

# 8

## Viacrozmerné pole

# Pojmy zavedené v 7. prednáške<sub>(1)</sub>

- obal'ovacie triedy
  - primitívne typy ako objekty
- cykly
  - for
  - do-while
- operátory ++, --

# Pojmy zavedené v 7. prednáške<sub>(2)</sub>

- pole
  - kontajner s pevným počtom prvkov
  - definícia
  - vytvorenie a inicializácia
  - práca s poľom ako celkom
  - práca s prvkami poľa
  - dĺžka poľa - length

# Pojmy zavedené v 7. prednáške<sub>(3)</sub>

- diagramy aktivít
  - foreach
  - while
  - for
  - do-while

# Cieľ prednášky

- viacrozmerné polia
- vnorené cykly
- this
- sekcie rozhrania
- rekurzia
- príklad: Sudoku

# Sudoku<sub>(1)</sub>

9				8			5	
2	5		7			9		4
							8	6
	8		1	3				2
		6		4		1		
5				6	9		4	
3	7							
8		2			3		1	5
	1			9				3

# Sudoku<sub>(2)</sub>

- hlavolam – v každých novinách
- cieľ: čiastočne vyplnenú mriežku doplniť tak, aby obsahovala každé číslo 1 až 9 práve raz v troch rôznych zoskupeniach.

# Sudoku<sub>(3)</sub>

- mriežka 9x9 políček
- tri typy zoskupení políček mriežky
  - riadky – 9 riadkov
  - stĺpce – 9 stĺpcov
  - bloky 3x3 – 9 blokov v zostave 3x3



# Sudoku – zoskupenia políček

9				8			5	
2	5		7			9		4
							8	6
	8		1	3				2
		6		4		1		
5				6	9		4	
3	7							
8		2			3		1	5
	1			9				3

# Sudoku – 1. cieľ projektu

- podpora pre riešiteľa
- požadované funkcie:
  - zobrazenie mriežky
  - vloženie čísla do políčka
  - načítanie zadania
  - priebežná kontrola pravidiel

# Sudoku – rozhranie

## Sudoku

- + new(): Sudoku
- + vykresliMriezku(): void
- + nastavPolicko(riadok: int, stlpec: int, hodnota: int): void
- + nactajZadanie(): void

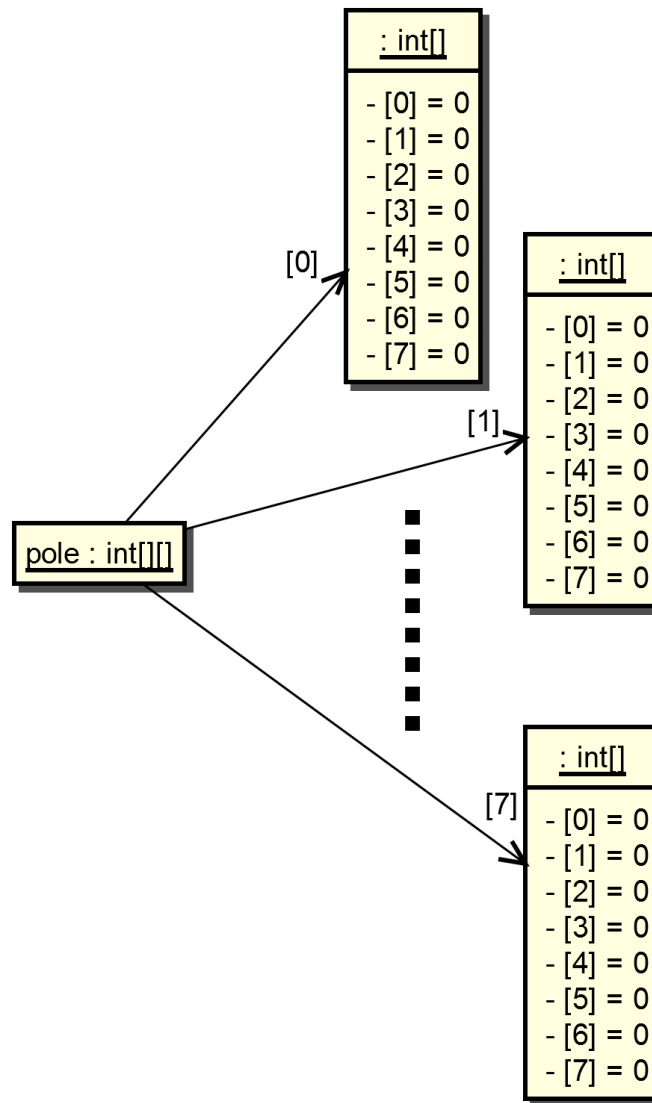
# Sudoku – mriežka

- doteraz – jednorozmerné problémy
  - Diár – zápis poznámok pod sebou
  - Analyzátor logu – hodiny idú po sebe
- Sudoku – dvojrozmerná mriežka
  - má políčka vedľa seba aj pod sebou
- podobné úlohy
  - Šach, Dáma, Skákaná – šachovnica
  - Maľované krížovky
  - Matematika – matice
  - ...

