6. prednáška (26.10.2026)





2D-polia



Poďme programovať hry

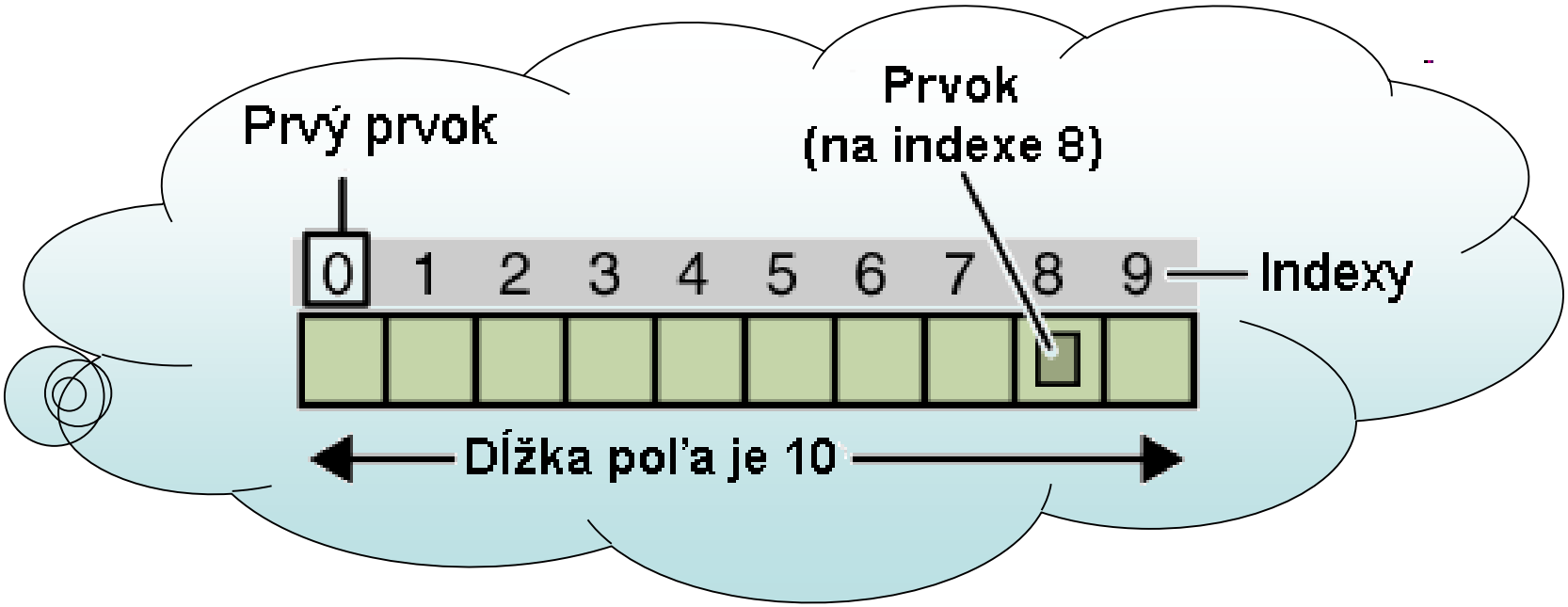


Čo už vieme...

- Vytvárať nové triedy „vylepšovaním“ existujúcich
- 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
- Lokálne aj inštančné **premenné**
  - primitívny typ (8 typov) vs. referenčný typ
- **Polia** („poľové objekty“)

Polia - opakovanie

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





Deklarácia „poľovej“ premennej

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

```
Lopta[] lopty;
```

Typ políčok poľa

Názov premennej, ktorá referencuje objekt poľa („uchováva jeho rodné číslo“)

- u inštančných premenných pridáme **`private`**



Vytvorenie objektu poľa

```
lopty = new Lopta[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 jednu referenciu na objekt triedy `Lopta` (alebo **`null`**).
- Do premennej `lopty` sa uloží **referencia** na vytvorený „poľový“ objekt.



Prístup k políčkam poľa

