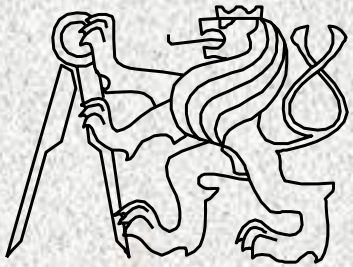
Pole

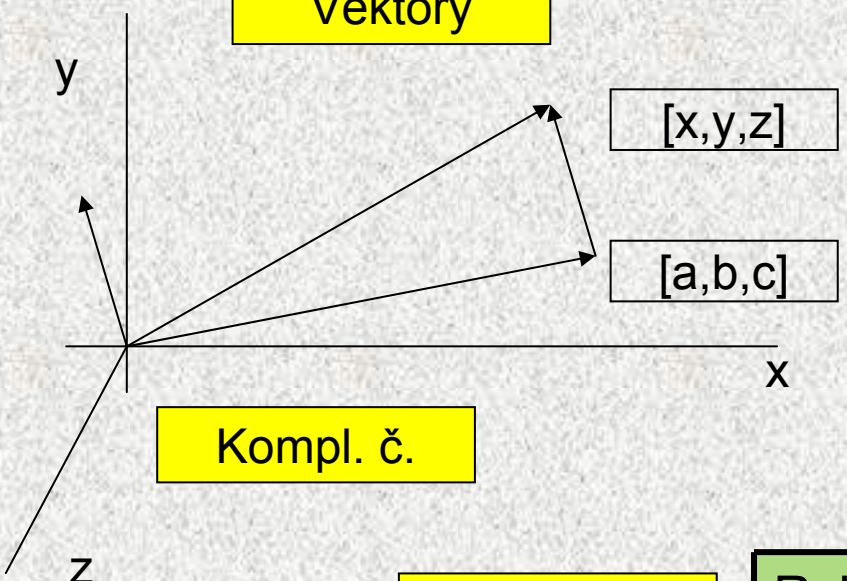


A0B36PR1-Programování 1

Fakulta elektrotechnická
České vysoké učení technické

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							

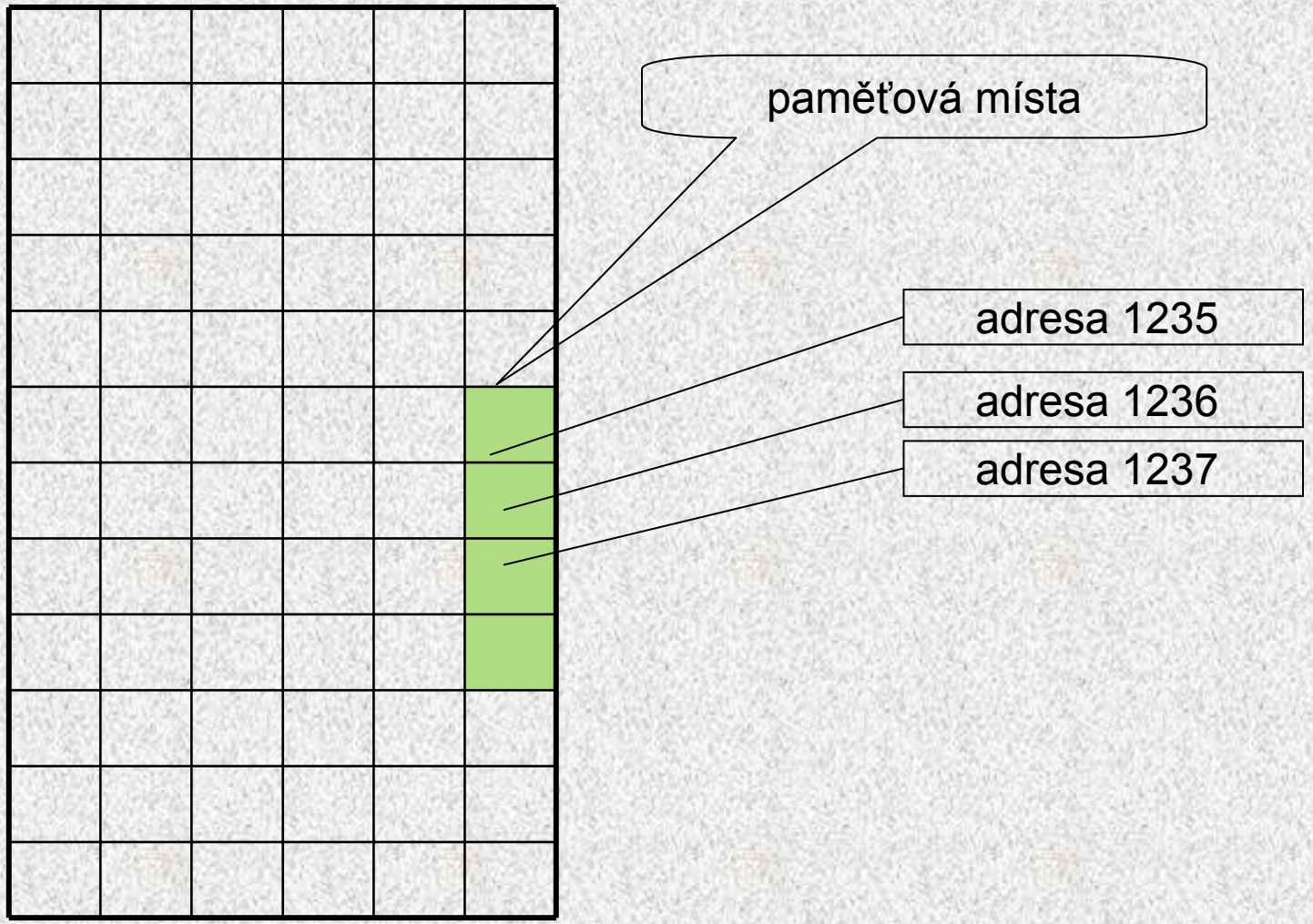
Řazení

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);
```

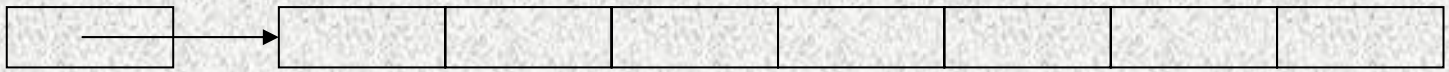
Řešení je těžkopádné a bylo by ještě horší, kdyby vstupními daty byly teploty za měsíc nebo za celý rok

Pole, řešení

- 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`



0 1 2 3 4 5 6

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

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);
```

Deklarace pole a přidělení paměti poli

- Deklarace má tento efekt:
  1. lokální proměnné **a** se přidělí paměťové místo (na zásobníku), které však neobsahuje prvky pole, ale odkaz (číslo reprezentujícího adresu jiného paměťového místa (!)) na prvky pole
  2. operátorem *new* se v jiné paměťové oblasti (haldě) rezervuje (přiděluje, alokuje) úsek potřebný pro pole (např. 3 prvků typu *int*)
  3. reference (něco jako adresa) tohoto úseku se uloží do *a*

Poznámka: pole v Javě nemusí být souvislý úsek paměti

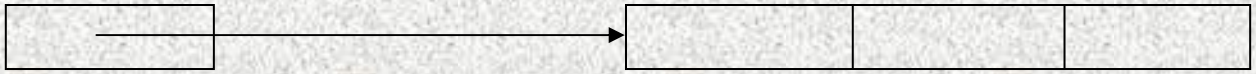
- Uvažujme např. lokální deklaraci, která vytvoří pole 3 prvků typu *int:*

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

Přidělení paměti poli

- Při grafickém znázornění reprezentace v paměti místo adres kreslíme šipky

deklarovaná proměnná — pole přidělené operátorem **`new`**



a a[0] a[1] a[2]

Poznámka: reprezentace pole obsahuje ještě počet prvků (členskou proměnnou): **`a.length`**

Pole v jazyku Java, shrnutí

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. 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;
   – nedovolená hodnota indexu způsobí chybu při výpočtu

   java.lang.ArrayIndexOutOfBoundsException: 7

Pole v jazyku Java, shrnutí

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

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

4. Počet prvků pole *p* lze zjistit pomocí zápisu

```
p.length
```

Příklad doporučeného použití:

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

Příklad – "obrat"

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

```
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 obráceně pořadí");
for (int i=pole.length-1; i>=0; i--)
      System.out.println(pole[i]);
  }
}
```

Referenční proměnná pole

- Shrnutí:
  1. pole *n* prvků typu *T* lze v jazyku Java vytvořit pouze dynamicky pomocí operace *new T*[*n*]
  2. Referenci na dynamicky vytvořené 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
  - buď nedefinovanou hodnotu, jde-li o lokální proměnnou
  - nebo speciální hodnotu *null*, která nereferencuje žádné pole, jde-li o statickou proměnnou třídy

