Algoritmizace a programování 1

- Struktura programu
- Příkazy jazyka Java
- Logický příkaz a logické výrazy
- Příkazy cyklu
- Příkaz `switch`
- Ostatní příkazy jazyka

SHRNUTÍ PŘEDCHOZÍCH TÉMAT

Program v jazyce Java – opakování

- Program v jazyce Java – třídy
- **Třídy** umístěné v **balících** – příslušnost do balíku
- Třída má **jméno** a **tělo**
  - Jméno – identifikátor – začínat velkým písmenem, další slova začínat velkým písmenem
  - Tělo obsahuje deklarace atributů a metod
- **Atributy** – členské proměnné – uchování dat
- **Metoda** obsahuje programový kód (výkonný kód, příkazy), příkazy se postupně provedou po zavolání metody
- **Program** – hlavní třída – metoda `main`
- Po spuštění programu – je spuštěn kód metody `main` hlavní třídy programu
- Každá deklarace atributu (proměnné) musí být ukončená středníkem
- Každý příkaz musí být ukončený středníkem

```
public class <jméno třídy> {

  // schéma deklarace atributu/proměnné
  <typ> <jméno>;

  // schéma metody
  <typ> <jméno>
   (<deklarace parametrů>) {
      <příkazy>
  }

  public static void main(String[] args) {
    // kód
    // příkazy
  }

}
```

Co už umíme

- Vytvořit nový konzolový aplikaci v jazyce Java – projekt v NetBeans
- Základní strukturu třídy
- Co je třeba pro to, aby třída mohla být hlavní třídou programu
- Co je v kódu metody
- Jak deklarovat proměnné
- Některé základní typy
- Aritmetické operace
- Jak vypsat zprávy a výsledky na konzoli (standardní výstup)
- Jak načíst celočíselné, reálné hodnoty, slova, textové zprávy a znaky ze standardního vstupu (z konzole)

- Vše zatím pouze v základním přehledu bez podrobností

```
public class VypocetObdelnika {

 /**
  * @param args
  */
 public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    float a, b;
    float obvod;
    float obsah;
    System.out.println(„Zadej delku stran obdelnika ");
    a = sc.nextFloat();
    b = sc.nextFloat();
    obvod = 2 * (a + b);
    obsah = a * b;

    System.out.println("Obdelnik o stranach " +
       a + " a " + b + " ma ");
    System.out.println("obvod " + obvod);
    System.out.println("plochu " + obsah);
    System.out.println("Konec programu.");
  }
}
```

Doplnění předchozího tématu

- Aritmetické operace – konstrukce aritmetických výrazů – výpočet složitějších matematických funkcí – odmocnina, goniometrické funkce, logaritmy, exponenciály
  - Výpočet matematických funkcí pomocí prostředků (metod třídy `Math`)
- Výpis na standardní výstup programu
  - Atribut `out` třídy `System` – `System.out`
  - Standardní výstup implicitně směrován na konzoli
  - Pro výpis na standardní výstup použijeme přímo prostředky proudu, který je v `System.out`
- Čtení ze standardního vstupu
  - Atribut `in` třídy `System` – `System.in`
  - Standardní vstup nasměrován implicitně z konzole
  - Pro načítání ze standardního vstupu lze použít přímo prostředky proudu `System.in`. Tyto prostředky neumožňují přímé načtení hodnoty jednotlivých primitivních typů. Z toho důvodu používáme `Scanner`. Objekt třídy `Scanner` potom filtruje požadovaný vstup a poskytuje řadu prostředků pro přímé načtení hodnot různých typů. Třída `Scanner` je v balíku `java.util`, nutný import.
- Kromě standardního vstupu a výstupu má aplikace i standardní chybový výstup `System.err`
- Pojmenování proměnných, metod, tříd, balíků, konstant

