ALG1 – přehled základních algoritmů a primitiv – Java

Náhodná čísla

1. Pomocí **Math.random()** – standardně od [0.0, 1.0)

```
public int getRandomNumber(int min, int max) {
    return (int) ((Math.random() * (max - min)) + min);
}
```

2. Pomocí objektu **Random** – standardně od [0.0, 1.0)

```
private final static Random random = new Random();
public int getRandomNumber (int min, int max) {
    return random.nextInt(max - min) + min;
}
```

Blok try-catch-finally

- větev **finally** se provede vždy (nezáleží, že předtím vracím z metody hodnotu)

```
static int testTryCatchFinally() {
     try {
         return 1;
     } catch (Exception ex) {
         return -1;
     } finally {
         System.out.println("Vetev finally"); // provede se vždy
     }
}
```

Sekvenční načítání dat

1. mám zadaný **počet**, následně načítám čísla

```
int pocetPrvku = sc.nextInt(); // předem zadaný počet prvků
int cislo, suma = 0;
System.out.println("Zadej posloupnost:");
for (int i = 0; i < pocetPrvku; i++) {
    cislo = sc.nextInt(); // načítám jednotlivé členy posloupnosti
    suma += cislo;
}
```

2. zadávání je ukončeno vhodným **terminátorem** (typicky 0 nebo záporné číslo)

```
int cislo, suma = 0;
System.out.println("Zadej posloupnost, ukonči nulou:");
while ((cislo = sc.nextInt()) != 0) {
    suma += cislo;
}
```

3. zjištění **minima** a **maxima**

```
int cislo = 0;
int min = Integer.MAX_VALUE;
int max = Integer.MIN_VALUE;
System.out.println("Zadej posloupnost, ukonci nulou:");
while ((cislo = sc.nextInt()) != 0) {
    if (cislo < min) min = cislo;
    if (cislo > max) max = cislo;
}
```

4. zjištění, zda je posloupnost **vzestupně/sestupně seřazená**

```
int pocetPrvku = sc.nextInt();
int cislo;
System.out.println("Zadejte posloupnost:");
int cisloPred = sc.nextInt();
boolean jeVzestup = true;
boolean jeSestup = true;

for (int i = 1; i < pocetPrvku && (jeVzestup || jeSestup); i++) {
    cislo = sc.nextInt();
    if (cisloPred > cislo) jeVzestup = false;
    if (cisloPred <= cislo) jeSestup = false;
    cisloPred = cislo;
}
```

Prvočísla

```
int cislo = sc.nextInt();
boolean jePrvocislo = true;
if (cislo < 2) jePrvocislo = false;
// v cyklu staci testovat od i do Math.sqrt(cislo) - matematika
for (int i = 2; i * i <= cislo && jePrvocislo; i++) {
    if (cislo % i == 0) {
        jePrvocislo = false;
    }
}
```

Knihovní třída

- má modifikátor **final** (je zapečetěná – nelze od ní dědit), má privátní konstruktor (zamezení vytvoření instance) a všechny metody uvnitř jsou statické – pak se volá *trida.metoda()*

```
public final class MathTools {
    private MathTools() { }
    public static int addNumbers(int a, int b) {
        return a + b;
    }
}
```

Metody pro práci s polem

- při načítání dat do pole je nejlepší cestou nejdříve zadat **počet**, a pak alokovat pole s touto velikostí, jinak je potřeba použít metodu na (neefektivní) **realokaci pole**

```
public static int[] resizeArray(int[] arr, int newSize) {
    int[] newArr = new int[newSize];
    for (int i = 0; i < Math.min(arr.length, newSize); i++) {
        newArr[i] = arr[i];
    }
    return newArr;
}
```

- hezký **výpis**

```
public static void printNiceArray(int[] arr) {
    System.out.print("[");
    for (int i = 0; i < arr.length - 1; i++) {
        System.out.print(arr[i] + ", ");
    }
    System.out.print(arr[arr.length - 1]);
    System.out.println("]");
}
```

