Algoritmizace a programování

Typ pole

Referenční datové typy

Typ pole

Deklarace typu pole a přidělení paměti

Přístup k položkám pole

Délka pole

Verze pro akademický rok 2012/2013

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`**

Typ pole

- 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 a vytvoření pole

- 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];
```

- Přidělení paměti často zároveň s deklarací

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

- Nelze pracovat s polem, které nemá přidělenou paměť
- Alokace paměti operátorem **new** – obsah přidělené paměti je inicializován

Délka pole, přístup k položkám pole

- 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 uvedeme v hranatých závorkách za jménem proměnné typu pole

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

- Délku aktuálně používaného pole nelze dynamicky měnit, novou alokací paměti ztratíme přístup k původnímu poli

```
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);
```

```
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;
}
```

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