PŘÍKAZY JAZYKA JAVA

Tělo metody

- V těle metod jsou umístěny příkazy
- Každý příkaz musí být ukončený středníkem
- Před příkazem (ne deklarací) může být uvedeno „návěští“ – návěští se předem nedeklarují
  `<návěští> : <příkaz>;`
- Na místě příkazu může být
  - Deklarace lokálních proměnných
  - Prázdný příkaz
  - Výraz použitý jako příkaz
    - Přiřazení, inkrementace, dekrementace
    - Volání metody
    - Vytvoření instance třídy
  - Standardní příkaz jazyka Java, kterými jsou
    - Logický příkaz `if`
    - Příkaz cyklu `while`
    - Příkaz cyklu `do-while`
    - Příkaz cyklu `for` – 2 varianty
    - Příkaz `switch`
    - Příkazy `break`, `continue`
    - Příkaz `return`
    - Příkazy `throw`, `try`, `assert`, `synchronized`
- Sekvenci příkazů lze sdružit do příkazového bloku – { }

Deklarace proměnných a konstant

- Deklarace proměnných v metodě

```
<typ> <jmeno>;
<typ> <jmeno>, <jmeno>, <jmeno>;
<typ> <jmeno> = <výraz>;
```

- Deklarace konstant v metodě

```
final <typ> <jmeno> = <výraz>;
final <typ> <jmeno>;
```

- Deklarace dané proměnné určuje typ a jméno (identifikátor)
- V jedné metodě nelze identifikátor předeklarovat
- Proměnná z hlediska programu představuje paměťové místo do kterého lze zapsat hodnotu (data) nebo hodnotu (data) číst – uchování hodnoty, čtení hodnoty, přiřazení hodnoty
- V jazyce Java nelze používat hodnotu proměnné, pokud hodnota nebyla definována (přiražena) – přiřazení inicializací, operací přiřazení, načtením
- Typ určuje velikost paměti a to, jakým způsobem budou hodnoty ukládány (konvertovány do binárního kódu) – jazyk Java je jazyk se silnou typovou kontrolou – typ tak určuje rozsah zobrazitelných hodnot, množinu povolených operací
- Deklarace může být na libovolném místě kódu, ale vždy před prvním použitím proměnné
- Deklarace stojící na místě příkazu – při provádění tohoto příkazu je pro danou proměnnou přiděleno paměťové místo
- Pozdější použití proměnné v programu – odkazuje na dané paměťové místo

Výraz použitý jako příkaz

- Operace přiřazení
  `<jméno_proměnné> = <výraz>`
- Operátory přiřazení
  - =
  - **+= -= *= /= %=**
  - **…**
- Inkrementace, dekrementace hodnoty proměnné
  - i++
  - ++i
  - i--
  - --i
- Volání metody
- Vytvoření instance třídy – operátor new

```
int a, b;
// operace přiřazení na místě příkazu
a = 7;
a++;
b = Math.pow(a, 3);
b /= 16;

// výměna proměnných
int pom;
pom = a;
a = b;
b = pom;

// volání metody na místě příkazu
System.out.println(a);
System.out.println(b);

// vytvoření instance
new Scanner(System.in);
```

Blok příkazů

- Při zápisu jednotlivých řídících konstrukcí, lze na některých místech kódu zapsat pouze jediný příkaz jazyka Java – například v těle cyklu. Pokud na daném místě chceme provádět více akcí – více příkazů jejichž sekvence se má postupně vykonat – je nutné na daném místě použít **blok příkazů**
- Základní konstrukce bloku příkazů je tvořena složenými závorkami, uvnitř složených závorek může být libovolné množství příkazů, každý příkaz musí být ukončen středníkem

```
{ <prikaz1>; <prikaz2>; … <prikazN>;}
```

- Blok příkazů nesmí být ukončen středníkem
- Tělo metody (používáme zatím pouze metodu `main`) je tvořeno blokem příkazů – viz struktura programu
- Deklarace třídy je tvořena blokem, ve kterém jsou deklarace atributů (členských proměnných) a metod – viz struktura programu

