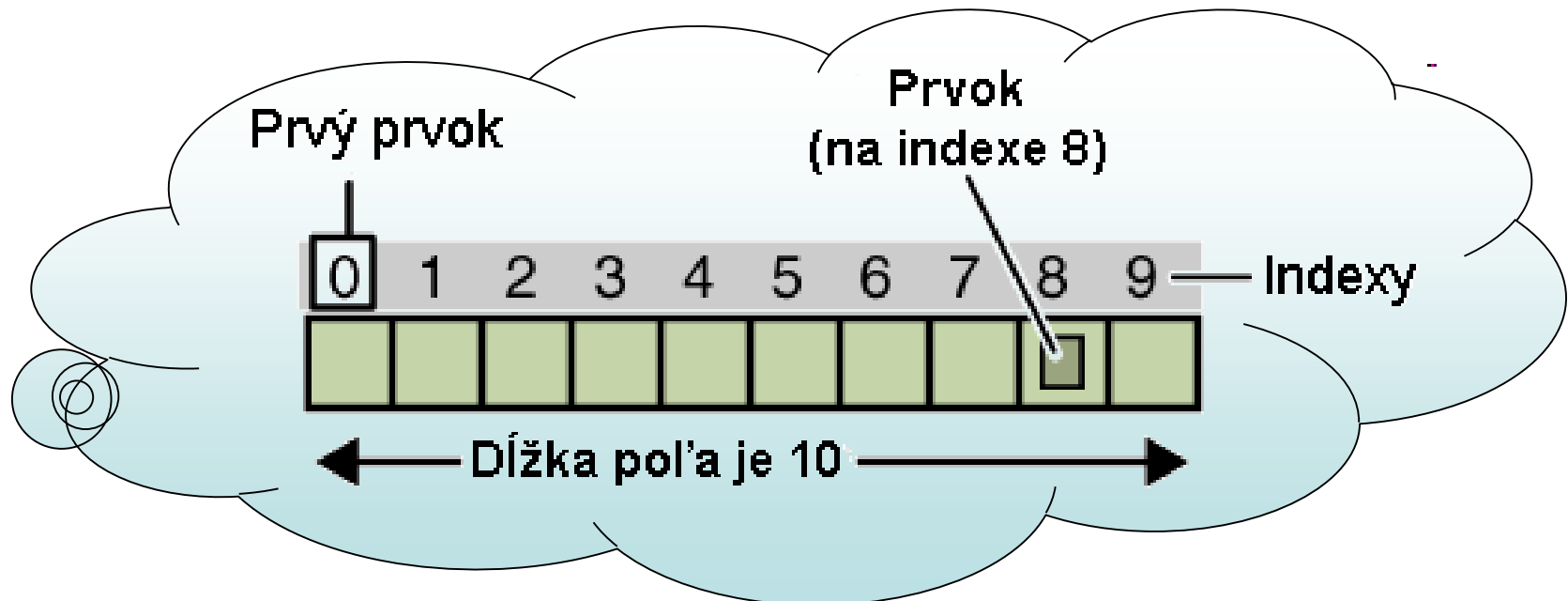
```
private int vZahone0;  
private int vZahone1;  
private int vZahone2;  
private int vZahone3;  
private int vZahone4;
```



Čo ak budeme
chcieť väčšiu farmu
so 100 záhonmi?



- Pole je špeciálny Java **objekt**, ktorý obsahuje veľa hodnôt (**políček**) rovnakého typu. Jednotlivé políčka sú číslované (indexované) od 0.





Pole - metafora

„skrinka“ na 7 **int**-ov



Objekt, ktorý má „**veľa** šuflíkov“ na uchovanie hodnôt nejakého typu (primitívneho alebo referenčného).

Všetky „šuflíky“ sú **rovnakého typu**.