- **maximální hodnota**

```
public static int findMaxValue(int[] arr) {
    int max = arr[0]; // prip. Integer.MIN_VALUE, pak prochazet od 0
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] > max) {
            max = arr[i];
        }
    }
    return max;
}
```

- **obrácení** hodnot
  1. vracím **nové pole**

```
public static int[] reverseValues(int[] arr) {
    int[] revArr = new int[arr.length];
    for (int i = 0; i < arr.length; i++) {
        revArr[i] = arr[arr.length - i - 1];
    }
    return revArr;
}
```

  2. upravuji **staré pole**

```
public static void reverseValues(int[] arr) {
    int temp;
    for (int i = 0; i < arr.length/2; i++) {
        temp = arr[i];
        arr[i] = arr[arr.length-1-i];
        arr[arr.length-1-i] = temp;
    }
}
```

- **binární vyhledávání** – vyhledávání **půlením intervalu** – pole musí být **SEŘAZENÉ**!

```
public static int binarySearch(int[] array, int target) {
    int position = -1;
    int start = 0;
    int end = array.length - 1;

    do {
        int middle = (start + end) / 2; // puleni intervalu
        if (array[middle] == target) position = middle;
        else if (array[middle] > target) end = middle - 1;
        else start = middle + 1;
    } while ((position == -1) && (start <= end));

    return position;
}
```

Řadicí algoritmy

- **Selection Sort** – řazení výběrem

```
int minIndex, temp;
for (int i = 0; i < pole.length - 1; i++)
{
    minIndex = i;
    for (int j = i + 1; j < pole.length; j++)
    {
        if (pole[j] < pole[minIndex]) minIndex = j;
    }
    if (i != minIndex) {
        temp = pole[minIndex];
        pole[minIndex] = pole[i];
        pole[i] = temp;
    }
}
```

- **Insertion Sort** – řazení vkládáním

```
int temp, j;
for (int i = 1; i < pole.length; i++)
{
    temp = pole[i];
    j = i - 1;
    while (j >= 0 && pole[j] > temp) {
        pole[j+1] = pole[j];
        j--;
    }
    pole[j+1] = temp;
}
```

- **Bubble Sort** – řazení záměnou

```
int i, j, temp;
boolean swapped = true; // abych zbytecne neprochazel pole, pokud jiz je serazeno
for (i = 0; i < pole.length - 1 && swapped; i++) {
    swapped = false;
    for (j = 0; j < pole.length - i - 1; j++) {
        if (pole[j] > pole[j + 1]) {
            temp = pole[j];
            pole[j] = pole[j + 1];
            pole[j + 1] = temp;
            swapped = true;
        }
    }
}
```

Proměnný počet parametrů

- v metodě s tím pak pracuji jako s **polem**

```
public static void printNumbers(int... numbers) {
    for (int num : numbers) {
        System.out.print(num + " ");
    }
    System.out.println();
}
```

Dvourozměrné pole (matice)

```
int[][] pole = {{1,2,3,4}, {5,8,9,10}, {10,11,12,13}, {100,100,100,100}};
// vyska (pocet radku)
int height = pole.length;
// sirka (pocet sloupcu)
int width = pole[0].length; // nemusi byt jen nulty index (radek), pole nemusi mit
stejny pocet sloupcu v kazdem radku
// posledni prvek
int last = pole[height-1][width-1];
```

- **násobení matic**

```
public static int[][] matrixProduct(int[][] mat1, int[][] mat2) {
    int rows1 = mat1.length;
    int cols1 = mat1[0].length;
    int rows2 = mat2.length;
    int cols2 = mat2[0].length;

    if (cols1 != rows2) {
        throw new IllegalArgumentException("Neni splnena podminka pro nasobeni“);
    }

    int[][] result = new int[rows1][cols2];

    for (int i = 0; i < rows1; i++) {
        for (int j = 0; j < cols2; j++) {
            for (int k = 0; k < cols1; k++) {
                result[i][j] += mat1[i][k] * mat2[k][j];
            }
        }
    }

    return result;
}
```