ROZHODOVÁNÍ, VĚTVENÍ

Logický příkaz `if`

- Logické příkazy řídí vykonávání kódu na základě splnění nebo nesplnění zadané podmínky
  - Neúplná podmínka – příkaz `if`
  - Úplná podmínka – příkaz `if-else`
- **Neúplná podmínka** – podmíněné vykonání příkazu nebo sady příkazů

  `if (` `<podmínka>` **`)`** `<část_T>`
- **Úplná podmínka** – větvení kódu do dvou větví v závislosti na hodnotě (splnění/nesplnění) podmínky

  `if (` `<podmínka>` **`)`** `<část_T>`
  `else` `<část_E>`
  - **`if`**, **`else`** – klíčová slova jazyka Java
  - `<podmínka>` – výraz, který po vyhodnocení má hodnotu logického typu (`true`, `false`)
  - `<část_T>`, `<část_F>` – buď příkaz jazyka Java ukončený středníkem, nebo blok příkazů, pokud na místě příkazu stojí blok příkazů, potom za tímto blokem středník být nesmí

- Význam (sémantika) příkazu, způsob vykonávání příkazu
- Úplná podmínka
  - Při provádění příkazu se vyhodnotí nejdříve výraz `<podminka>`. Pokud má výraz po vyhodnocení hodnotu `true` potom se provede příkaz `<prikazT>` za podmínkou v opačném případě je příkaz `if` ukončen a pokračuje se prováděním dalšího příkazu za příkazem `if`.
- Neúplná podmínka
  - Při provádění příkazu se vyhodnotí nejdříve výraz `<podminka>`. Pokud má výraz po vyhodnocení hodnotu `true` potom se provede příkaz za podmínkou v opačném případě se provede příkaz `<prikazF>` za klíčovým slovem `else` . Po provedení příkazu v jedné z větví je příkaz `if` ukončen a pokračuje se prováděním dalšího příkazu za příkazem `if`

Podmínky – konstrukce a vyhodnocení

- Podmínka – logický výraz – výraz, který má po vyhodnocení hodnotu typu `boolean` – pravdivý/nepravdivý `true/false`
- Typ `boolean` – primitivní typ jazyka Java, množina hodnot `true/false`
- Proměnné typu `boolean` – deklarace a využití
- Výraz logického typu
- Porovnávání číselných (znakových, logických) hodnot – relační operátory – výsledkem relační operace je hodnota typu `boolean`

  `==` `!=` `<` `>` `<=` `>=`

- Pro sestavení komplexních logických výrazů z jednodušších používáme logické operátory – některými logickými operátory jsou
  - Operátor pro logický součet – logický OR – binární operátor `||`
  - Operátor pro logický součin – logický AND – binární operátor `&&`
  - Operátor logická negace – logický NOT – unární operátor (stojí před výrazem, který neguje) `!`

Ternární operátor – podmíněné výrazy

- Jazyk Java obsahuje logický ternární operátor – **`? :`**
- Ternární operátor má do jisté míry podobný význam jako úplná podmínka `if-else`
- Ternární operace má tři operandy – vlastní operátor je tvořen dvěma částmi – `? a :`
- Ternární operátor – konstrukce podmíněných výrazů

- **Syntaxe** – způsob zápisu podmíněného výrazu

```
<podmínka> ? <výrazT> : <výrazF>
```

  - `? :` – znaky operátoru
  - `<podmínka>` – výraz, který má po vyhodnocení hodnotu logického typu
  - `<výrazT>, <výrazF>` – výrazy jazyka Java

- **Sémantika** – význam, způsob interpretace
  - Při vyhodnocování operace je nejdříve vyhodnocen výraz `<podmínka>`
  - Pokud `<podmínka>` má hodnotu `true`, vyhodnocuje se dále výraz `<vyrazT>` uvedený za symbolem **`?`** a jeho hodnota určuje výslednou hodnotu celého podmíněného výrazu
  - Pokud `<podmínka>` má po vyhodnocení hodnotu `false`, vyhodnocuje se dále výraz `<výrazF>` uvedený za symbolem **`:`** a jeho hodnota určuje výslednou hodnotu celého podmíněného výrazu