```
lopty[0] = new Lopta();
lopty[0].pohniSaSmerom(x, y);
```



Index políčka poľa, s ktorým chceme pracovať

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



Defaultné hodnoty a dĺžka poľa

- 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<lopty.length; i++) {

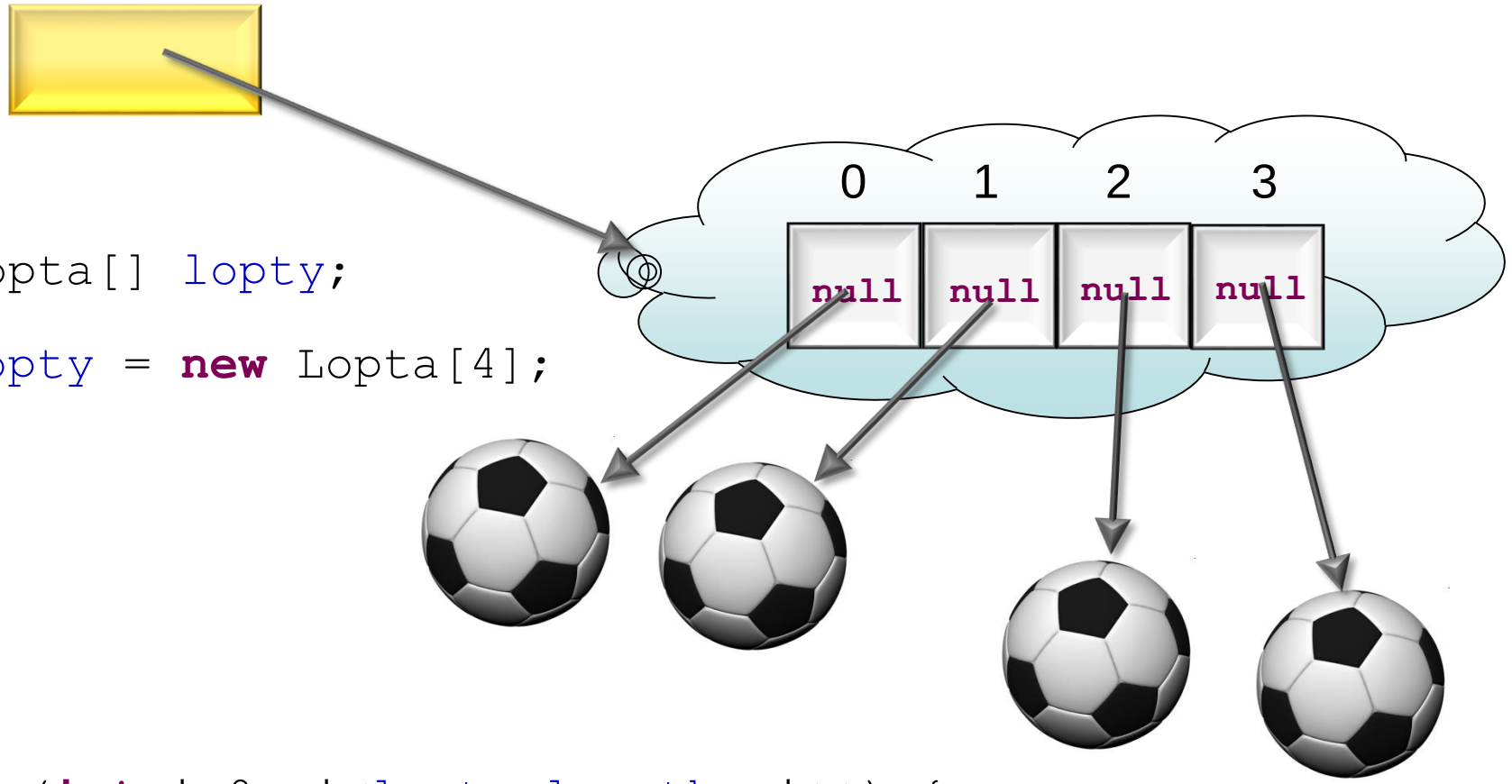
}
```



Pozor, tu žiadne zátvorky ()

Lopty v poli

```
Lopta[] lopty
```



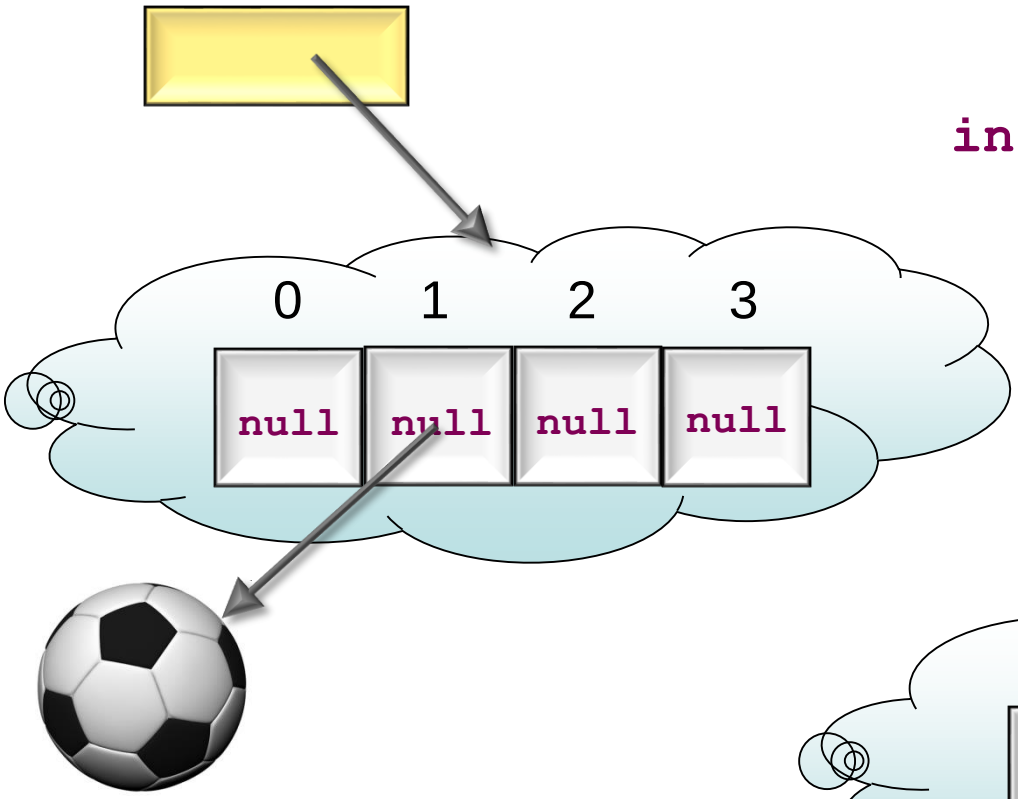

```
Lopta[] lopty;

lopty = new Lopta[4];
```