# Polia ako prvky iného poľa

- prvkom poľa môže byť objekt
  - môže byť prvkom poľa iné pole?
  - pole je objekt
- =>
- prvkami poľa môžu byť aj iné polia

# Polia ako prvky iného poľa



# Pole polí – definícia

- Java – špeciálna syntax – historické dôvody

- definícia poľa:

```
typPrvkov[] menoPola;
```

typ prvkov

- definícia poľa polí

```
typPrvkov[][] menoPola;
```

typ prvkov?

# Pole polí – vytvorenie

- Java – špeciálna syntax – historické dôvody

- vytvorenie poľa:

```
menoPola = new typPrvkov[pocetPrvkov];
```



- vytvorenie poľa polí

```
menoPola = new typPrvkov[pocetRiadkov][];
```



typ prvkov



# Polia ako prvky iného poľa

```
mriezka = new int[9][];
```

pole : int[][]

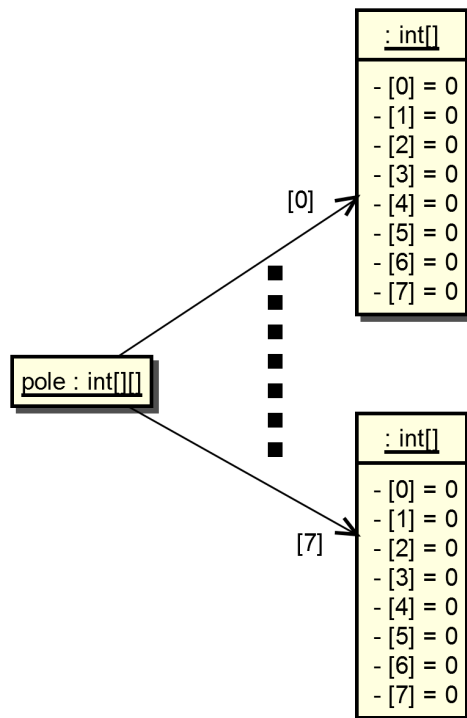
- [0] = null
- [1] = null
- [2] = null
- [3] = null
- [4] = null
- [5] = null
- [6] = null
- [7] = null

# Pole polí – inicializácia<sub>(1)</sub>

```
for (int i = 0; i < pole.length; i++) {  
    // pocetPrvkov – pocet prvkov vnoreného pola  
    pole[i] = new typPrvkov[pocetPrvkov];  
}
```

# Pole polí – inicializácia<sub>(2)</sub>

```
for (int i = 0; i < pole.length; i++) {  
    pole[i] = new typPrvkov[pocetPrvkov];  
}
```



# Pole polí

- prvok-pole
- prvky rovnakých rozmerov → obdĺžniková forma (matica)
- rozmery – riadky a stĺpce
- nepravidelné viacrozmerné polia
  - jednotlivé riadky – rôzny počet prvkov

# Pole polí – matica

<u>pole : int[][]</u>								
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
- [0]	= 0	0	0	0	0	0	0	0
- [1]	= 0	0	0	0	0	0	0	0
- [2]	= 0	0	0	0	0	0	0	0
- [3]	= 0	0	0	0	0	0	0	0
- [4]	= 0	0	0	0	0	0	0	0
- [5]	= 0	0	0	0	0	0	0	0
- [6]	= 0	0	0	0	0	0	0	0
- [7]	= 0	0	0	0	0	0	0	0

# Java – prístup k prvkom poľa polí

- pomenovanie prvkov – meno poľa + index riadku + index stĺpca

```
menoPola[indexRiadku][indexStĺpca]
```

- $0 \leq \text{index riadku} < \text{počet riadkov}$
- $0 \leq \text{index stĺpca} < \text{počet stĺpcov}$
- prvok poľa – premenná
- operácie – pravidlá pre typ prvkov
- index riadku, index stĺpca – celočíselné aritmetické výrazy

# Java – prístup k prvkom

// 3. riadok, 4. stlpec

```
System.out.println(mriezka[2][3]);
```

# Java – vytvorenie poľa polí – matica

- zjednodušená syntax

```
typPrvkov[][] menoPola  
    = new typPrvkov[pocetRiadkov][pocetStlpcov];
```

- príklad

```
int[][] mriezka = new int[9][9];
```



# n-rozmerné pole

- dvojrozmerné pole – matica

```
int[][] matica;
```

- trojrozmerné pole

```
int[][][] kocka;
```

- ...