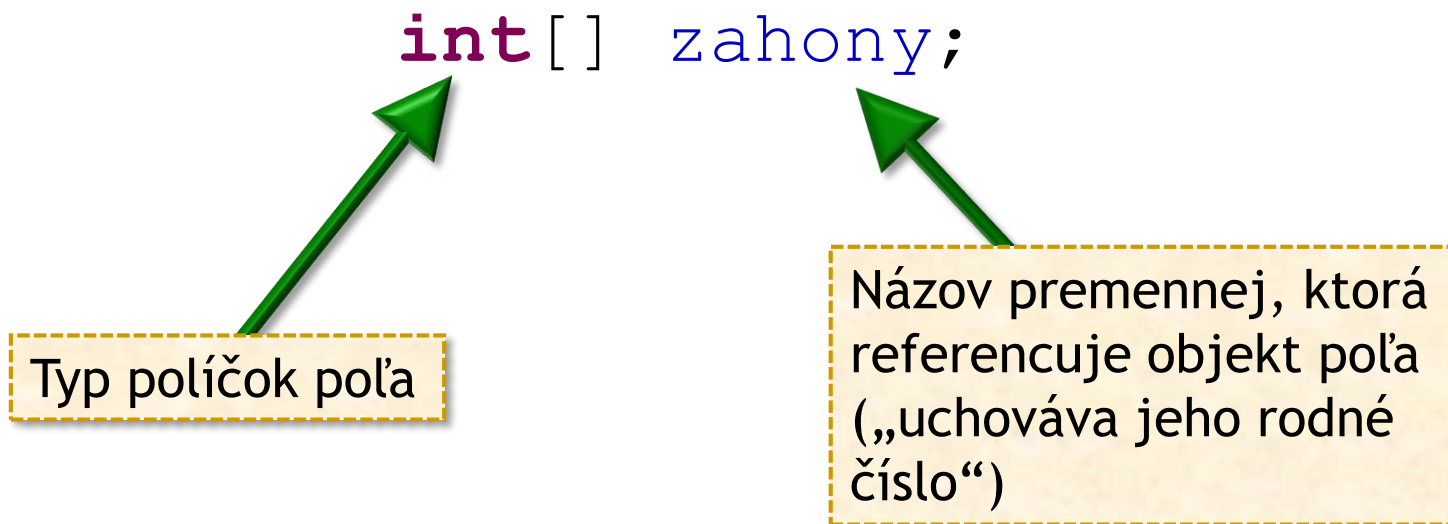
„Šuflíky“ sú označené číslami - **indexmi** počnúc indexom 0





Deklarácia „poľovej“ premennej

- Deklarácia premennej referencujúcej „poľový“ **objekt**:



- pri inštančných premenných pridáme **private**



Vytvorenie objektu poľa

```
zahony = new int [5];
```

Typ políčok poľa

Počet políčok
vytváraného
poľa („poľového
objektu“)

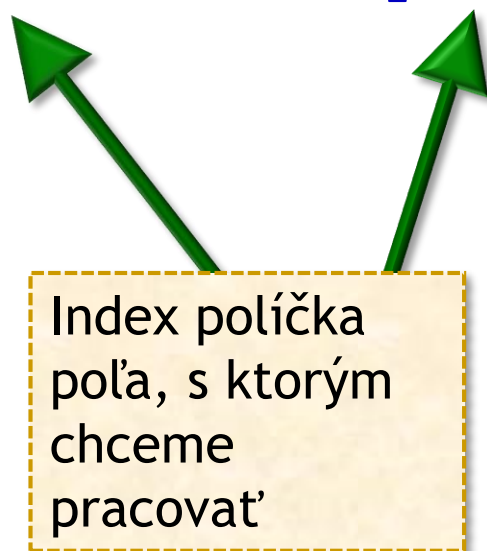
- Vytvorí sa **objekt** „poľa“, ktorý sa skladá z **5 políčok**. Každé políčko je schopné uchovať jedno celé číslo (hodnotu typu **int**).
- Do premennej **zahony** sa uloží **referencia** na vytvorený „poľový“ objekt.



Prístup k políčkam poľa

```
zahony[0] = 1;
```

```
zahony[1] = zahony[0] + 5;
```



- Medzi hranaté zátvorky píšeme **index** políčka
- Indexy štartujú od 0!



Čo o poliach ešte nevieme

- Po vytvorení poľa sú v jednotlivých políčkach poľa „**defaultné**“ **hodnoty** pre typ políčok poľa
- Počet políčok poľa získame pomocou „**length**“ (pozor, je to špeciálna vec, nie metóda, ako pri objektoch triedy String):

```
for (int i=0; i<zahony.length; i++) {  
  
}
```



Pozor, tu žiadne
zátvorky ()



„Defaultné“ hodnoty

- Pre `int`, `byte`, `short`, `long`, `float`, `double` je to hodnota `0`
- Pre `boolean` je to `false`
- Pre `char` je to Unicode znak s kódom `0`
- Pre premenné referenčného typu (referencujúce objekty tried ako *String*, *Turtle*, ...) je to `null`.





Farma: čo potrebujeme?