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

Pole referencií vs. pole hodnôt

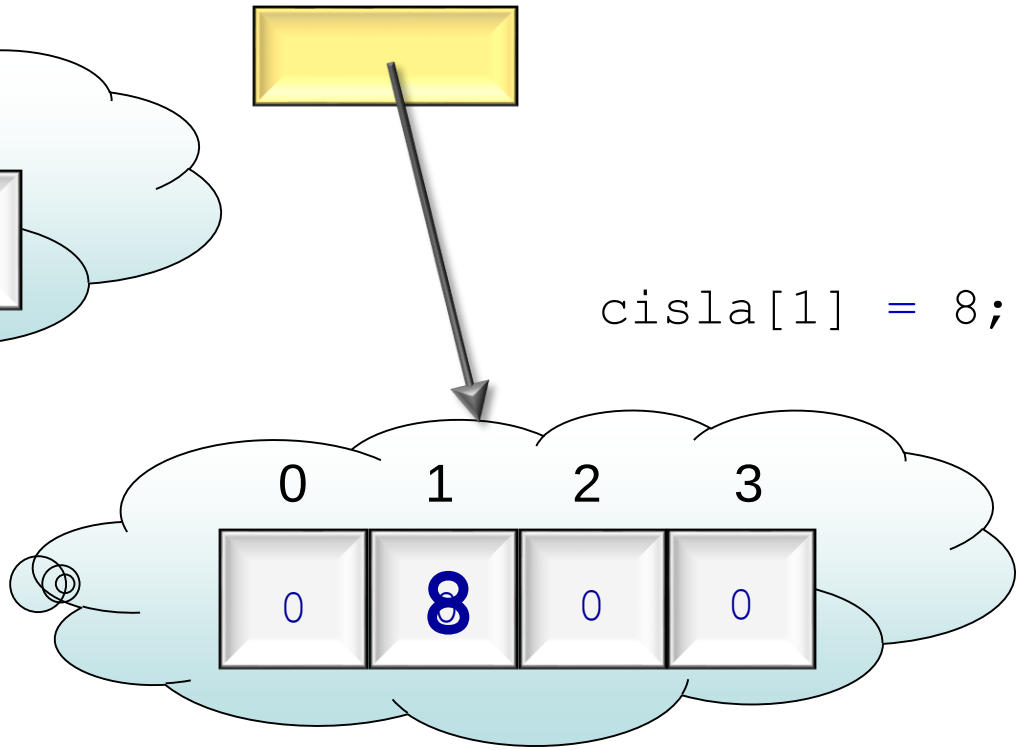
```
Lopta[] lopty = new Lopta[4];
```




```
lopty[1] = new Lopta();
```

```
int[] cisla = new int[4];
```

```
cisla[1] = 8;
```



Preplnené pole...

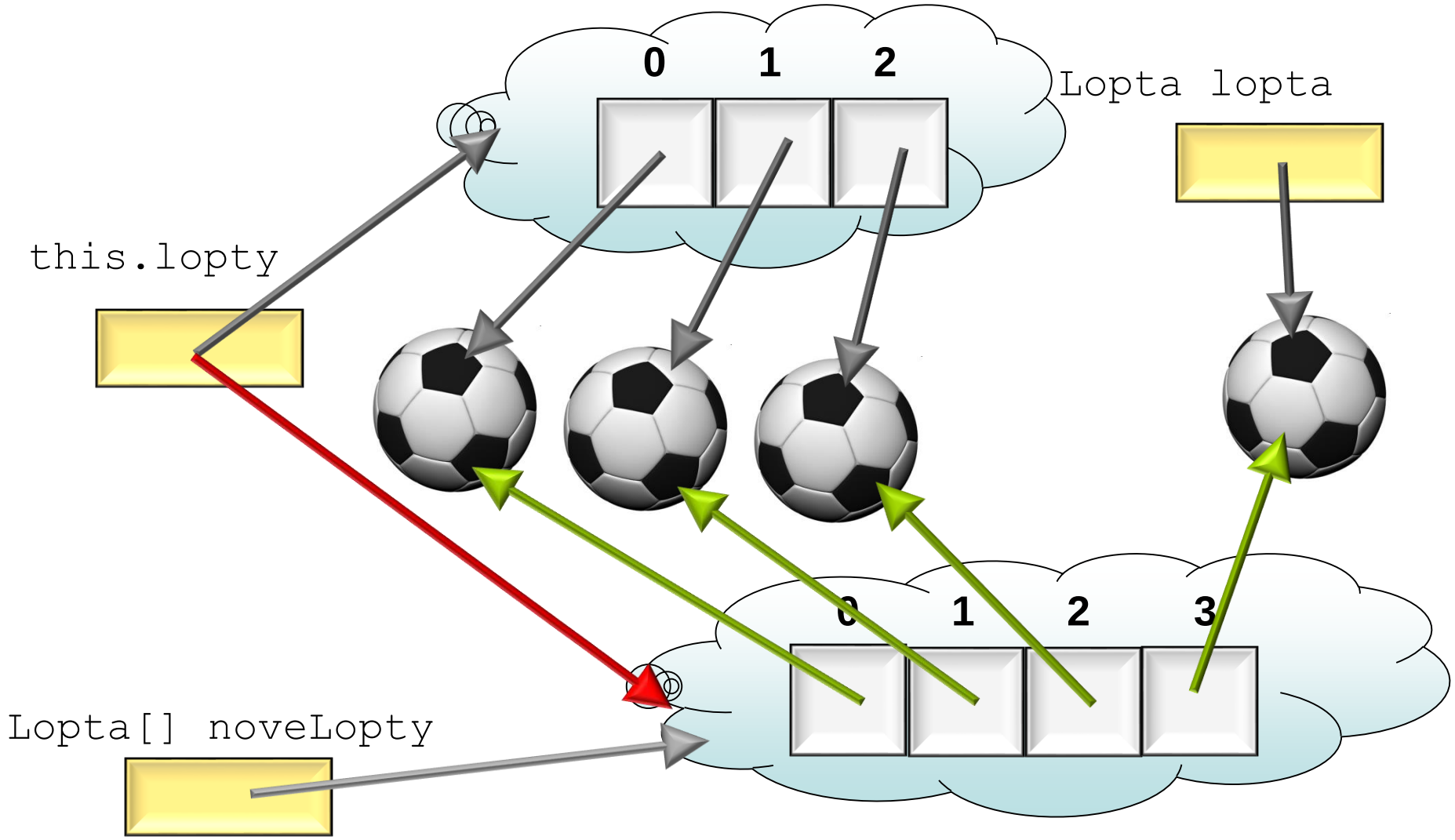
- Čo ak sa rozhodneme do ihriska pridať ďalšiu loptu?
  - Veľkosť poľa po vytvorení nemožno zmeniť...