# Sudoku – vnútorný pohľad

## Sudoku

- mriezka: int[][]

+ Sudoku()

+ vykresliMriezku(): void

+ nastavPolicko(riadok: int, stlpec: int, hodnota: int): void

+ nacistajZadanie(): void

# Reprezentácia mriežky Sudoku

- Mriežka = matica
- prvky matice – čísla 1-9
- nevyplnené hodnoty?
- náhrada prázdneho políčka číslom mimo rozsahu 1-9
- môžeme teda použiť číslo 0

# Sudoku – definícia triedy

```
public class Sudoku {  
    private int[][] mriezka;  
  
    ...  
}
```

# Sudoku – konštruktor

```
public Sudoku() {  
    this.mriezka = new int[9][9];  
}
```



matica celých čísel – núl

# Sudoku – metóda načítaj zadanie

- načíta zadanie vložené do zdrojových kódov
- testovacie sudoku

# Vytvorenie poľa polí pomocou konštanty

- vytvorenie a inicializácia poľa:

```
typPrvkov[] menoPola = {zoznamPrvkov};
```

- vytvorenie a inicializácia poľa polí:

```
typPrvkov[][] menoPola = {  
    {zoznamPrvkovPrvehoRiadku},  
    {zoznamPrvkovDruhehoRiadku},  
    ...  
};
```

# Vytvorenie poľa pomocou konštanty

- s definíciou:

```
typPrvkov[] menoPola = {zoznamPrvkov};
```

- bez definície:

```
menoPola = new typPrvkov[]{zoznamPrvkov};
```

- v prípade inicializácie s definíciou nie je new povinné – môže sa písať

```
typPrvkov[] menoPola
```

```
= new typPrvkov[]{zoznamPrvkov};
```

nepovinné





# Sudoku – metóda nacistajZadanie

```
public void nacistajZadanie() {  
    this.mriezka = new int[][] {  
        {9, 0, 0, 0, 8, 0, 0, 5, 0},  
        ...  
        {8, 0, 2, 0, 0, 3, 0, 1, 5},  
        {0, 1, 0, 0, 9, 0, 0, 0, 3}  
    };  
}
```

# Sudoku – vykreslenie mriežky

- vypísanie na konzolu
- znaky:
  - „1“-„9“: Známe hodnoty v mriežke
  - „.“: Nevyplnená hodnota

# Sudoku – metóda vykresliMriezku

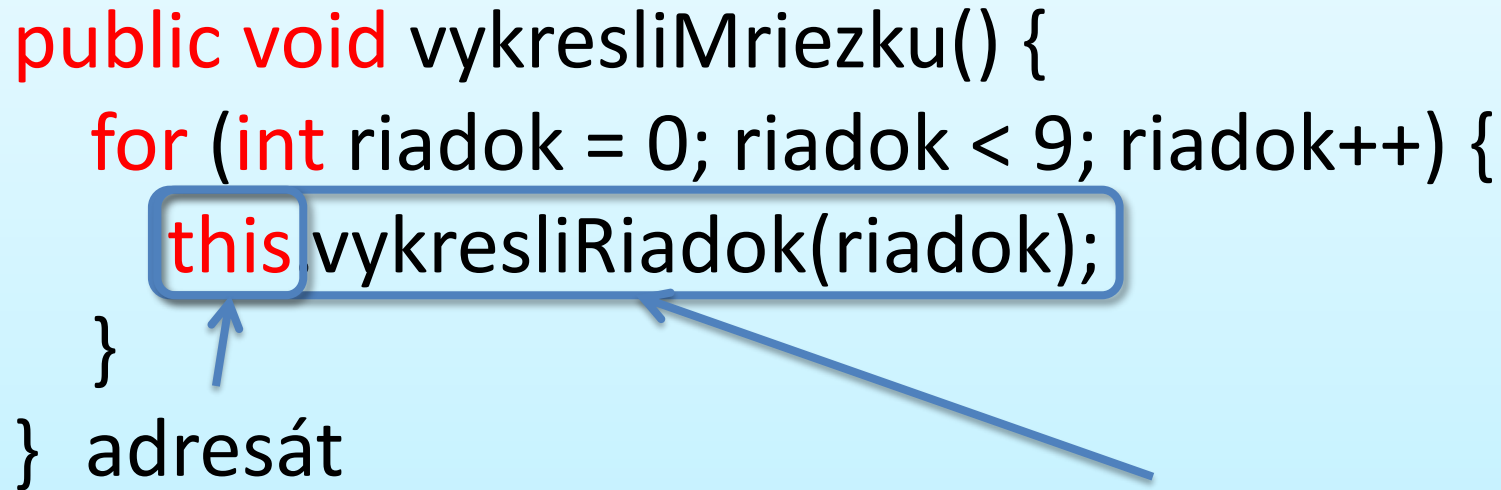
```
public void vykresliMriezku() {  
    for (int riadok = 0; riadok < 9; riadok++) {  
        this.vykresliRiadok(riadok);  
    }  
}
```