Referenční proměnné pole

- Před prvním použitím referenční proměnné pole, je třeba přiřadit referenci na vytvořené pole (!), např. příkazem

```
a = new int[10];
```

- všechny hodnoty jsou nastaveny na nulové:
  - byte, short, int ... 0
  - long 0L
  - double, float 0.0
  - boolean ... false
  - char \u0000
  - refrenční proměnné null

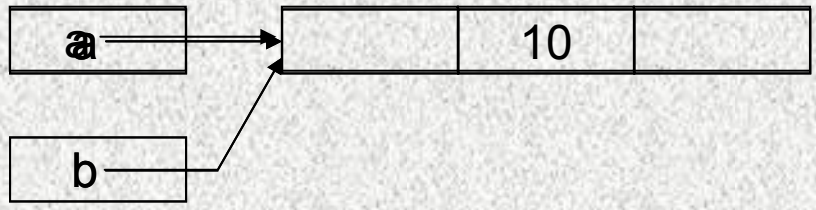
Přiřazení mezi referenčními proměnnými pole

- Příklad:

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

b[1] = 10;

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



- Po přiřazení pak obě proměnné referencují totéž pole!
- 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 jako parametr

```
class Pole {
  public static void main(String[] args) {
       Scanner sc = new Scanner(System.in);
       int n=sc.nextInt(); if (n > 9)n = 9;
       int pole1[] = new int[n];
       for (int i = 0; i < pole1.length; i++)
       pole1[i]=sc.nextInt();
       int pole2[] = {9, 8, 7, 6, 5, 4, 3, 2, 1};
       System.out.println(soucin(pole1, pole2));
 }
 static int soucin(int[] p1, int[] p2){
      int souc=0;
      for (int i = 0; i < p1.length; i++)
          souc=souc + p1[i]*p2[i];
      return souc;
  }
}
```

Pole jako parametr a výsledek funkce

- Reference pole může být parametrem funkce i jejím výsledkem!

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

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

Pole jako parametr a výsledek funkce

```
static int[] ctiPole() {
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();
   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++)
    System.out.println(pole[i]);
 }
```

Způsob předávání parametrů

- Pro předávání parametrů (skutečné → formální) dvě metody:
  - "Volání hodnotou" ("Call by Value")
  - "Volání odkazem" ("Call by Reference")
- Volání hodnotou:
  - V místě předání skutečných parametrů do formálních se předá kopie hodnoty skutečného parametru do parametru formálního.
  - Důsledek – změnou formálního parametru v metodě nelze změnit hodnotu původního skutečného parametru (metoda zná jen kopii hodnoty nikoliv adresu skutečného parametru). Je to bezpečnostní prvek (metoda nemůže ovlivnit své okolí) a zároveň omezení (formální parametr volaný hodnotou nelze použít jako výstupní bod z metody).
- Volání odkazem
  - V místě předání skutečných parametrů do formálních se předá reference („adresa“) na skutečný parametr (reference na pole!!!)

- **Java zná jen volání hodnotou!**
  - **volání hodnotou pro referenční proměnné lze využít pro volání odkazem - týká se pouze polí a objektů!**

Změna pole daného parametrem

- Ve funkci *obratPole* nevytvoříme nové pole, ale obrátíme pole referencované 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[] vstupni_vystupniPole = ctiPole();
      obratPole(vstupni_vystupniPole);
      vypisPole(vstupni_vystupniPole);
   }
```

Proč to funguje?

Protože funkce **obratPole** dostane referenci na pole

Pole jako tabulka

- 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];
```

Příklad – tabulka četnosti čísel

```
static int[] tabulka() {
int[] tab = new int[MAX-MIN+1];
System.out.println("zadejte celých čísel zakončenou nulou");
int cislo = sc.nextInt();
while (cislo!=0) {
   if (cislo>=MIN && cislo<=MAX)
        tab[cislo-MIN]++;
    }
    cislo = sc.nextInt();
}
return tab;
}

static void vypis(int[] tab) { // vypíše tabulku četnosti
for (int i=0; i<tab.length; i++)
  if (tab[i]!=0)
    System.out.println("četnost č. "+(i+MIN)+" je "+tab[i]);
}
```

Příklad – 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) {
      ...
    }
}
```

Příklad – tabulka četnosti písmen