- Ternární operátor – zápis operace – konstrukce složitějších výrazů – po vyhodnocení má konkrétní hodnotu

Ukázka použití logických příkazů, výrazů

- Úloha:
  - Určete větší ze dvou zadaných čísel
  - Zadaná čísla v proměnných a, b – proměnné libovolného číselného typu
  - Další pomocná proměnná max – totožného typu jako proměnné a, b

```
// použití neúplného logického příkazu
max = a;
if (b > a) max = b;

// použití úplného logického příkazu
if (b > a) max = b;
else max = a;

// použití ternárního operátoru
max = (a > b)? a : b;

// využití metody z třídy Math jazyka Java
max = Math.max(a,b);
```

Poznámky k logickým příkazům

Logický příkaz – úplná podmínka
Podmíněný výraz

- **Úplná podmínka**
  - Jedná se o **příkaz** – v jednotlivých větvích jsou příkazy, které mohou být zastoupeny blokem příkazů
- **Podmíněný výraz**
  - Jedná se **výraz** – po vyhodnocení má hodnotu konkrétního typu, tuto hodnotu je možné použít ve složitějším výrazu, přiřadit proměnné, vypsat

Kombinování úplných a neúplných podmínek

```
if (<podmínka1>)
    if (<podmínka2>) <příkaz1>;
else <příkaz2>;
```

- Příkaz `<příkaz2>` se provede v případě, že podmínka `<podmínka1>` má hodnotu `true` a podmínka `<podmínka2>` hodnotu `false`, větev `else` se přiřazuje nejbližšímu `if`.
- Pokud si chceme vynutit jinou interpretaci – provedení příkazu `<příkaz2>` pro `<podmínka1>==false`, potom je nutné použít blok příkazů:

```
if (<podmínka1>){
    if (<podmínka2>) <příkaz1>;
} else <příkaz2>;
```

- Obecné doporučení – používat blok příkazů v obou větvích (pro jednoznačné vymezení obdobně i v dalších příkazech `while` a `for`)

CYKLY, OPAKOVÁNÍ

Příkaz cyklu `while`

- Cyklus – opakované provádění operací (příkazu, sady příkazů)
- Cyklus `while` – cyklus řízený podmínkou – opakování/ukončení je prováděno na základě splnění/nesplnění zadané podmínky

- **Syntaxe** příkazu (způsob zápisu)

```
while (<podminka>) <tělo_cyklu>
```

  - **`while`** – rezervované slovo jazyka Java
  - `<podmínka>` – libovolný výraz jazyka, který má po vyhodnocení hodnotu logického typu
  - `<tělo_cyklu>` – příkaz jazyka Java nebo příkazový blok

- **Sémantika** (význam, způsob vykonání)
  1. Vyhodnotí se výraz `<podmínka>`
  2. Pokud má výraz `<podmínka>` po vyhodnocení hodnotu `true` provede se část `<tělo_cyklu>` a dále se přejde k bodu 1
  3. Pokud má výraz `<podmínka>` po vyhodnocení hodnotu `false`, příkaz cyklu se ukončí

Příkaz cyklu do-while

- Cyklus – opakované provádění operací (příkazu, sady příkazů)
- Cyklus `do-while` – cyklus řízený podmínkou – opakování/ukončení je prováděno na základě splnění/nesplnění zadané podmínky

- **Syntaxe**

```
do {
    <příkaz1>;
    <příkaz2>;
        …
} while (<podmínka>);
```

  - **`do, while`** – rezervovaná slova jazyka Java
  - `<podmínka>` – libovolný výraz jazyka, který má po vyhodnocení hodnotu logického typu
  - `<příkaz1>, <příkaz2>` … – příkaz/příkazy jazyka Java