# Posielanie správ

- Digitálne hodiny – objekt celok posiela správy častiam
- projekt Sudoku – trieda Sudoku
- objekt posiela správy sám sebe
- formát správy
  - adresát.selektor(parametre)
- adresát this – kľúčové slovo
- implicitný parameter každej metódy
- objekt sám seba označuje this (self)

# explicitné použitie this

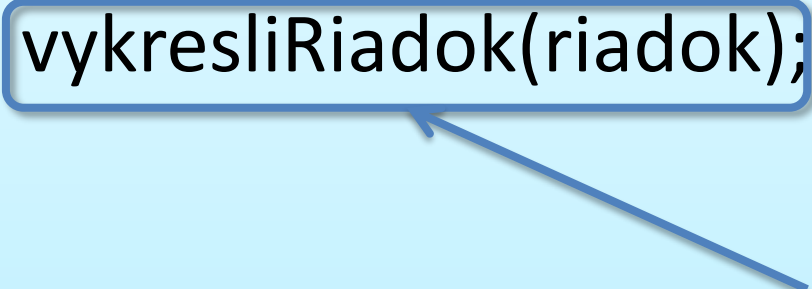
```
public void vykresliMriezku() {  
    for (int riadok = 0; riadok < 9; riadok++) {  
        this.vykresliRiadok(riadok);  
    }  
} adresát
```



objekt si posiela správu

# implicitné použitie this

```
public void vykresliMriezku() {  
    for (int riadok = 0; riadok < 9; riadok++) {  
        vykresliRiadok(riadok);  
    }  
}
```



this je podľa syntaxe nepovinné  
pozor na konvencie

# Sudoku – metóda vykresliRiadok

```
public void vykresliRiadok(int riadok) {  
    for (int stlpec = 0; stlpec < 9; stlpec++) {  
        System.out.print(this.mriezka[riadok][stlpec]);  
    }  
    System.out.println();  
}
```

# Sekcie rozhrania

- verejné – obsahuje správy, ktoré môže poslať ľubovoľný objekt
  - definícia triedy obsahuje metódy public
- neverejné – obsahuje správy, ktoré si môže poslať len objekt sám
  - definícia triedy obsahuje metódy private



## Sudoku

- mriezka: int[][]

+ Sudoku()

+ vykresliMriezku(): void

+ nastavPolicko(riadok: int, stlpec: int, hodnota: int): void

+ nacistajZadanie(): void

- vykresliRiadok(riadok: int): void

existujú neverejné správy

# Sudoku – metóda vykresliRiadok

```
private void vykresliRiadok(int riadok) {  
    for (int stlpec = 0; stlpec < 9; stlpec++) {  
        System.out.print(this.mriezka[riadok][stlpec]);  
    }  
    System.out.println();  
}
```

# Rozdeľuj a panuj

- vykresliMriezku:


```
for (int riadok = 0; riadok < 9; riadok++) {  
    this.vykresliRiadok(riadok);  
}
```

- vykresliRiadok:

```
for (int stlpec = 0; stlpec < 9; stlpec++) {  
    System.out.print(aMriezka[paRiadok][stlpec]);  
}  
System.out.println();
```