???

Pridanie lopty (pridajLoptu)





Pridanie lopty

```
public void pridajLoptu(double x, double y) {
    Lopta novaLopta = new Lopta();
    this.add(novaLopta);
    novaLopta.setPosition(x, y);

    Lopta[] noveLopty = new Lopta[this.lopty.length + 1];

    for (int i = 0; i < this.lopty.length; i++) {
        noveLopty[i] = this.lopty[i];
    }

    noveLopty[noveLopty.length - 1] = novaLopta;

    this.lopty = noveLopty;
}
```

Vytvoríme loptu

Vytvoríme nové o 1 políčko väčšie pole (referencií na lopty)

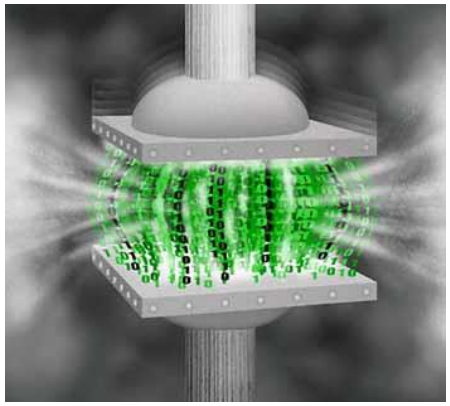
Skopírujeme referencie z pôvodného poľa do nového

Do posledného políčka uložíme referenciu na pridávanú loptu

Inštančnú premennú lopty necháme referencovať zväčšené pole

Zmenšenie poľa

- Idea:
  - Vytvorí sa **nové pole** „zmenšenej“ veľkosti
  - Prvky z pôvodného poľa, ktoré majú ostať, sa vhodne **skopírujú** do nového-zmenšeného poľa
  - Premenná referencujúca pôvodné pole sa zmení tak, **aby referencovala** „zmenšené“ pole





Korytnačka – poliarka (1)

- **Algoritmus** na nájdenie indexu políčka s maximálnou hodnotou je **rovnaký** – je jedno, či políčko predstavuje počet kvetov v záhone, počet ľudí v byte, ...

- Vytvorme, si korytnačku, ktorá bude poznať základné metódy (a triky) na prácu s poliami...



Korytnačka – poliarka (2)

- **Metódy na:**
  - Výpis poľa do konzoly
  - Index najväčšieho prvku v poli celých čísel
  - Index najmenšieho prvku v poli celých čísel
  - Orezanie hodnoty (ak je hodnota väčšia ako zadané x, tak sa oreže na x)
  - Vytvorenie poľa náhodných čísel zadanej dĺžky
  - Zistenie, či v poli čísel sú nejaké 2 rovnaké čísla



Korytnačka – poliarka (3)

```
public void vypisPole(int[] pole)
```

Ak v poli niečo zmeníme, zmena sa prejaví všade, kde toto pole referencujeme...

Parametrom je **referencia** na pole („poľový objekt“)

```
public int[] nahodnePole(int dlzka)
```

Metóda vráti **referenciu** na vytvorené pole („poľový objekt“), ktorého políčka sú schopné uchovávať `int`-y (celé čísla).



2 rovnaké čísla v poli?

- Idea:
  - Vyskúšame **všetky dvojice** indexov a overíme, či tam nie sú rovnaké čísla...

Dvojice indexov pri dĺžke poľa **4** (indexy: 0, 1, 2, 3):

Stačí porovnať **[a,b]**, netreba aj **[b, a]**

~~0 0~~	~~1 0~~	~~2 0~~	~~3 0~~
0 1	~~1 1~~	~~2 1~~	~~3 1~~
0 2	1 2	~~2 2~~	~~3 2~~
0 3	1 3	2 3	~~3 3~~

Neporovnávame políčko so sebou samým



2 rovnaké čísla v poli?

Porovnávané dvojice ak `p.length` je 4.

0 1
0 2 1 2
0 3 1 3 2 3

```
for (int i=0; i<p.length-1; i++) {
        for (int j=i+1; j<p.length; j++) {
                ...
        }
}
```

Pre aktuálne **i** generujeme v **j** postupne čísla od *i+1* po *length-1*



Vytvorenie inicializovaných polí

- Polia môžeme vytvoriť a inicializovať pripravenými hodnotami:

```
int[] pole = {2, 4, 8, 10, 1};
```

Premenná *pole* bude referencovať vytvorené pole.

Vytvorí pole čísel (intov) dĺžky 5 a naplní ho vymenovanými hodnotami
Kombinácia: `new int[5]` a piatich príkazov priradenia.



