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=Sys.readInt();
...
t7=Sys.readInt();
prumer = (t1+t2+t3+t4+t5+t6+t7)/7;
Sys.pln(t1-prumer);
...
Sys.pln(t7-prumer);
```


- Řešení je těžkopádné a bylo by ještě horší, kdyby vstupními daty byly teploty za měsíc nebo za celý rok
- 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 celými čísly 0, 1, ... počet prvků – 1

  kde počet prvků je dán při vytvoření pole

Pole

- Řešení pomocí pole
  - 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.
  - vstupní data přečteme a do prvků pole uložíme cyklem

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


  - průměrnou teplotu vypočteme jako 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++)
  Sys.pln(teploty[i]-prumer);
```

Pole v jazyku Java

- Pole *p* obsahující *n* prvků typu *T* vytvoříme deklarací
  `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
- Lze zavést pole tvořené prvky s danými hodnotami
  `int p[] = {1,2,3,4,5,6};`
- Zápis
  `p[i]`
  kde *i* je celočíselný výraz, jehož hodnota je nezáporná a menší než počet prvků, označuje prvek pole *p* s indexem *i* a má vlastnosti proměnné typu *T*; nedovolená hodnota indexu způsobí chybu při výpočtu
- Počet prvků pole *p* lze zjistit pomocí zápisu
  `p.length`
  Příklad použití:
  `for (int i=0; i<p.length; i++) Sys.pln(p[i]);`

Příklad

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

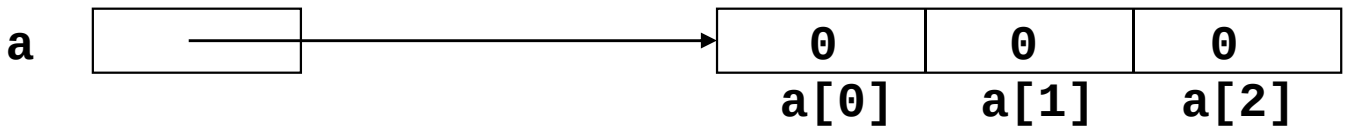
```
package alg6;
import sugar.Sys;

public class ObratPole1 {
  public static void main(String[] args) {
    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();
    Sys.pln("výpis čísel v obráceném pořadí");
    for (int i=pole.length-1; i>=0; i--)
      Sys.pln(pole[i]);
  }
}
```

Přidělení paměti poli

- 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



Poznámka: reprezentace pole je zjednodušená, obsahuje ještě počet prvků

Referenční proměnné pole

- 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;
  ```

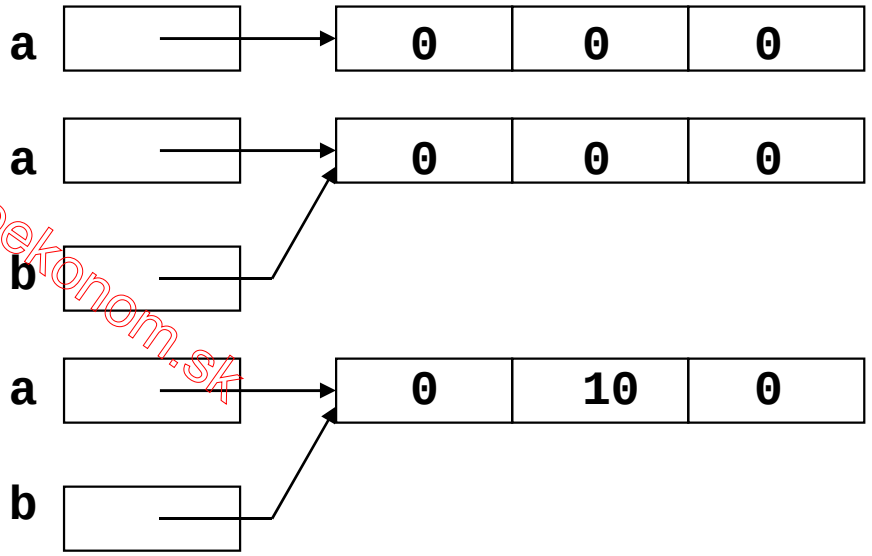
  se zavede referenční proměnná, která má nedefinovanou hodnotu, jde-li o lokální proměnnou, nebo speciální hodnotu *null*, která nereferencuje žádné pole, jde-li o statickou proměnnou třídy
- V obou předchozích případech je třeba, před dalším použitím referenční proměnné pole, jí přiřadit referenci na vytvořené pole, např. příkazem

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

Přiřazení mezi referenčními proměnnými pole

- V jazyku Java je dovoleno přiřazení mezi dvěma referenčními proměnnými polí stejných typů
- Po přiřazení pak obě proměnné referencují totéž pole
- Příklad:

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

int[] b = a;



b[1] = 10;



Sys.pln(a[1])  // vypíše se 10
```



- Přiřazení mezi dvěma poli není v jazyku Java definováno

Pole jako parametr a výsledek funkce

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

```
package alg6;
import sugar.Sys;

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 jako parametr a výsledek 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

- Ve funkci *obratPole* 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);
 }
```

- Proč to funguje? Protože funkce *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 $1 \le x \le 100$, 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

- 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]);
}
```

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) {
    ...
  }
}
```

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;
     // hledání nejbližšího čísla k p
     do {
       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]) 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];
    ```

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

- 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);
  }
```

Příklad – 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();
   }
 }
```

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;
 }
```