Pole

- Referenční datový typ
- Pole – deklarace, vytvoření, použití
- Délka pole
- Statický inicializátor
- Iterátor
- Vícerozměrná pole
- Uložení pole v paměti

Referenční datový typ

- Pole a třídy v jazyce Java používají referenční model
- Doposud probrané datové typy – primitivní datové typy – nepoužívají referenční model tj.:
  - Deklarace proměnné primitivního datového typu – přidělení paměti potřebné pro uchování příslušné hodnoty – paměťové místo obsahuje konkrétní hodnotu
- Referenční model
  - Deklarace proměnné – přidělení paměti pro uchování reference (odkazu, ukazatele), deklarací proměnné ještě není alokována paměť pro hodnotu příslušného typu, rozumnou číselnou hodnotu reference neumíme získat

```
Scanner sc;
```

  - Přidělení paměti v jazyce Java zajistíme operátorem `new` – použití závisí na konkrétním referenčním typu – zda se jedná o pole nebo třídu

```
sc = new Scanner(System.in);
```

  - Hodnota neplatné (neexistující) reference (odkazu) je hodnota konstanty `null`

Pole – deklarace, vytvoření

- Typ pole je obdobou pole v matematice
- Proměnná typu pole obsahuje hodnoty (položky) stejného typu
  - Jednotlivé položky pole jsou číslovány
  - Položky jsou indexovány od 0
  - K položkám pole přistupujeme pomocí indexů

- Deklarace proměnné typu pole

```
<typ polozky>[] <identifikator promenne>;
byte[] poleBytu;
float[] pole;
```

- Přidělení paměti před prvním použitím – pomocí operátoru new

```
<identif. promenne> = new <typ polozky>[<delka pole>];
poleBytu = new byte[100];
```

- Spojení deklarace a vytvoření pole

```
int[] a = new int[20];
```

- Nelze pracovat s polem, které nemá přidělenou paměť
- Alokace paměti operátorem **new** – přidělená pamět je inicializována/vynulována

Pole – délka, přístup k položkám

- Prvky (položky) pole jsou číslovány od 0
- Každá proměnná typu pole má k dispozici „členskou proměnnou“ `length` a některé metody

```
int[] a = new int[5];
```

- Proměnná a má indexy od `0` do `a.length-1`
- K položkám pole (jednotlivým hodnotám v poli) přistupujeme přes indexy, index uvádíme v hranatých závorkách za jménem proměnné typu pole

```
for (int i = 0; i < a.length; i++){
      a[i] = sc.nextInt();
}
```

- Jedná se o statická pole –
  - Délku aktuálně používaného pole nelze dynamicky měnit,
  - Pokud do původní proměnné přiřadíme nově vytvořené pole (třeba delší) ztratíme přístup k původnímu poli,
  - Pokud potřebujeme zvětšit délku pole – možná realokace – použijeme další pomocnou proměnnou typu pole, vytvoříme nové pole, do něj zkopírujeme hodnoty z původního pole, původní proměnné přiřadíme referenci na nové pole

Načtení hodnot, součet

```
int[] a;
int sum;
int n = sc.nextInt();

a = new int[n];
for (int i = 0; i < a.length; i++){
       a[i] = sc.nextInt();
}

sum = 0;
for (int i = 0; i < a.length; i++){
       sum = sum + a[i];
}
System.out.println(sum);
```

Načtení hodnot, součet

```
final int KAPACITA = 100;
int[] a;
int sum, x;
int n;
n = 0;
a = new int[KAPACITA];

while ((n < a.length) && ((x = sc.nextInt()) > 0)) {
  a[n] = x;
  n++;
  if (n == a.length) {
    System.out.println("Vycerpana kapacita pameti");
  }
}

sum = 0;
for (int i = 0; i < n; i++) {
  sum = sum + a[i];
}
System.out.println(sum);
```

Inicializované pole

- Pole můžeme alternativně vytvořit pomocí statického inicializátoru

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

- V tomto případě se nejedná o pole konstant – jednotlivé hodnoty v inicializovaném poli můžeme měnit

```
byte[] poctyCifer = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
long cislo = sc.nextLong();
byte cifra;
while (cislo > 0) {
      cifra = (byte)(cislo % 10);
      poctyCifer[cifra]++;
      cislo = cislo / 10;
}
```

Iterace přes všechny prvky pole

- Modifikace příkazu cyklu `for`, varianta `for-each`

```
for (<typ_položky> <identifikátor> : <existující_pole>) {
       //
}
```

- Příklad použití

```
byte[] poctyCifer = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
long cislo = sc.nextLong();
byte cifra;
while (cislo > 0) {
  cifra = (byte)(cislo % 10);
  poctyCifer[cifra]++;
  cislo = cislo / 10;
}
// využití cyklu for jako iterátoru
for (byte c : poctyCifer) System.out.print(c + "  ");
// varianta téhož
for (int i = 0; i < poctyCifer.length; i++)
   System.out.print(poctyCifer[i] + "  ");
```

Dvourozměrné pole

- Deklarace a použití dvourozměrného pole

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

int[][] b = new int[10][10];

for (int i = 0; i < a.length; i++)
  for (int j = 0; j < a[i].length; j++)
    a[i][j] = sc.nextInt();
```

- Proměnná a je pole tří hodnot – každá z těchto hodnot je referencí na pole délky 4 hodnot typu `int`, obdobně proměnná b
- Jedná se o pole hodnot typu pole – jednotlivé vnořené pole nemusí mít stejnou délku
- Použití – reprezentace matic (viz matematika), reprezentace aktuálního stavu deskových her, uchování dvourozměrného rastrového obrazu

- Obdobně pracujeme s vícerozměrnými poli

Další aspekty práce s poli

- Uložení pole v paměti
- Vícerozměrná pole a jejich uložení v paměti

- Vícerozměrné pole s nestejnou délkou jednotlivých vektorů
  - Alokace pole
  - Alokace jednotlivých vektorů

- Přiřazení mezi proměnnými typu pole

- Vzájemné porovnání dvou polí
  - Aplikace relačních operátorů na proměnné typu pole
  - Jak porovnat obsah dvou polí

Pole a metody

- Pole jako parametr metod
  - Parametry metod jsou předávané hodnotou
  - V případě pole je „předána hodnota reference“
  - Mechanismus předávání – význam
  - V parametru typu pole můžeme vracet i výsledek činnosti metody – pole musí být vytvořené před voláním metody
- Pole může být i návratovým typem metody
- Metoda může vracet pouze jedinou hodnotu, v případě více výstupních hodnot je nutné tyto sdružit do jediné struktury (objektového typu, typu pole – dle možností a potřeb)

- Metody s libovolným počtem parametrů

```
double soucet(double... a) {
  double vysledek = 0;
  for (double x: a) {
    vysledek += x;
  }
  return vysledek;
}

...

System.out.println(soucet(1,2,3,4,5));
double[] pole = {1,2,3,4,5};
System.out.println(soucet(pole));
```

- Význam formálního parametru
- Skutečné parametry
  - Výčet hodnot příslušného typu
  - Pole hodnot příslušného typu