A0B36PRI - PROGRAMOVÁNÏ

Pole

Jazyk JAVA

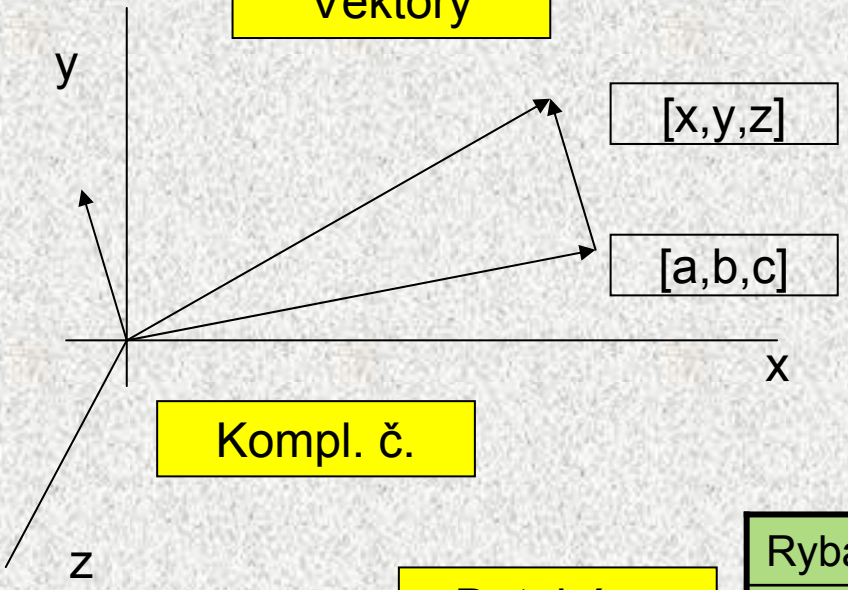
České vysoké učení technické Fakulta elektrotechnická

v1.01

Obsah

Pole

Vektory



Kompl. č.

Matice

$$\begin{bmatrix} 3 & 45 & 0 & 5 \\ f & 3 & 5 & 1 \\ 5 & 5 & 0 & 7 \\ 4 & 4 & 33 & 3 \end{bmatrix}$$