```
static int[] tabulka() {
Scanner sc = new Scanner(System.in);
int[] tab = new int[POCET_PISMEN];
System.out.print("\nzadejtext zakončený:.\n");
String sss = sc.nextLine();
int i=0;
char z = sss.charAt(i++);
while (z!='.') {
 if (jeMalePismeno(z))
    tab[z-'a']++;
    z = sss.charAt(i++);
    }
return tab;
  }
```

Příklad – tabulka četnosti písmen

```
static void vypis(int[] tab) {
for (int i=0; i<POCET_PISMEN; i++)
 if (tab[i]!=0) {
 char z = (char)('a'+i);
 System.out.println("četnost písmena " + z +" je  "+tab[i]);
      }
  }

  static boolean jeMalePismeno(char c) {
    return c>='a' && c<='z';
  }
```

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`**)

Příklad - Eratosthenovo síto

- 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;
  do {                // hledání nejbližšího prvočísla k p
   p++;
     } while (!mnozina[p]);
   } while (p<=pmax);
   return mnozina;
 }
```

Příklad - Eratosthenovo síto

- Funkce pro výpis množiny

```
static void vypis(boolean[] mnozina) {
    for (int i=2; i<mnozina.length; i++)
      if (mnozina[i]) System.out.print(i + " ");
  }
```

- Hlavní funkce

```
public static void main(String[]
    System.out.println("zadejte max");
    int max = sc.nextInt();
    boolean[] mnozina = sito(max);
    System.out.println("Prvočísla od 2 do " + max);
    vypis(mnozina);
  }
```

```
zadejte max
17
prvočísla od 2 do 17
2 3 5 7 11 13 17
```

Vícerozměrné pole

- Vícerozměrným polem se obecně rozumí takové pole, k jehož prvkům se přistupuje pomocí více než jednoho indexu

- 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}};
```

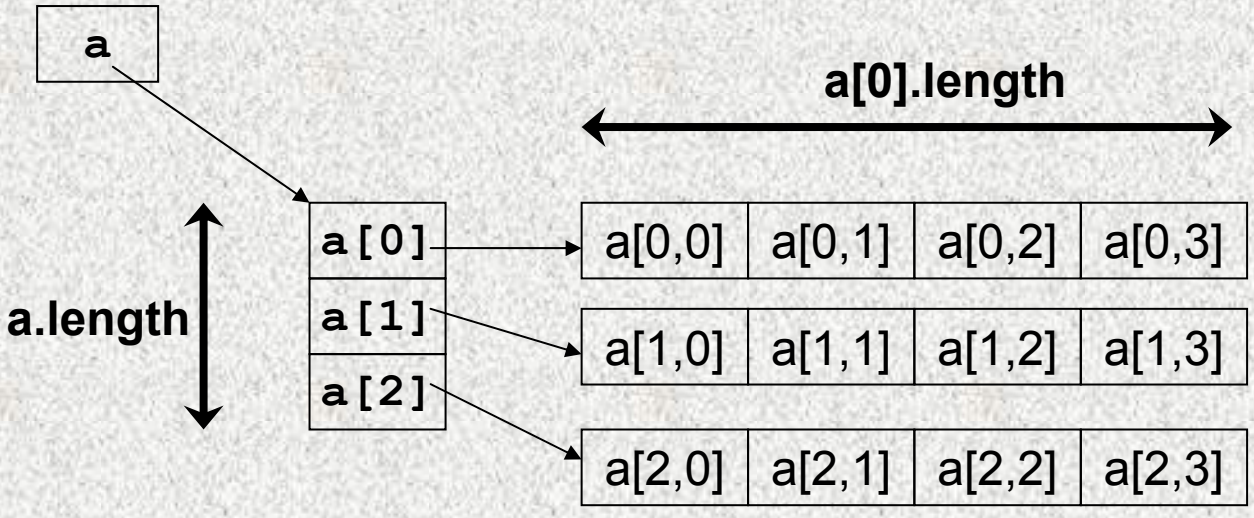
– součet všech prvků pole *mat*

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



Příklad – 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

```
public class Matice {
public static void main(String[] args) {
System.out.println("p. řádků a p.sloupců matice");
int r = sc.nextInt();
int s = sc.nextInt();
int[][] m1 = ctiMatici(r, s);
int[][] m2 = ctiMatici(r, s);
int[][] m3 = soucetMatic(m1, m2);
System.out.println("součet matic");
vypisMatice(m3);
  }
```

Příklad – součet matic