- **Sémantika**
  1. Postupně se provedou příkazy v těle cyklu `<příkaz1>, <příkaz2>` atd.
  2. Vyhodnotí se výraz `<podmínka>`
  3. Pokud má výraz `<podmínka>` po vyhodnocení hodnotu `true`, přejde se opakovaně k bodu 1
  4. Pokud má výraz `<podmínka>` po vyhodnocení hodnotu `false`, ukončí se vykonávání cyklu

Příkaz cyklu `for`

- Cyklus – opakované provádění operací (příkazu, sady příkazů)
- Cyklus `for` umožňuje kromě podmínky definovat i tzv. „inicializační“ a „iterační“ části

- **Syntaxe** (způsob zápisu)

```
for (<init_část>; <podmínka>; <iter_část>) <tělo_cyklu>
```

  - **`for`** – rezervované slovo jazyka Java
  - `<init_část>` – inicializační část příkazu, výraz nebo sada výrazů oddělených čárkami
  - `<podmínka>` – libovolný výraz jazyka, který má po vyhodnocení hodnotu logického typu
  - `<iter_část>` – iterační část příkazu, výraz nebo sada výrazů oddělených čárkami
  - `<tělo_cyklu>` - příkaz jazyka Java nebo příkazový blok

- **Sémantika** (význam, způsob vykonání)
  1. Vyhodnotí se výraz `< init_část >`.
  2. Vyhodnotí se výraz `<podmínka>`.
  3. Pokud má výraz `<podmínka>` po vyhodnocení hodnotu `false`, příkaz cyklu se ukončí.
  4. Pokud má výraz `<podmínka>` po vyhodnocení hodnotu `true` :
     a) Provede se `<tělo_cyklu>` – příkaz respektive sada příkazů příkazového bloku
     b) Vyhodnotí se výraz `<iter_část>`
     c) Přejde se k bodu 2

- V inicializační a iterační části může být více výrazů navzájem oddělených čárkami
- Libovolná z částí `<init_část>`, `<podmínka>`, `<iter_část>` může být prázdná

Varianta cyklu `for` – cyklus `for-each`

- Opakované provádění sady operací pro každou z hodnot pole/kontejneru
- Pole a manipulace s poli – bude probíráno později

- **Syntaxe** (způsob zápisu)

```
for (<typ> <proměnná> : <pole>) <tělo_cyklu>
```

  - **`for`** – rezervované slovo jazyka Java
  - `<typ>` – typ jazyka Java kompatibilní s položkami `<pole>`
  - `<proměnná>` – libovolný výraz jazyka, který má po vyhodnocení hodnotu logického typu
  - `<pole>` – pole nebo proměnná některého z kontejnerových typů, která obsahuje položky typu `<typ>`
  - `<tělo_cyklu>` - příkaz jazyka Java nebo příkazový blok

- **Sémantika** (význam, způsob vykonání)
  1. Alokuje se paměť pro proměnnou `<proměnná>` typu `<typ>`
  2. Pokud `<pole>` neobsahuje další položky je cyklus ukončen v opačném případě se provede:
     a) Do proměnné `<proměnná>` je přiřazena další hodnota z `<pole>`
     b) Provede se `<tělo_cyklu>` – příkaz respektive sada příkazů příkazového bloku
     c) Přejde se k bodu 2

Použití příkazů cyklu – úloha

- Úloha:
  - Výpočet a^n – tj. celočíselné kladné mocniny čísla (celého nebo reálného)
  - Vypočet provedeme dle: `a^n = 1 * a * a * a * a * a ...`
  - Při výpočtu provádíme opakované násobení – `n-krát` násobíme hodnotou `a`

  - Vstupní hodnoty algoritmu v proměnných
    - a – základ mocniny, proměnná libovolného číselného typu
    - n – exponent (mocnitel) proměnná celočíselného typu
  - Další proměnné
    - `i` – počítadlo provedeného počtu opakovaní, proměnná celočíselného typu
    - v – průběžný mezivýsledek a^i, po ukončení konečný výsledek a^n, počítaná hodnota proměnná číselného typu kompatibilního s typem proměnné a