Databáze

Ryba							
Žába							
Tchoř							
Muflon							
Veverka							
Husa							

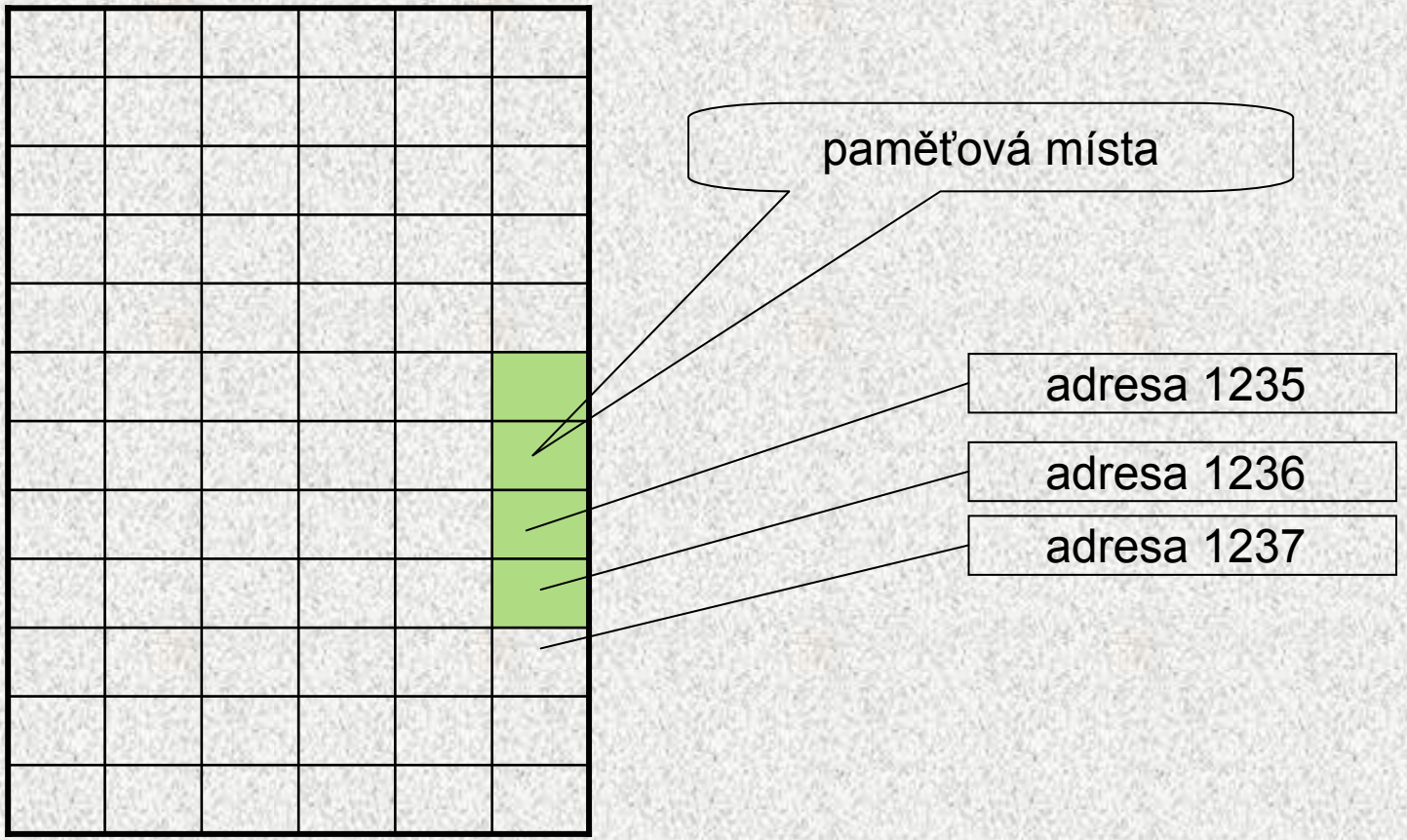
Řazení

Třídění

Zpracování řetězců znaků

Pole

- Abstrakce paměti počítače



Pole

- Strukturovaný typ
  - skupina proměnných stejného typu
  - přístup k jednotlivým proměnným pomocí indexu
  - práce s polem jak s celkem či s jednotlivými složkami – proměnnými pole

- Příklady:
  - souřadnice bodu, komplexní číslo
  - n-rozměrný vektor
  - matice – pole polí
  - řetězec znaků
- Referenční typ (druhým referenčním typem je objekt!)

Pole

- Příklad: přečíst teploty naměřené v jednotlivých dnech týdnu, vypočítat průměrnou teplotu a pro každý den vypsat odchylku od průměrné teploty
- Řešení s proměnnými typu *int*:

```
int t1, t2, t3, t4, t5, t6, t7, prumer;
t1=sc.nextInt();
...
t7=sc.nextInt();
prumer = (t1+t2+t3+t4+t5+t6+t7)/7;
System.out.println(t1-prumer);
...
System.out.println(t7-prumer);
```

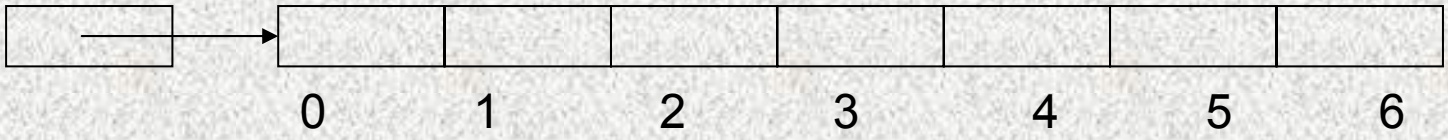
Řešení je těžkopádné a bylo by ještě horší, kdyby vstupními daty byly teploty za měsíc nebo za celý rok