```
static int[][] ctiMatici(int r, int s) {
int[][] m = new int[r][s];
System.out.println("zadejte matici "+r+"x"+s);
for (int i=0; i<r; i++)
   for (int j=0; j<s; j++)
     m[i][j] = sc.nextInt();
    return m;
  }
static void vypisMatice(int[][] m) {
for (int i=0; i<m.length; i++) {
   for (int j=0; j<m[i].length; j++)
     System.out.print(m[i][j] + " ");
     System.out.println();
        }
   }
```

Příklad – 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;
  }
```

Vícerozměrné pole – pokračování

Pole nemusí být nutně pravoúhlá, jsou to pole polí!

```
public class PoleTroj {
  public static void main(String[] args) {
    int[][] a = new int[4][];
    for (int i = 0;  i < a.length;  i++) {
        a[i] = new int[i + 1];
        for (int j = 0;  j < a[i].length;  j++) {
              a[i][j] = i * 10 + j;
              System.out.print(a[i][j] + "  ");
          }
        System.out.println();
    }
  }
}
```

```
0
10  11
20  21  22
30  31  32  33
```

Kolekce, kontejnery - Java

- Třídy a rozhraní slouží k uchování většího, předem neznámého počtu objektů a k přístupu nim, potomci či implemetátoři rozhraní **`Collection`**
- Nejznámější:
  - **`ArrayList`** - „ekvivalent pole“ index>>prvek
    - **`add(), set(), get()`**
  - **`HashSet`** – „ekvivalent množiny“
    - **`add(), remove(), contains()`**
- výhody oproti poli:
    - menší rozsah kódu (algoritmy připravené)
    - optimalizovaný cílový kód
    - zvýšení čitelnosti programu
- nevýhoda oproti poli:
    - nelze vkládat přímo primitivní typy (od Javy 1.5. „lze“, díky autoboxingu)
    - pomalejší přístup k objektům

Pro zájemce, podrobněji v PR2

Příklad – Tabulka, kontejner ArrayList I

```
static ArrayList inicializaceTabulky() {
      ArrayList tab = new ArrayList();
      for (int i = MIN; i <= MAX; i++) {
            tab.add(0);
            //~ tab[MIN]=0;… tab[MAX]=0;
      }
      return tab;
 }
```

Pro zájemce, podrobněji v PR2

Příklad – Tabulka, kontejner ArrayList II

```
static ArrayList tabulka(ArrayList tab) {
System.out.println("zadejte řadu celých zakončenou
 nulou");
int cislo = sc.nextInt();
while (cislo != 0) {
if (cislo >= MIN && cislo <= MAX){
Integer k = (Integer) tab.get(cislo); // k=tab[cislo];
int j = k.intValue() + 1;
tab.set(cislo, j); // tab[cislo]=j;
  }
  cislo = sc.nextInt();
    }
return tab;
  }
```


Pro zájemce, podrobněji v PR2

Příklad – Tabulka, kontejner ArrayList III

```
public static void main(String[] args) {
    ArrayList tab = inicializaceTabulky();
    System.out.println(" " + tab);
    tabulka(tab);
    System.out.println(" " + tab);
  }
```

Výsledek:

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

zadejte řadu celých čísel zakončenou nulou

2 2 2 9 9 9 9 0

[0, 0, 3, 0, 0, 0, 0, 0, 0, 4, 0]

Pro zájemce, podrobněji v PR2

Eratosthenovo síto - HashSet I

- Funkce pro vytvoření množiny prvočísel do *max*

```
static HashSet sito(int max) {
   HashSet mnozina = new HashSet();
   for (int i=2; i<=max; i++) mnozina.add(i);
   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.remove(i);
        do {    //  hledání nejbližšího prvočísla k p
                         p++;
        } while (!mnozina.contains(p));
   } while (p<=pmax);
   return mnozina;
 }
```


Pro zájemce, podrobněji v PR2

Eratosthenovo síto - HashSet II

- Funkce pro výpis množiny

```
static void vypis(HashSet mnozina, int max) {
for (int i=2; i<max; i++)
  if (mnozina.contains(i))
                    System.out.print(i + ", ");
}

public static void main(String[] args) {
System.out.println ("zadejte max");
int max = sc.nextInt();
HashSet mnozina = sito(max);
System.out.println("Množinou,prvočísla od 2 do "+ max);
vypis(mnozina, max);
 }
```


Pro zájemce, podrobněji v PR2