Použití příkazů cyklu – řešení úlohy

```
// použití while
i = 1;
v = 1;
while (i <= n) {
  v = v * a;
  i++;
}

// použití for
v = 1;
for (int i = 1; i <= n; i++) {
  v = v * a;
}
```

```
// použití do-while
v = 1;
i = 1;
do {
  v = v * a;
  i++;
} while (i <= n);

// ještě jednou do-while
v = 1;
i = 1;
if (n > 0) {
  do {
    v = v * a;
    i++;
  } while (i <= n);
}
```

Použití příkazů cyklu – opakované řešení úlohy

- Základní úloha
  - Výpočet obdélníka
  - Určení většího ze dvou zadaných čísel
  - Výpočet nezáporné celočíselné mocniny
- Zápis kódu, který bude úlohu řešit opakovaně

- Řešení základní úlohy

```
načti data
vypočti základní úlohu
vypiš výsledky
```

- Opakované řešení úlohy

```
opakuj_dle_potřeby {
    načti data
    vypočti základní úlohu
    vypiš výsledky
}
```

Použití příkazů cyklu – opakované řešení úlohy II

```
do {
  System.out.println("Zadej strany obdelnika ");
  a = sc.nextFloat();
  if (a > 0) {
    b = sc.nextFloat();
    obvod = 2 * (a + b);
    obsah = a * b;
    System.out.println("obvod " + obvod);
    System.out.println("plochu " + obsah);
    System.out.println();
  }
} while (a > 0);
```

```
System.out.println("Zadej strany obdelnika ");
while ((a = sc.nextFloat()) > 0) {
  b = sc.nextFloat();
  obvod = 2 * (a + b);
  obsah = a * b;
  System.out.println("obvod " + obvod);
  System.out.println("plochu " + obsah);
  System.out.println();
  System.out.println("Zadej strany obdelnika ");
}
```

- Základní úloha – výpočet obdélníka

```
Scanner sc = new Scanner(System.in);
float a, b;
float obvod;
float obsah;
```

```
System.out.println("Zadej strany obdelnika ");
for (a = sc.nextFloat(); a > 0;
     a = sc.nextFloat()) {

  b = sc.nextFloat();
  obvod = 2 * (a + b);
  obsah = a * b;
  System.out.println("obvod " + obvod);
  System.out.println("plochu " + obsah);
  System.out.println();
  System.out.println("Zadej strany obdelnika ");
}
```

Problematika cyklů

- Jazyk Java
  - Tři různé příkazy cyklu – `while`, `do-while`, `for`
  - Všechny tři cykly – řízené podmínkou
  - Pří splnění podmínky – opakované provádění těla cyklu
  - Nesplnění podmínky – ukončení cyklu, pokračování dalším příkazem za cyklem
  - `do-while` – podmínka na konci cyklu, tělo cyklu se provede minimálně jednou
  - `for` – záhlaví cyklu obsahuje kromě podmínky i inicializační a iterační část – zápis může být kompaktnější a přehlednější
  - Všechny tři příkazy cyklu jsou navzájem zaměnitelné – všechny tři příkazy mají prakticky stejné vyjadřovací schopnosti (jedná se o stejně silné nástroje) – libovolný příkaz cyklu můžeme zapsat pomocí libovolného jiného příkazu cyklu

- Při používání cyklů – **nutno zabránit realizace tzv. nekonečného cyklu**
  - Příkazy cyklu Java – řízené podmínkou
  - Pokud vykonávání kódu vstoupí do těla cyklu (podmínka cyklu nabyla hodnoty `true`) – příkazy v těle cyklu při konečném počtu opakování musí měnit/ovlivňovat hodnotu podmínky cyklu, tak aby tato podmínka v konečném počtu kroků nabyla hodnoty `false`