Pole

- Příklad vyřešíme pomocí pole
- Pole je obecně strukturovaný datový typ skládající se z pevného počtu složek (prvků) stejného typu, které se vzájemně rozlišují pomocí indexu
- V jazyku Java se pole indexuje čísly *0, 1,* ... *počet prvků – 1,*

kde *počet prvků* je dán při vytvoření pole

teploty



pro uložení teplot vytvoříme pole obsahující 7 prvků typu *int*

```
int teploty[] = new int[7];
```

první prvek pole má označení `teploty[0]`, druhý `teploty[1]` atd.

Pole

Řešení pomocí pole

- vstupní data přečteme a do prvků pole uložíme cyklem

```
for (int i=0; i<7; i++)
    teploty[i] = sc.nextInt();
```

- průměrnou teplota: součet prvků pole dělený 7

```
int prumer = 0;
for (int i=0; i<7; i++)
    prumer = prumer + teploty[i];
prumer = prumer / 7;
```

- na závěr pomocí cyklu vypíšeme odchylky od průměru

```
for (int i=0; i<7; i++)
    System.out.println (teploty[i]-prumer);
```

Pole v jazyku Java

1. Pole *p* obsahující *n* prvků typu *T* vytvoříme deklarací
   - `T p[] = new T[n];`
   - `T[] p = new T[n];`
   - kde *T* může být libovolný typ a *n* musí být celočíselný výraz s nezápornou hodnotou;
- prvky takto zavedeného pole mají nulové hodnoty

2. Inicializované pole
   - pole lze zavést definicí hodnot prvků pole

```
int p[] = {1,2,3,4,5,6};
```

Pole v jazyku Java

- Zápis **`p[i]`**
  - *i* je celočíselný výraz
    - hodnota je nezáporná
    - menší než počet prvků,
  - označuje prvek pole *p* s indexem *i*
  - má vlastnosti proměnné typu *T*;
  - Pozn.: nedovolená hodnota indexu způsobí chybu při výpočtu

    `java.lang.ArrayIndexOutOfBoundsException: 7`
- Počet prvků pole *p* lze zjistit pomocí zápisu

  `p.length`

Příklad použití:

```
for (int i=0; i<p.length; i++) System.out.print(p[i]);
```

Pole - příklad

- Vstup: $n\ a_1\ a_2\ \ldots\ a_n$ kde a_i jsou celá čísla
- Výstup: čísla a_i v opačném pořadí
- Řešení:

```
package pri06;
import java.util.*; // Scanner je v knihovně java.util

public class ObratPole1 {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.println("zadejte počet čísel");
    int[] pole = new int[sc.nextInt()];
    System.out.println("zadejte "+pole.length+" čísel");
    for (int i=0; i<pole.length; i++)
      pole[i] = sc.nextInt();
    System.out.println("výpis čísel v obráceném pořadí");
    for (int i=pole.length-1; i>=0; i--)
      System.out.println(pole[i]);
  }
}
```

Pole – přidělení paměti

- Všimněme si podrobněji mechanismu vytváření a přístupu k polím v jazyku Java
- Uvažujme např. lokální deklaraci, která vytvoří pole 3 prvků typu *int*

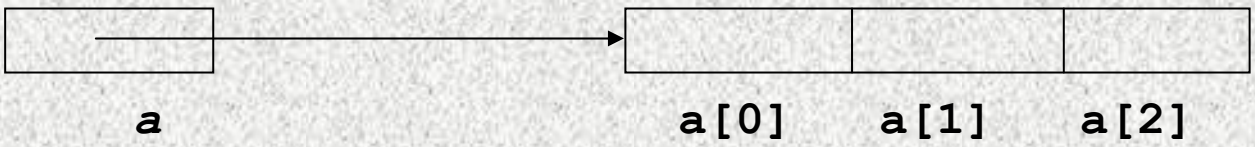
```
int[] a = new int [3];
```

Deklarace má tento efekt:

  - lokální proměnné *a* se přidělí paměťové místo na zásobníku, které však neobsahuje prvky pole, ale je dimensováno na uložení čísla reprezentujícího adresu jiného paměťového místa
  - operátorem *new* se v jiné paměťové oblasti rezervuje (alokuje) úsek potřebný pro pole 3 prvků typu *int*
  - adresa tohoto úseku se uloží do *a*

- Při grafickém znázornění reprezentace v paměti místo adres kreslíme šipky

deklarovaná proměnná pole vytvořené operátorem *new*



Poznámka: reprezentace pole je zjednodušená, obsahuje ještě počet prvků

Referenční proměnná

- Shrnutí:
  - pole *n* prvků typu *T* lze v jazyku Java vytvořit pouze dynamicky pomocí operace *new T*[*n*]
  - adresu dynamicky vytvořeného pole prvků typu *T* lze uložit do proměnné typu *T*[ ]; takovou proměnnou nazýváme referenční proměnnou pole prvků typu *T*

- Referenční proměnnou pole lze deklarovat bez vytvoření pole; deklarací
  ```
  int[] a;
  ```
  se zavede referenční proměnná, která má nedefinovanou hodnotu, jde-li o lokální proměnnou, nebo speciální hodnotu *null*, která nereferencuje žádné pole, jde-li o statickou proměnnou třídy

- V obou předchozích případech je třeba, před dalším použitím referenční proměnné pole, jí přiřadit referenci na vytvořené pole, např. příkazem
  ```
  a = new int [10];
  ```

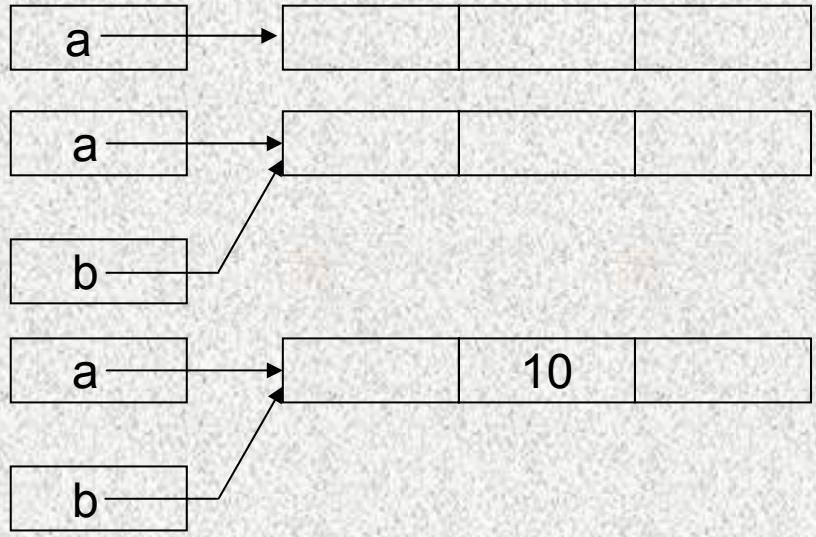
Přiřazení mezi referenčními proměnnými pole

- Po přiřazení pak obě proměnné referencují totéž pole!
- Příklad:

```
int[] a = new int[3];
int[] b = a;

b[1] = 10;

System.out.println(a[1])
// vypíše se 10
```



- Přiřazení hodnot mezi dvěma poli není v jazyku Java definováno:

```
b=new int[a.length];
for (int i=0; i<a.length; i++)b[i] = a[i];
// kopírování,  též System.arrayCopy()
```

Pole – vstupní a návratová hodnota funkce

- Reference pole může být parametrem funkce i jejím výsledkem
- Příklad:

```
package alg6;


public class ObratPole2 {
  public static void main(String[] args) {
    int[] vstupniPole = ctiPole();
    int[] vystupniPole = obratPole(vstupniPole);
    vypisPole(vystupiPole);
  }

  static int[] ctiPole() { ... }
  static int[] obratPole(int[] pole) { ... }
  static void vypisPole(int[] pole) { ... }

}
```