- Pre každý záhon potrebujeme vedieť **koľko** je tam práve **kvetín**:
 - pole hodnôt typu **int** - každé políčko poľa uchováva číslo, ktoré hovorí, koľko kvetov je práve v záhone

```
private int[] zahony;
```

- vytvorenie poľa:

```
this.zahony = new int[5];
```



Poččet prvkov s vlastnosťou

● Stratégia:

- Vytvoríme si **počítadlo**, v ktorom rátame koľko prvkov s danou vlastnosťou sme stretli.
- Prechádzame **postupne** všetky prvky poľa. Ak má prvok požadovanú vlastnosť (overíme **if**-om), **zvýšime** hodnotu počítadla o 1.
- Po skončení prechodu prvkami obsah počítadla je počet prvkov poľa s danou vlastnosťou.



Všetky prvky majú vlastnosť ...

● Stratégia:

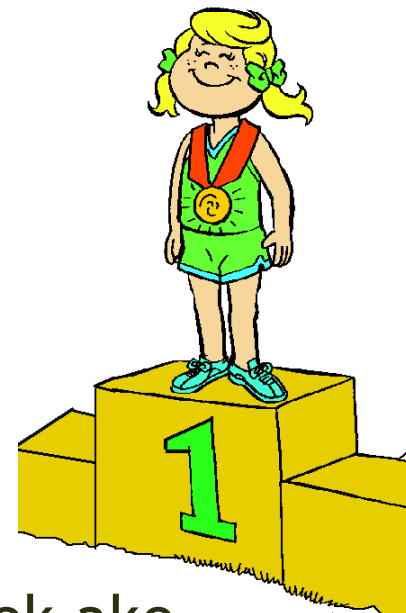
- Na začiatku **veríme**, že všetky prvky majú nejakú vlastnosť - nastavíme si nejakú *boolean* premennú (napr. s menom *allOK*) na **true**.
- Prechádzame postupne všetky prvky poľa. Ak je prvok „**zlý**“ (nemá požadovanú vlastnosť), nastavíme *boolean* premennú na **false** a „**break**“-neme prechádzanie prvkov poľa.
- Po skončení prechodu prvkami obsah *boolean* premennej hovorí, či všetky prvky majú skúmanú vlastnosť.



Naj-prvok poľa

- **Stratégia:**

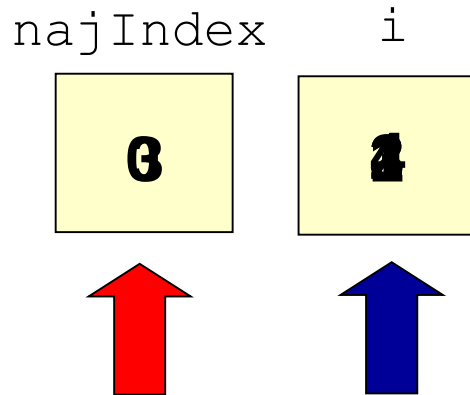
- vyberieme si nejaký prvok poľa ako **kandidáta na najlepší prvok**
- skúšame každý prvok, či náhodnou nie je lepší, ako aktuálny kandidát:
 - ak áno, zapamätáme si aktuálny prvok ako **nového kandidáta** na najlepší prvok



- Ak je zvolit' počiatočného kandidáta ťažké, tvárime sa, že za kandidáta máme zvolenú absolútne najhoršiu vec.

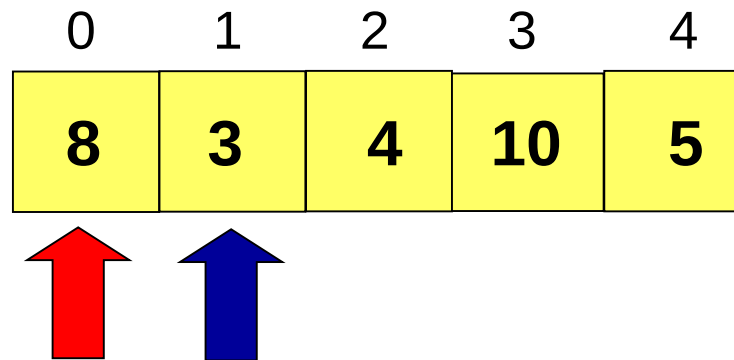
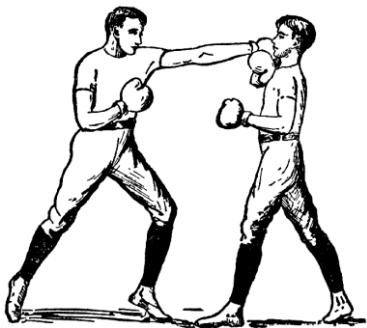


Naj-prvok poľa – simulácia



```
int[] pole = ...;
int najIndex = 0;
for (int i=1; i<pole.length; i++) {
    if (pole[najIndex] < pole[i]) {
        najIndex = i;
    }
}
```

80 vs 10





Na čo musíme pamätať

Polia sú objekty!