Poznámky

- Zadaná úloha – řešení zadaní úlohy
  - Specifikace zadání
  - Řešení – program, který poskytuje správné výsledky pro všechny kombinace vstupů definované zadáním
  - Ošetření chybových stavů
  - Ošetření chybného vstupu/zadání od uživatele

- Formátování textu programu
  - Odsazování – zarovnání
  - Příkazový blok
  - Umístění `else`
  - Umístění `while` cyklu `do-while`

DALŠÍ PŘÍKAZY

Příkaz `switch`

- **Syntaxe** příkazu (způsob zápisu)

```
switch (<výraz>) {
   case <hodnota1> : <větev1>
   case <hodnota2> : <větev2>
   ...
   case <hodnotaN> : <větevN>;
   default : <větevD>;
}
```

  - `switch, case, default` – rezervovaná slova jazyka Java
  - `<výraz>` … – výraz jazyka Java
  - `<hodnota1>` … – hodnota typově kompatibilní s typem výsledné hodnoty výrazu `<výraz>`
  - `<větev1>` … – příkaz, série příkazů nebo příkazový blok

- **Sémantika** (význam, způsob vykonání)
  1. Vyhodnotí se výraz `<výraz>`
  2. Hodnota výrazu se vyhledá v jednotlivých `case` větvích
  3. Hodnota výrazu je nalezena, provedou se příkazy v dané větvi a následující
  4. Hodnota výrazu není nalezena, provedou se příkazy ve větvi `default`
  5. Vykonání příkazu `break` uvnitř příkazu `switch` ukončí provádění příkazu `switch`

Použití příkazu switch

- Úloha – výpočet zadané aritmetické mezi dvěma čísly
- Příklad komunikace s programem

```
Zadej ulohu
12 * 3
36
```

```
Scanner sc = new Scanner(System.in);
double a, b, vysledek;
char operator;

System.out.println(”Zadej ulohu”);
a = sc.nextDouble();
operator = sc.next().charAt(0);
b = sc.nextDouble();

 switch (operator) {
    case '+': {vysledek = a + b; break; }
    case '*': {vysledek = a * b; break; }
    case '-': {vysledek = a - b; break; }
    case '/': {vysledek = a / b; break; }
    default : {vysledek = Double.NaN;}
}
System.out.println(vysledek);
```

Příkazy `break` a `continue`

- Příkaz `break` ukončí
  - vykonávání cyklu
  - vykonávání bloku příkazů
  - vykonávání příkazu `switch`

- Použití

  `break;`
  - Ukončí vykonávání **nejvnitřnějšího** cyklu nebo bloku příkazů nebo příkazu `switch`

  `break` `<návěští>;`
  - Ukončí vykonávání cyklu nebo bloku příkazů, jehož začátek je označen příslušným návěštím

- Příkaz `continue` ukončí aktuální vykonávání těla cyklu, běh programu pokračuje poté dalším krokem cyklu

- Použití

  `continue;`
  - Ukončí vykonávání **nejvnitřnějšího** cyklu nebo bloku příkazů nebo příkazu `switch`

  `continue` `<návěští>;`
  - Ukončí vykonávání aktuálního kroku cyklu, jehož začátek je označen příslušným návěštím

Přehled zbývajících příkazů

- Příkaz `return` – ukončení vykonávání metody, bude uvedeno v souvislosti s metodami
- Příkaz `throw` – generování výjimky a její předání systému obsluhy výjimek, bude uvedeno v souvislosti s generováním a obsluhou výjimek
- Příkaz `try` (`try-catch`, `try-finally`) – vytvoření chráněného bloku – části kódu, ve které potenciálně očekáváme výskyt chybových stavů, , bude uvedeno v souvislosti s generováním a obsluhou výjimek
- Příkaz `assert` – generování výjimky na základě nesplnění podmínky, použití především ve fázi ladění programu
- Příkaz `synchronized` – vytvoření (uvození) tzv. synchronizované kritické sekce – používá se u úseků kódu, jejichž paralelní běh může způsobit problémy