# Urob všetko sám

```
public void vykresliMriezku() {  
    for (int riadok = 0; riadok < 9; riadok++) {  
        for (int stlpec = 0; stlpec < 9; stlpec++) {  
            System.out.print(this.mriezka[riadok][stlpec]);  
        }  
        System.out.println();  
    }  
}
```



vnorený cyklus

# For-each pre viacrozmerné polia

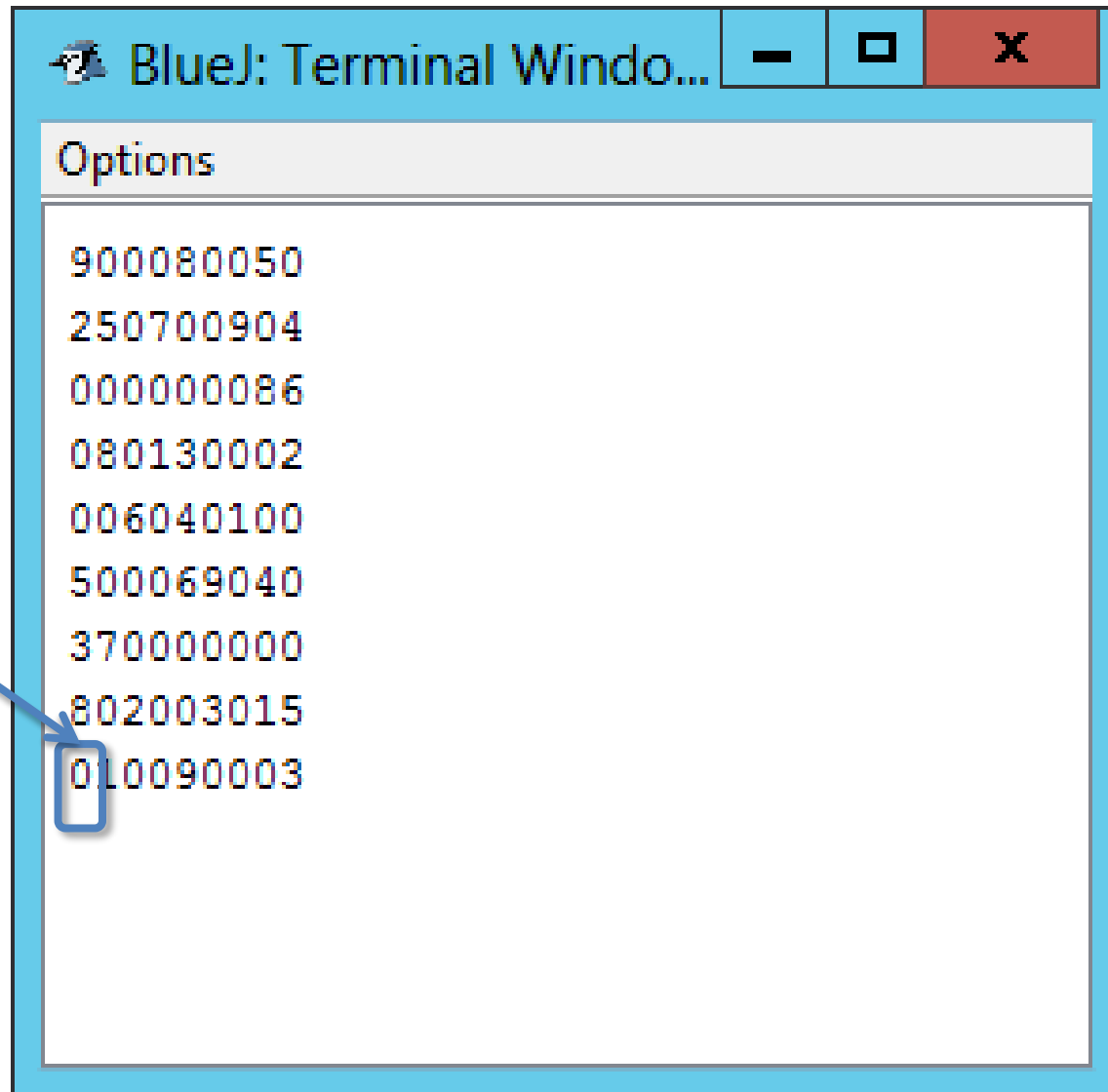
- pole je objekt
- pole je kontainer
- na prechádzanie poľa teda môžeme použiť for-each
- prvkami dvojrozmerných polí sú polia – riadky

# For-each pre viacrozmerne polia – príklad

```
public void vykresliMriezku() {  
    for (int[] riadok : this.mriezka) {  
        for (int policko : riadok) {  
            System.out.print(policko);  
        }  
        System.out.println();  
    }  
}
```

# Výsledok

mala byť  
bodka



```
BlueJ: Terminal Windo...  
Options  
900080050  
250700904  
000000086  
080130002  
006040100  
500069040  
370000000  
802003015  
010090003
```