**Premenné a parametre „pol'ové“
objekty len referencujú!**

Veľkosť poľa nemožno zmeniť.



Korytnačí futbal



- Chceme vytvoriť:
 - Triedu **Ihrisko** rozširujúcu triedu **WinPane**
 - Plocha ihriska je zelená
 - Je tam lopta na náhodnej pozícii
 - Lopta sa presunie smerom, kam sme klikli, a dokážeme ju presúvať ťahaním



Vytvárame ihrisko

- Pri **vytvorení** ihriska spravíme:
 - Nastavíme farbu pozadia na zelenú
 - Vytvoríme korytnačku „loptu“ **s tvarom lopty**
 - Umiestnime ju na náhodnú pozíciu
 - V inštančnej premennej si zapamätáme referenciu na korytnačku-loptu, aby sme s ňou mohli komunikovať



Korytnačka - lopta

- V inicializačnej metóde korytnačky „lopty“ jej zmeníme tvar a vypneme kresliace pero

```
public class Lopta extends Turtle {  
  
    public Lopta() {  
        // nastavenie tvaru ...  
    }  
  
}
```



Ako správne ťahať objekty

- Pri `onMousePressed`:
 - overíme, či nemáme začať ťahanie (kliklo sa na objekt, ktorý chceme ťahať)
- Pri `onMouseDragged`:
 - ak niečo ťaháme, tak to priestňujeme
- Pri `onMouseReleased`:
 - ak sa niečo ťahalo, tak to prestaneme ťahať



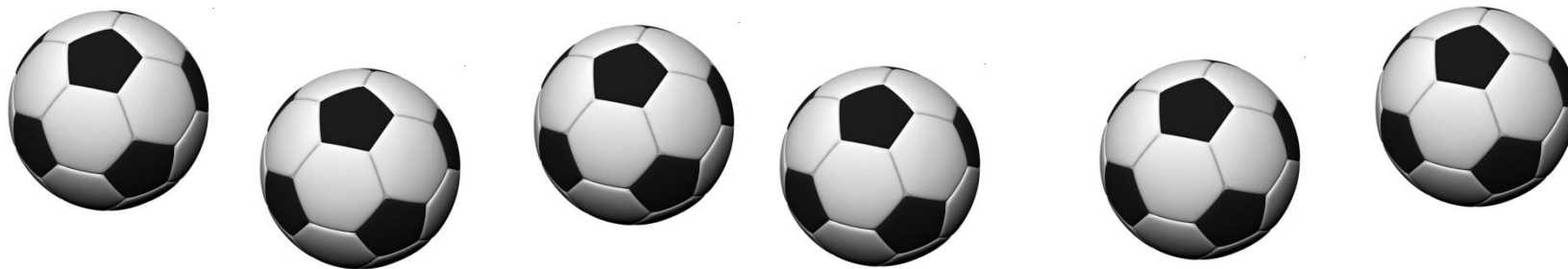
Potrebujeme si pamätať, či
niečo ťaháme – inštančná
premenná.



Jedna lopta je pre amatérov

- Chceme, aby na ihrisku bolo 10 lôpt ...
- ... potrebujeme si pamätať referencie na ne („ich rodné čísla“), aby sme s nimi mohli „komunikovať“:

```
private Lopta[] lopty;
```