Vytvorenie inicializovaných polí

- `int[] pole = {2, 4, 8, 10, 1};`
- `char[] znaky = {'a', 'x', 'r'};`
- `String[] retazce = {"Dobre", "rano", "Java"};`
- Ako inicializačné hodnoty políčok môžeme písať nielen literály, ale aj **výrazy** ...

```
for (int i=0; i<10; i++) {
        int[] p = {i, i+1, i+2};
}
```



Pár užitočných metód

- `Arrays.toString()` - vyrobí reťazec obsahujúci "pekný **výpis**" prvkov poľa:

```
int[] p = {5, 8, 9};
System.out.println(Arrays.toString(p));
```

- `System.arraycopy()` – **skopíruje** prvky z jedneho poľa do druhého:

```
int[] p1 = {5, 8, 9};
int[] p2 = new int[4];
System.arraycopy(p1, 0, p2, 1, 3);
```

Koľko políčok

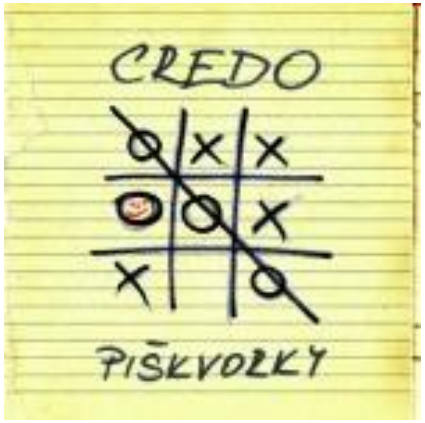
Odkiaľ

Od akého indexu

Kam

Od akého indexu

Piškvorky



- **Hracia plocha:** 15 x 15
- **1. hráč:** červené bodky
- **2. hráč:** modré bodky
- Víťazom je prvý, kto označí aspoň 5 rovnakých bodiek v rade, stĺpci alebo po uhlopriečke



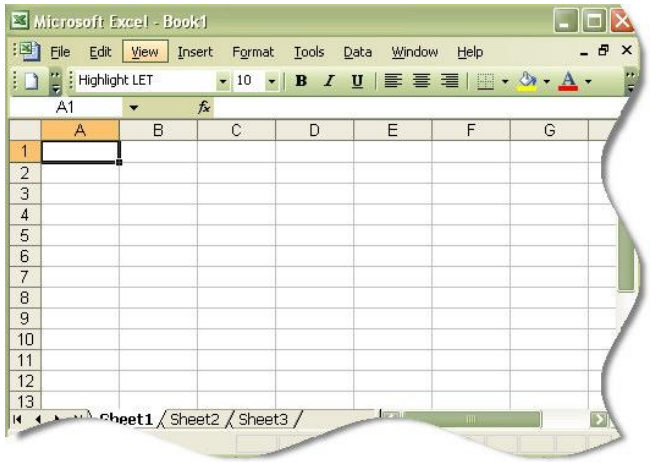
Piškvorky - predpríprava

- Postup (bolo na cvičeniach):
  - Metóda na nakreslenie mriežky
  - Nakreslenie mriežky pri vytvorení plochy
  - Metóda na nakreslenie bodky do políčka
  - Obsluha klikania
  - Striedavá zmena farby
- Ako vyriešiť to, aby sa jedno políčko nedalo obsadiť 2-krát ?
  - **Riešenie:** **pamätajme si obsadenosť políčok**

Spomínate si na projekt **Kvetinová farma** s poľom, ktoré uchovávalo počty kvetín v záhonoch?

2-rozmerné polia

- **Predstava:** 2-rozmerné pole = *m* x *n* matica („tabuľka”)
- Všetky políčka „tabuľky“ sú rovnakého typu
- Políčka sú prístupné pomocou **dvojice indexov**
- 2D pole = matica

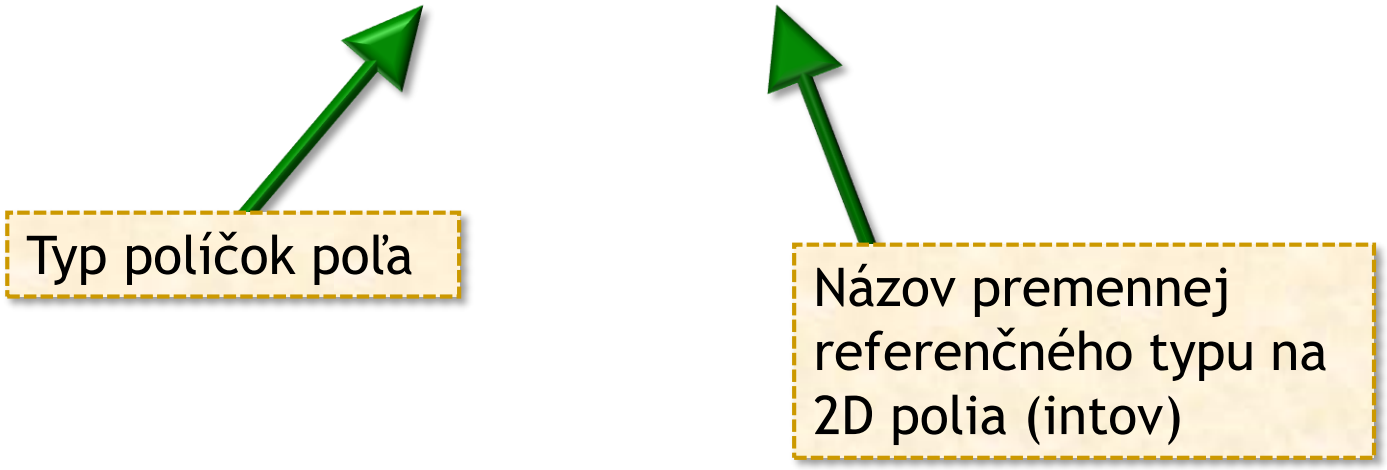


[0][0]	[0][1]	[0][2]	[0][3]	[0][4]
[1][0]	[1][1]	[1][2]	[1][3]	[1][4]
[2][0]	[2][1]	[2][2]	[2][3]	[2][4]
[3][0]	[3][1]	[3][2]	[3][3]	[3][4]
[4][0]	[4][1]	[4][2]	[4][3]	[4][4]