Pole – vstupní a návratová hodnota funkce

```
static int[] ctiPole() {
    Sys.pln("zadejte počet čísel");
    int[] pole = new int [Sys.readInt()];
    Sys.pln("zadejte "+pole.length+" čísel");
    for (int i=0; i<pole.length; i++)
      pole[i] = Sys.readInt();
    return pole;
  }

  static int[] obratPole(int[] pole) {
    int[] novePole = new int [pole.length];
    for (int i=0; i<pole.length; i++)
      novePole[i] = pole[pole.length-1-i];
    return novePole;
  }

  static void vypisPole(int[] pole) {
    for (int i=0; i<pole.length; i++)
      Sys.pln(pole[i]);
  }
```

Změna pole daného parametrem

Touto metodou nevytvoříme nové pole, ale obrátíme pole dané parametrem

```
static void obratPole(int[] pole) {
   int pom;
   for (int i=0; i<pole.length/2; i++) {
     pom = pole[i];
     pole[i] = pole[pole.length-1-i];
     pole[pole.length-1-i] = pom;
   }
}
```

- Použití:

```
public static void main(String[] args) {
   int[] vstupniPole = ctiPole();
   obratPole(vstupniPole);
   vypisPole(vstupniPole);
}
```

- To funguje tak, že metoda *obratPole* dostane referenci na stejné pole, jaké referencuje proměnná *vstupniPole*

Pole jako tabulka

- Předchozí příklady ilustrovaly použití pole pro uložení posloupnosti
- Pole lze použít též pro realizaci tabulky (zobrazení), která hodnotám typu indexu (v jazyku Java to je pouze interval celých čísel počínaje nulou) přiřazuje hodnoty nějakého typu
- Příklad: přečíst řadu čísel zakončených nulou a vypsat tabulku četnosti čísel od 1do 100 (ostatní čísla ignorovat)
- Tabulka četnosti bude pole 100 prvků typu *int*, počet výskytů čísla *x*, kde $1 \leq x \leq 100$, bude hodnotou prvku s indexem *x*-1
- Aby program byl snadno modifikovatelný pro jiný interval čísel, zavedeme dvě konstanty:

```
final static int MIN = 1;
  final static int MAX = 100;
```

a pole vytvoříme s počtem prvků *Max*-*Min*+1

```
int[] tab = new int[MAX-MIN+1];
```

Tabulka četnosti čísel

- Příklad:
- Funkce, která přečte čísla a vytvoří tabulku četnosti:

```
static int[] tabulka() {
  int[] tab = new int[MAX-MIN+1];
  Sys.pln("zadejte řadu celých čísel zakončenou nulou");
  int cislo = Sys.readInt();
  while (cislo!=0) {
    if (cislo>=MIN && cislo<=MAX) tab[cislo-MIN]++;
    cislo = Sys.readInt();
  }
  return tab;
}
```

- Funkce, která tabulku četnosti vypíše:

```
static void vypis(int[] tab) {
  for (int i=0; i<tab.length; i++)
    if (tab[i]!=0)
      Sys.pln("četnost čísla "+(i+MIN)+" je "+tab[i]);
}
```

Tabulka četnosti čísel

- Celkové řešení:

```
public class CetnostCisel {

  final static int MIN = 1;
  final static int MAX = 100;

  public static void main(String[] args) {
    vypis(tabulka());
  }

  static int[] tabulka() {
    ...
  }

  static void vypis(int[] tab) {
    ...
  }
}
```