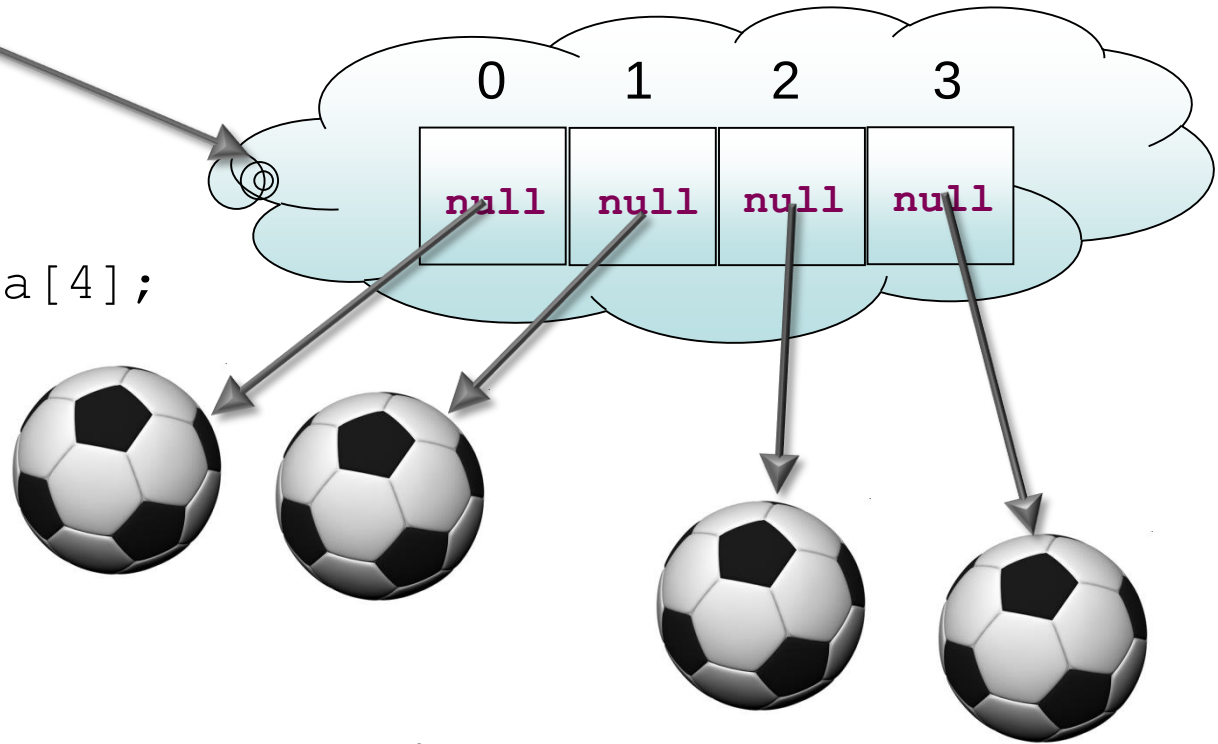
Lopty v poli

Lopta[] lopty



```
Lopta[] lopty;
```

```
lopty = new Lopta[4];
```



```
for (int i=0; i<lopty.length; i++) {  
    lopty[i] = new Lopta();  
}
```



Na čo musíme pamätať

**Ak chceme pole referencií
na objekty nejakej triedy,
tak vytvorením tohto poľa
tieto
objekty nevzniknú.**



Korytnačí viacloptový futbal

- **Ťahanie lopty:**

- V `onMousePressed` musíme zistiť, na ktorý objekt sa kliklo - do objektov triedy `Lopta` pridáme metódu, ktorá povie, či sa kliklo na danú loptu
- Namiesto toho, či je ťahaná lopta, si musíme pamätať aj, ktorú loptu ťaháme

- **Posunutie lopty:**

- Posunieme loptu, ktorá je najbližšie k miestu kliknutia: ako ju nájsť?



Najbližšia lopta

- Pamätáme si, ktorá lopta je **zatiaľ najbližšie**
 - v referenčnej premennej
- Na začiatku predpokladáme, že najbližšie je **prvá** lopta (index 0)
- **Skúšame každú loptu**, či nie je bližšie ako lopta, ktorá je doposiaľ najbližšie:
 - ak je skúšaná lopta bližšie, tak si ju zapamätáme ako doposiaľ najbližšiu
- Po skončení máme najbližšiu loptu

Stratégia: naj prvok v poli



Sumarizácia

- Na uchovanie väčšieho počtu hodnôt **rovnakého typu** používame polia („poľové objekty“)
- Pole je objekt, preto na prácu s ním potrebujeme **referenčnú premennú**, ktorá uchová referenciu naň („jeho rodné číslo“)
 - `int[] zahony;`
 - `Lopta[] lopty;`
- Pole vytvárame cez **new**:
 - `new Lopta[4];`
- K prvkom poľa pristupujeme cez **indexy** (od 0):
 - `zahony[0] = 1;`



Sumarizácia

- **Základné stratégie** algoritmov na prácu s poľom
 - počet prvkov poľa s nejakou vlastnosťou
 - overenie, či má nejaký prvok poľa zadanú vlastnosť
 - nájdenie „naj“ prvku v poli
- **Ďalšie špecialitky:**
 - „inicializačná metóda“ pre objekty
 - zmena tvaru korytnačky
 - pár stratégií ako na myšacie udalosti



to be continued ...

ak nie sú otázky...

Ďakujem za pozornosť !