2-rozmerné polia

- Deklarácia premennej schopnej referencovať 2-rozmerné pole, ktorého políčka sú daného typu:

```
int[][] pole;
```



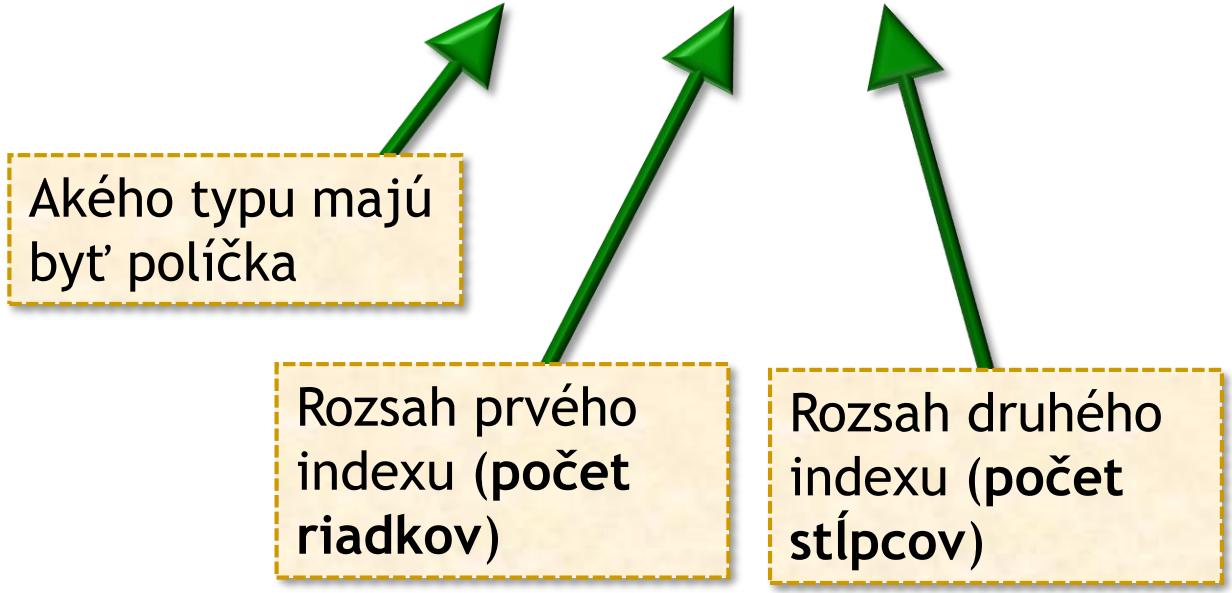

```
boolean[][] pole;
Turtle[][] pole;
```



2-rozmerné polia - vytvorenie

- Vytvorenie 2-D poľa („poľového“ objektu):

```
int[][] pole = new int[10][15];
```



Celkovo sa vyrobí **10** x **15** = **150** „`int`-ových“ políčok.



2-rozmerné polia - prístup

- Prístup k položkám 2D poľa:

```
pole[3][4] = 100;
```

Názov premennej, ktorá referencuje objekt 2D poľa

Indexy políčka v 2D poli **[riadok][stlpec]**

2-rozmerné polia - inicializácia

```
int[][] pole = {{1, 0}, {0, 1}, {1, -1}};
```

Vytvorenie a inicializácia poľa „v jednom“

Vytvorené pole je vytvorené akoby cez `new int[3][2]`

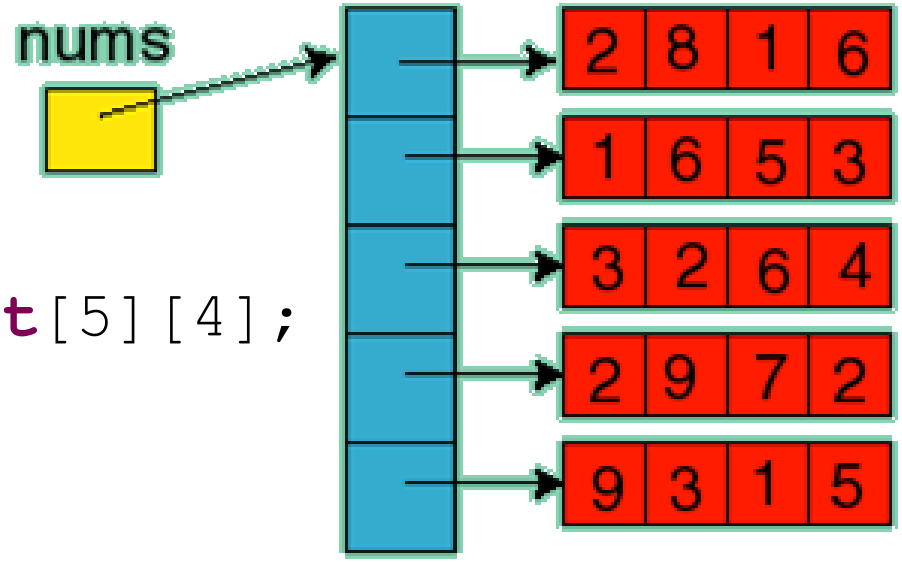
- Inicializujeme „po riadkoch“:

1	0
0	1
1	-1



Tajomstvo pre pokročilých

- V skutočnosti `int[][]` označuje premennú schopnú referencovať jednorozmerné pole, ktorého každé políčko je schopné referencovať nejaké jednorozmerné pole (riadok) s políčkami typu `int`