Pole reprezentující množinu

- Příklad: vypsat všechna prvočísla menší nebo rovna zadanému *max*
- Algoritmus:
  1. Vytvoříme množinu obsahující všechna přirozená čísla od 2 do *max*.
  2. Z množiny vypustíme všechny násobky čísla 2.
  3. Najdeme nejbližší číslo k tomu, jehož násobky jsme v předchozím kroku vypustili, a vypustíme všechny násobky tohoto čísla.
  4. Opakujeme krok 3, dokud číslo, jehož násobky jsme vypustili, není větší než odmocnina z *max*.
  5. Čísla, která v množině zůstanou, jsou hledaná prvočísla.
- Pro reprezentaci množiny čísel použijeme pole prvků typu *boolean*
  - prvek *mnozina*[*x*] bude udávat, zda číslo *x* v množině je (true) nebo není (false)

Eratosthenovo síto

Příklad:

- Funkce pro vytvoření množiny prvočísel do *max*

```
static boolean[] sito(int max) {
   boolean[] mnozina = new boolean[max+1];
   for (int i=2; i<=max; i++) mnozina[i] = true;
   int p = 2;
   int pmax = (int)Math.sqrt(max);
   do {
     // vypuštění všech násobků čísla p
     for (int i=p+p; i<=max; i+=p) mnozina[i] = false;
     // hledání nejbližšího čísla k p
     do {
       p++;
     } while (!mnozina[p]);
   } while (p<=pmax);
   return mnozina;
 }
```

Eratosthenovo síto

- Funkce pro výpis množiny

```
static void vypis(boolean[] mnozina) {
   for (int i=2; i<mnozina.length; i++)
     if (mnozina[i]) Sys.pln(i);
 }
```

- Hlavní funkce

```
public static void main(String[] args) {
   Sys.pln("zadejte max");
   int max = Sys.readInt();
   boolean[] mnozina = sito(max);
   Sys.pln("prvočísla od 2 do "+max);
   vypis(mnozina);
 }
```

Vícerozměrné pole

- Vícerozměrným polem se obecně rozumí takové pole, k jehož prvkům se přistupuje pomocí více než jednoho indexu
- V jazyku Java se s vícerozměrnými poli pracuje jako s poli, jejichž prvky jsou opět pole
- Příklady dvojrozměrného pole prvků typu *int*:
  - deklarace referenční proměnné

```
int mat[][];
```

  - vytvoření pole se 3 x 4 prvky (3 řádky, 4 sloupce)

```
mat = new int[3][4]; // prvky mají hodnotu 0
```

  - deklarace referenční proměnné a vytvoření pole 3 x 4 inicializací

```
int mat[][] = {{1,2,3,4}, {5,6,7,8}, {9,10,11,12}};
```

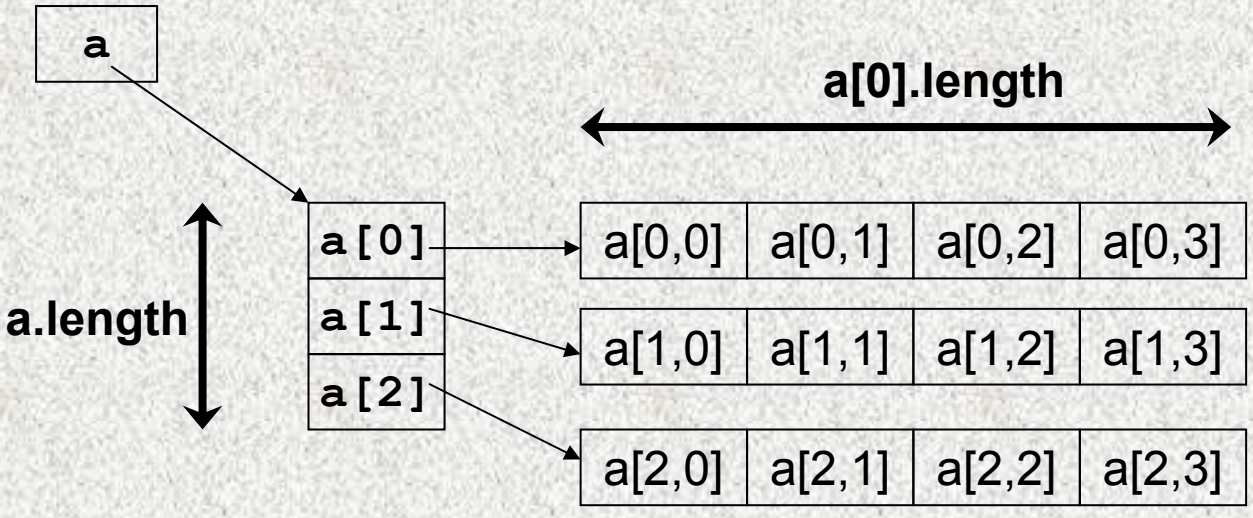
  - součet všech prvků pole *mat*

```
int suma = 0;
for (int i=0; i<mat.length; i++)
   for (int j=0; j<mat[i].length; j++)
     suma += mat[i][j];
```