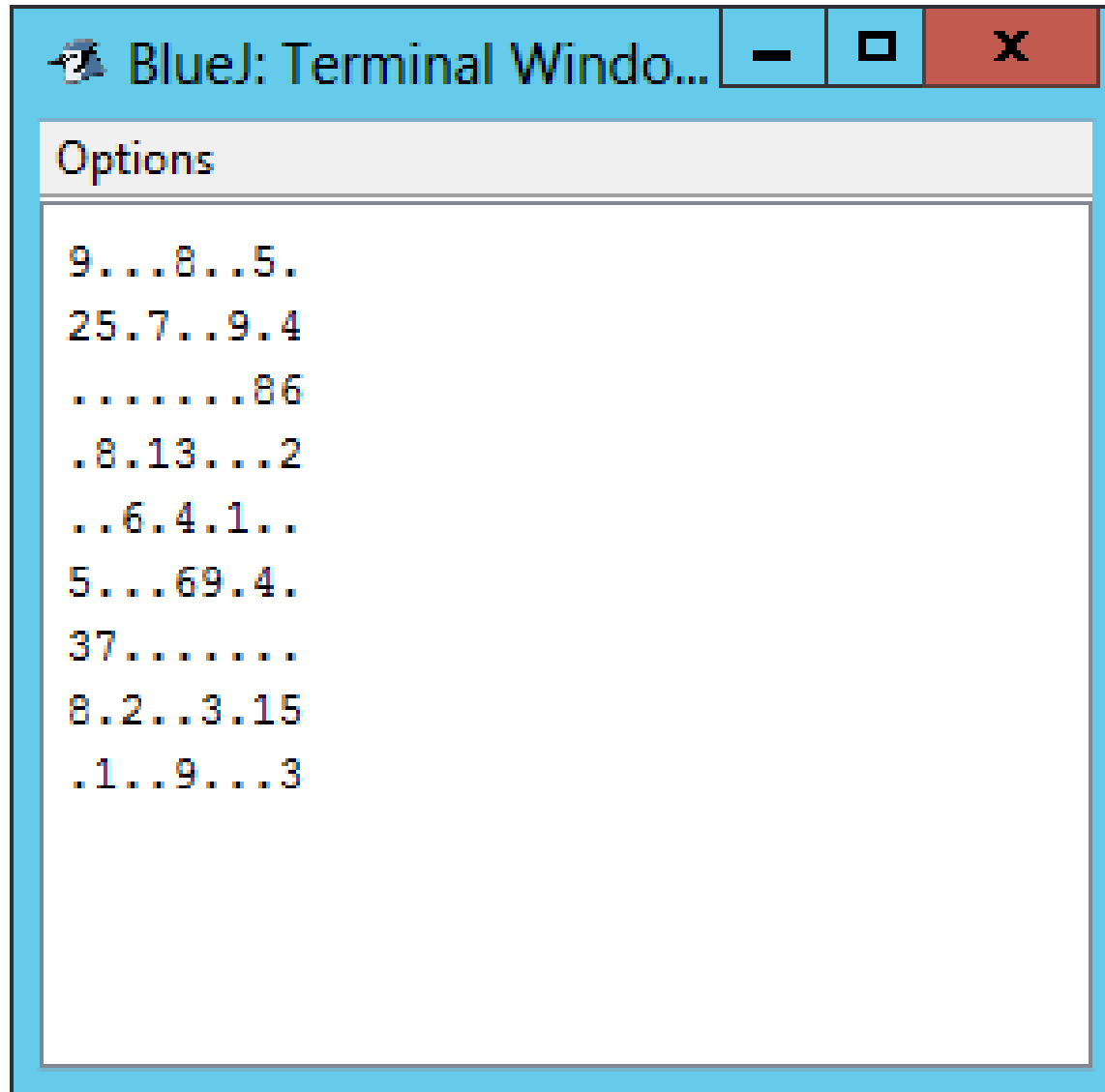
# Sudoku – metóda vykresliMriezku

```
System.out.print(policko);
```

```
if (policko == 0) {  
    System.out.print(".");  
} else {  
    System.out.print(policko);  
}
```