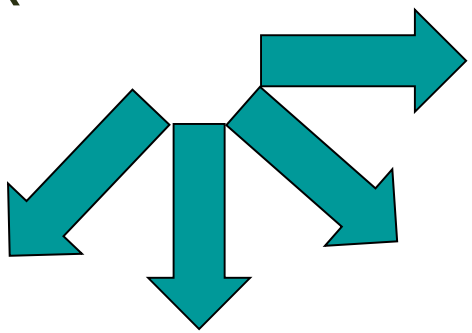



```
int[][] nums = new int[5][4];
```

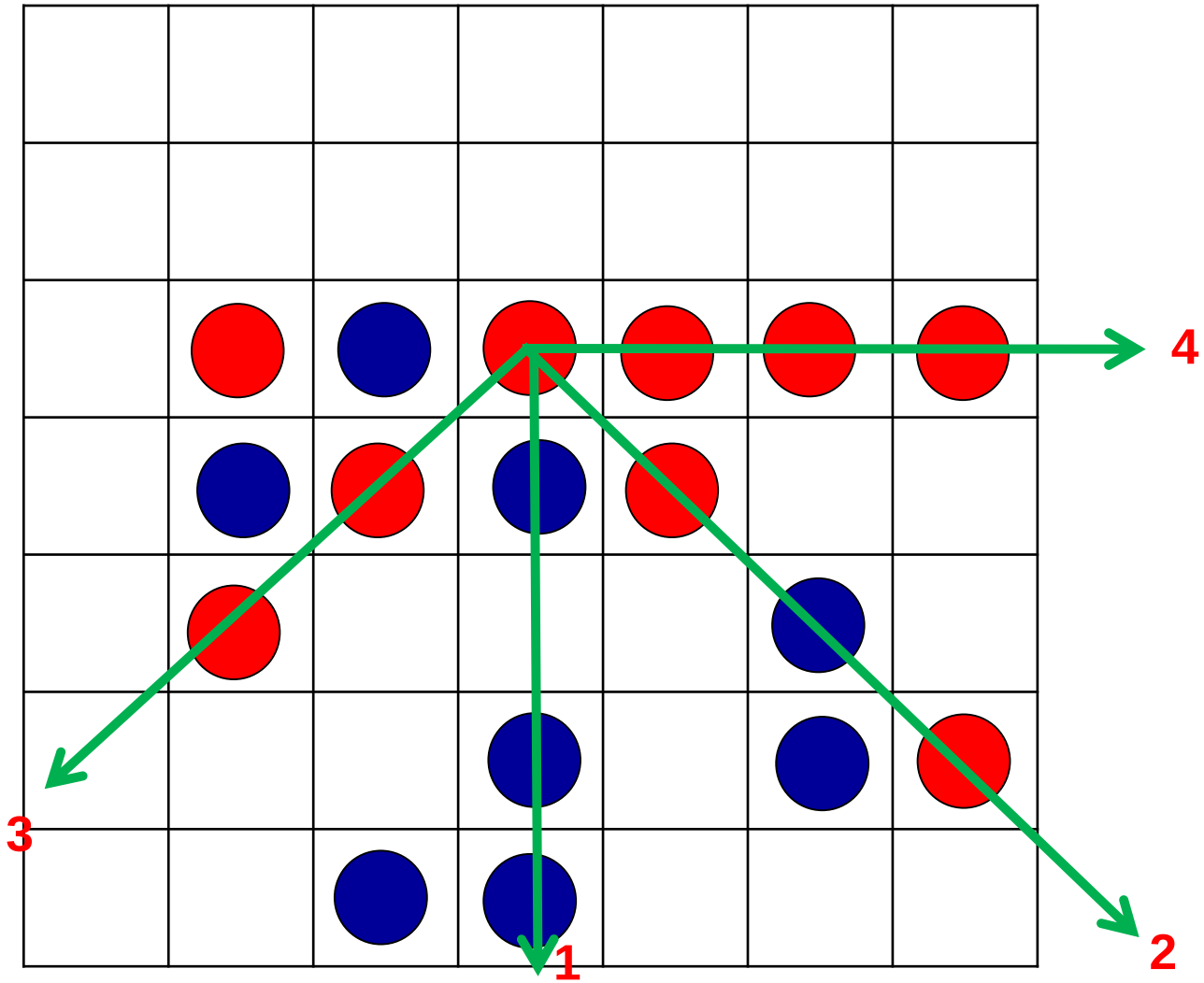


Piškvorky s 2D poľom

- Čo potrebujeme:
  - Ukladať si obsadenosť políčok `(0, 1, 2)`
  - Nedovoliť znovu obsadiť obsadené políčko
  - Zistiť, či už nemáme výhru
- **Stratégia na zistenie výhry:**
  - Pre každé políčko zistíme, či ak sa z neho vyberieme **jedným zo 4 smerov**, tak prejdeme viac ako 5 rovnakých políčok za sebou