Vícerozměrné pole

V jazyku Java se s vícerozměrnými poli pracuje jako s poli, jejichž prvky jsou opět pole

```
int[][] a = new int[3][4];
```



Součet matic

- Vstupní data:

  r s kde *r* je počet řádků a *s* je počet sloupců matice

  prvky první matice *r* x *s*

  prvky druhé matice *r* x *s*
- Výstup:

  součet matic
- Hlavní funkce:

```
public class Matice {
  public static void main(String[] args) {
    Sys.pln(
           "zadejte počet řádků a počet sloupců matice");
    int r = Sys.readInt();
    int s = Sys.readInt();
    int[][] m1 = ctiMatici(r, s);
    int[][] m2 = ctiMatici(r, s);
    int[][] m3 = soucetMatic(m1, m2);
    Sys.pln("součet matic");
    vypisMatice(m3);
  }
```

Součet matic

- Funkce pro čtení matice:

```
static int[][] ctiMatici(int r, int s) {
   int[][] m = new int[r][s];
   Sys.pln("zadejte celočíslenou matici "+r+"x"+s);
   for (int i=0; i<r; i++)
     for (int j=0; j<s; j++)
       m[i][j] = Sys.readInt();
   return m;
 }
```

- Funkce pro výpis matice:

```
static void vypisMatice(int[][] m) {
   for (int i=0; i<m.length; i++) {
     for (int j=0; j<m[i].length; j++)
       Sys.p(m[i][j]+" ");
     Sys.pln();
   }
 }
```

Součet matic

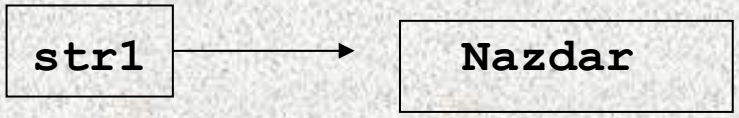
- Funkce pro součet matic:

```
static int[][] soucetMatic(int[][] m1, int[][] m2) {
   int r = m1.length;
   int s = m1[0].length;
   int[][] m = new int[r][s];
   for (int i=0; i<r; i++)
     for (int j=0; j<s; j++)
       m[i][j] = m1[i][j]+m2[i][j];
   return m;
 }
```

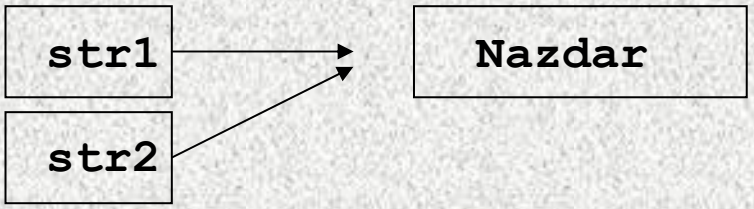
Řetězec znaků - String

- Objekty knihovní třídy *String* jsou řetězy znaků
- Od ostatních tříd se liší třemi specialitami:
  - objekt typu *String* lze vytvořit literálem
    (posloupnost znaků uzavřená mezi uvozovky)
  - hodnotu objektu typu *String* nelze jakkoli změnit
  - operací konkatenace čili zřetězení je nejen realizována metodou concat, ale i přetíženým operátorem +
- Příklady referečních proměnných typu *String:*

```
String str1 = "Nazdar";
```




```
String str2 = str1;
```