# Výsledok

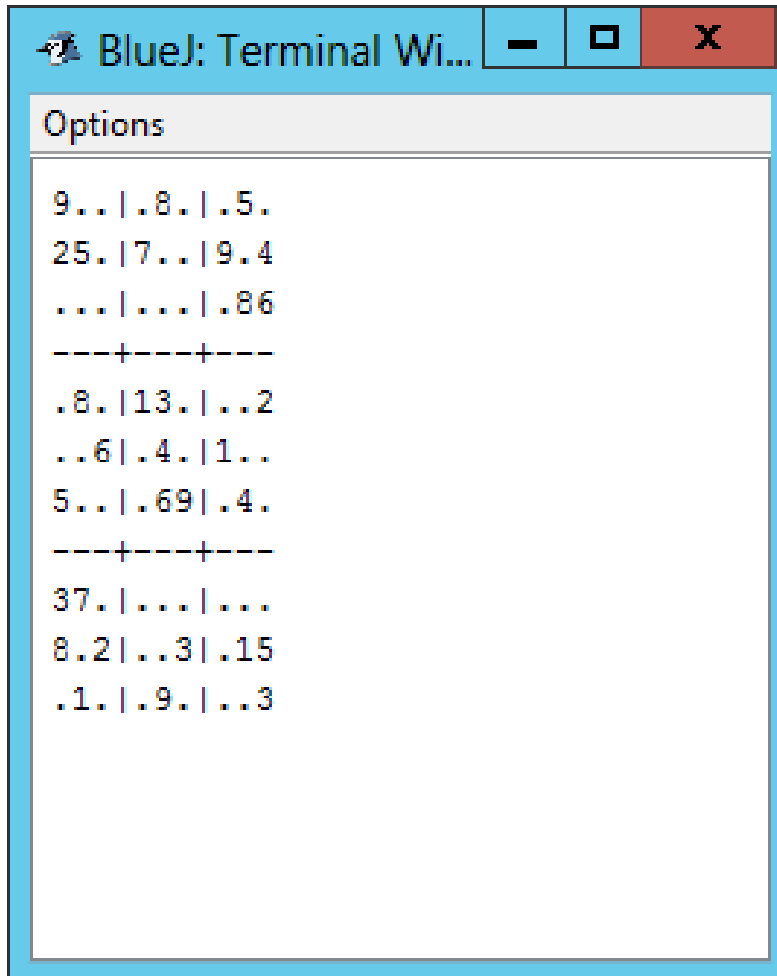


A screenshot of a BlueJ Terminal Window. The window has a title bar with a blue icon, the text "BlueJ: Terminal Windo...", and three buttons: a minus sign, a square, and a red button with a white 'X'. Below the title bar is a tab labeled "Options". The main area of the window displays a sequence of numbers on ten lines, with some digits highlighted in blue. The numbers are: 9...8..5., 25.7..9.4, .....86, .8.13...2, ..6.4.1.., 5...69.4., 37....., 8.2..3.15, .1..9...3.

```
9...8..5.  
25.7..9.4  
.....86  
.8.13...2  
..6.4.1..  
5...69.4.  
37.....  
8.2..3.15  
.1..9...3
```

# Jednoduchá úloha

- doplňte deliace čiary pre bloky sudoku



```
BlueJ: Terminal Wi...
Options
9..|.8.|.5.
25.|7..|9.4
...|...|.86
---+---+---
.8.|13.|..2
..6|.4.|1..
5..|.69|.4.
---+---+---
37.|...|...
8.2|..3|.15
.1.|.9.|..3
```


# Sudoku – metóda nastavPolicko

```
public void nastavPolicko
    (int riadok, int stlpec, int hodnota) {
    if (riadok >= 0 && riadok < 9 &&
        stlpec >= 0 && stlpec < 9 &&
        hodnota > 0 && hodnota <= 9) {
        this.mriezka[riadok][stlpec] = hodnota;
    }
}
```

# Priebežné kontroly pravidiel

- kedy kontrolovať?
- odpoveď: pri vkladaní čísla
- typy kontrol:
  - riadková – nemôžem vložiť číslo, ktoré sa už v riadku nachádza
  - stĺpcová – nemôžem vložiť číslo, ktoré sa už v stĺpci nachádza
  - bloková – nemôžem vložiť číslo, ktoré sa už v bloku nachádza

# Sudoku – kontroly

9				8			5	
2	5		7			9		4
							8	6
	8		1	3				2
		6		4		1		
5				6	9		4	
3	7							
8		2			3		1	5
	1			9				3

# Sudoku – metóda kontrolaRiadkova

```
private boolean kontrolaRiadkova
                                (int riadok, int hodnota) {
    for (int stlpec = 0; stlpec < 9; stlpec++) {
        if (this.mriezka[riadok][stlpec] == hodnota) {
            // predcasne ukoncenie cyklu
            return false;
        }
    }
    return true;
}
```

# Sudoku – metóda kontrolaStlpcova

```
private boolean kontrolaStlpcova
                                   (int stlpec, int hodnota) {
    for (int riadok = 0; riadok < 9; riadok++) {
        if (this.mriezka[riadok][stlpec] == hodnota) {
            // predcasne ukoncenie cyklu
            return false;
        }
    }
    return true;
}
```