Rovnakých v smere

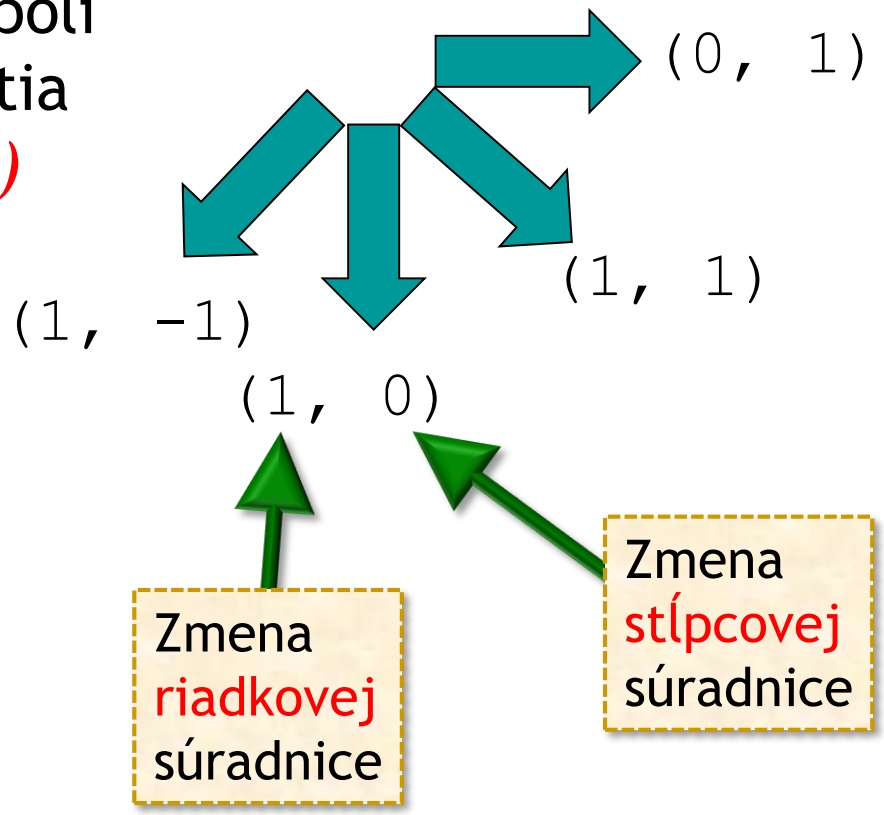


Smer: 0
Smer: 1
Smer: 2
Smer: 3

Vektory posunutia

- Pri rôznych pohyboch v 2D poli sú užitočné vektory posunutia (zmeny): *(v riadku, v stĺpci)*

(-1, -1)	(-1, 0)	(-1, +1)
(0, -1)	(0, 0)	(0, +1)
(+1, -1)	(+1, 0)	(+1, +1)



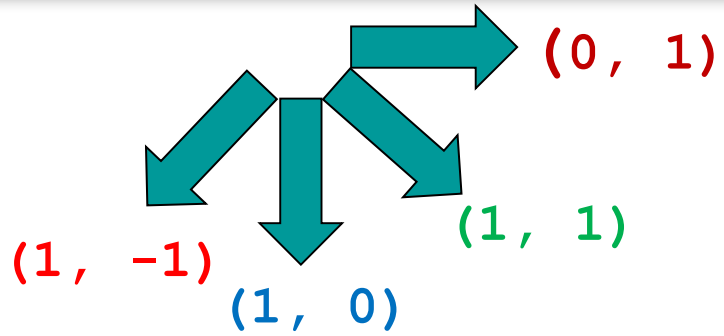


Piškvorky

- **Užitočné metódy:**
  - `jePolicko(r, s)` - povie, či políčko so súradnicami `[r][s]` existuje
  - `rovnakychVSmere(r, s, rPosun, sPosun)` - aká dlhá je postupnosť rovnakých políčok počnúc políčkom `[r][s]` a hýbajúc sa v určenom smere (v danom posune)
  - `overPolicko(r, s)` – overí, či políčko `[r][s]` je začiatkom nejakej postupnosti 5 za sebou idúcich políčok rovnakej farby
  - `overVyhru()` - overí, či aktuálne obsadenie políčok je výherné

overPolicko

Overíme, či v niektorom zo smerov už nemáme 5 kameňov rovnakej farby



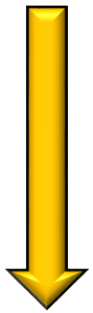

```
if (this.rovnakychVSmere(r, s, 1, -1) >= 5) {
    this.hraBezi = false;
}
if (this.rovnakychVSmere(r, s, 1, 0) >= 5) {
    this.hraBezi = false;
}
if (this.rovnakychVSmere(r, s, 1, 1) >= 5) {
    this.hraBezi = false;
}
if (this.rovnakychVSmere(r, s, 0, 1) >= 5) {
    this.hraBezi = false;
}
```

If-y sa líšia len 2 číslami.
Vieme zlepšiť kód?



overPolicko

```
if (this.rovnakychVSmere(r, s, 1, -1) >= 5) {
    this.hraBezi = false;
}
if (this.rovnakychVSmere(r, s, 1, 0) >= 5) {
    this.hraBezi = false;
}
...
```



Konkrétne „vektory“ posunutia uložíme do inicializovaného poľa … a využijeme cyklus

```
int[][] posuny =
{{1, -1}, {1, 0}, {1, 1}, {0, 1}};
```

```
for (int smer = 0; smer < posuny.length; smer++) {
 int posunR = posuny[smer][0];
 int posunS = posuny[smer][1];
 if (this.rovnakychVSmere(r, s, posunR, posunS) >= 5) …
```



Tajomstvá Javy pre pokročilých

- Zložené („kučeravé“) zátvorky:
  - ak by v kučeravých zátvorkách mal byť len jeden príkaz, tak ich nemusíme písať:

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

Menej skúseným odporúčame tieto „skratky“ zatiaľ **nepoužívať**!

- this:
  - **this.** nemusíme (takmer nikdy) písať



Sumarizácia

- **Referencie na polia** môžeme používať ako parametre metód, metóda môže vrátiť referenciu na pole („poľový objekt“)

- „Ďalší rozmer“ pre polia: **2-rozmerné polia**
  - piškvorky a iné stolové (doskové) hry
  - matice a výpočty nad nimi (násobenie matíc, redukcie matíc, ...)
  - zachytenie vzťahov a relácií (viac na PAZ1b)
  - spracovanie bitmapovej grafiky (počítačová grafika)

the end of part I.

Ďakujem za pozornosť !