Operace s řetězci znaků

- Spojení řetězů (konkatenace) +
- Příklad:
  `"abc" + "123"` výsledek je `"abc123"`
- Jestliže jeden operand operátoru + je typu *String* a druhý je jiného typu, pak druhý operand se převede na typ *String* a výsledkem je konkatenace řetězů
- Příklady:
  `"abc" + 5` výsledek je `"abc5"`
  `"a" + 1 + 2` výsledek je `"a12"`
  `"a" + 1 + (-2)` výsledek je `"a1-2"`
- Porovnávání řetězů
  - relační operátory == a != porovnávají reference, nikoliv obsah řetězů
  - pro porovnání řetězů na rovnost slouží metoda *equals*

```
String s1 = "abcd" ;
String s2 = "ab" ;
String s3 = s2 + "cd" ;
Sys.pln(s1==s3);                    // vypíše false
Sys.pln(s1.equals(s3));             // vypíše true
```

Operace s řetězci znaků

- Pro lexikografické porovnání řetězů slouží metoda *compareTo*:

```
String s = "abcd";
Sys…print(s1.compareTo( "abdc" ));      // vypíše -1
Sys…print(s1.compareTo( "abcd" ));      // vypíše 0
Sys…print("abdc".compareTo(s1));        // vypíše 1
```

- Některé další operace:

```
String s = "nazdar";
int delka = s.length();             // délka je 5
char znak = s.charAt(1);            // znak je 'a'
String ss = s.substring(2,4);       // ss je "zd"
int z1 = s.indexOf('a');            // z1 je 1
int z2 = s.lastIndexOf('a');        // z2 je 4
int z3 = s.lastIndexOf('A');        // z3 je -1
```

- Hodnotu referenční proměnné typu *String* lze změnit (odkazuje pak na jiný řetěz), vlastní řetěz změnit nelze

Palindrom

- Napišme program, který přečte jeden řádek a zjistí, zda se po vynechání mezer jedná o palindrom (čte se stejně zpředu jako zezadu, např. “kobyla ma maly bok”)
- Řešení – funkce s parametrem typu *String* a výsledkem typu *boolean*:

```
static boolean jePalindrom(String str) {
   int i = 0, j = str.length()-1;
   while (i<j) {
     while (str.charAt(i)==' ') i++;
     while (str.charAt(j)==' ') j--;
     if (str.charAt(i)!=str.charAt(j))return false;
     i++; j--;
   }
   return true;
 }
```

Palindrom

- Výsledný program:

```
public class Palindrom {
  public static void main(String[] args) {
    Sys.pln( "Zadejte jeden řádek" );
    String radek = Sys.readLine();
    String vysl;
    if (jePalindrom(radek)) vysl = "je" ;
    else vysl = "není" ;
    Sys.pln( "Na řádku " +vysl+ " palindrom" );
  }

  static boolean jePalindrom(String str) {
    ...
  }
}
```

Pole znaků a řetězec

- Příklad: funkce pro převod celého čísla na řetěz tvořený zápisem čísla v hexadecimální soustavě

```
final static String hexa = "0123456789abcdef";

static String hexadecimal(int x) {
  if (x==0) return "0" ;
  char[] znaky = new char[9];
  int y;
  if (x<0) y=-x; else y=x;
  int prvni = 9;
  do {
    prvni--;
    znaky[prvni] = hexa.charAt(y%16);
    y = y / 16;
  } while (y>0);
  if (x<0) {
    prvni--; znaky[prvni] = '-';
  }
  return new String(znaky, prvni, 9-prvni);
}
```

Strukturované datové typy

Jazyk JAVA

Konec

České vysoké učení technické Fakulta elektrotechnická