# Sudoku – metóda kontrola

```
private boolean kontrola  
    (int riadok, int stlpec, int hodnota) {  
    return  
        this.kontrolaRiadkova(riadok, hodnota) &&  
        this.kontrolaStlpcova(stlpec, hodnota) &&  
        this.kontrolaBlokova(riadok, stlpec, hodnota);  
}
```



# Sudoku – prehľadnejšia kontrola

**private boolean** kontrola

```
        (int riadok, int stlpec, int hodnota) {  
    if (!this.kontrolaRiadkova(riadok, hodnota))  
        return false;  
    if (!this.kontrolaStlpcova(stlpec, hodnota))  
        return false;  
    if (!this.kontrolaBlokova(riadok, stlpec, hodnota))  
        return false;  
    return true;  
}
```

# Sudoku – metóda nastavPolicko

```
public void nastavPolicko
    (int riadok, int stlpec, int hodnota) {
    if (riadok >= 0 && riadok < 9 &&
        stlpec >= 0 && stlpec < 9 &&
        hodnota > 0 && hodnota <= 9 &&
        this.kontrola(riadok, stlpec, hodnota)) {
        this.mriezka[riadok][stlpec] = hodnota;
    }
}
```

# Opakovanie kódu bez cyklu?

- rekurzia – iný spôsob opakovania
- z matematiky – rekurzívny zápis postupnosti/funkcie
- $a_n = a_{n-1} + d$ , pre ľubovoľné  $a_0$  a  $d$
- $$n! = \begin{cases} 1 & \text{pre } n = 0 \\ n \times (n-1)! & \text{pre } n > 0 \end{cases}$$
- $$\binom{n}{k} = \begin{cases} \binom{n-1}{k-1} + \binom{n-1}{k} & \text{pre } k \in \langle 0, n \rangle \\ 0 & \text{inak} \end{cases}$$

# Faktoriál Java

```
public int faktorial(int n) {  
    if (n < 1) {  
        return 1;  
    } else {  
        return n * this.faktorial(n - 1);  
    }  
}
```

# Vykonávanie rekurzie

matematika.faktorial(4)

výsledok: 24

Zásobník:

n = 4      return 24

n = 3      return 6

n = 2      return 2

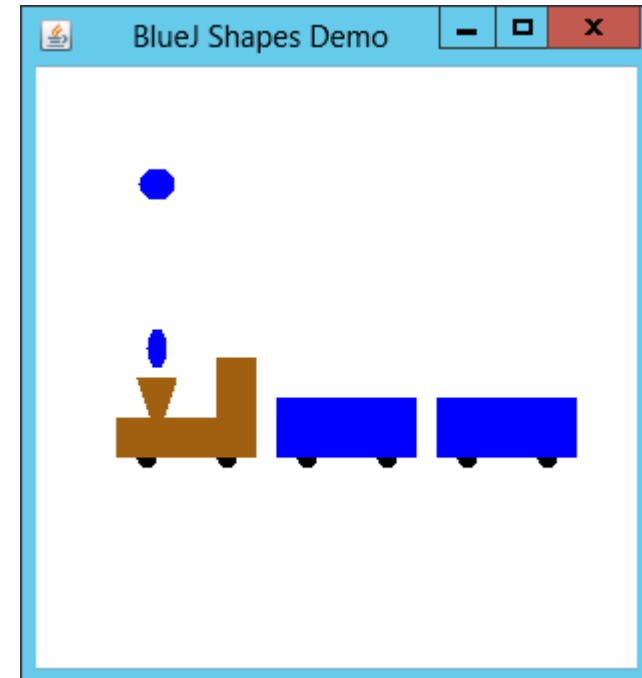
n = 1      return 1

n = 0      return 1

```
public int faktorial(int n) {  
    ➔ if (n < 1) {  
        ➔ return 1;  
    } else {  
        ➔ return n * this.faktorial(n - 1);  
    }  
}
```

# Rekurzia v $OP_{(1)}$

- Vlak = rušeň + vagón
- rušeň – posunutie dopredu
  - posuň vagón
- vagón – posunutie dopredu
  - posuň nasledujúci vagón



# Rekurzia v $OP_{(2)}$

- Osoba si pamätá otca
- získanie odkazu na Adama

```
public Osoba dajAdama() {  
    if (this.otec == null) {  
        return this;  
    } else {  
        return this.otec.dajAdama();  
    }  
}
```

# Vďaka za pozornosť