A0B36PRI - PROGRAMOVÁNÏ

Datové typy, výrazy vstup, výstup

Jazyk JAVA

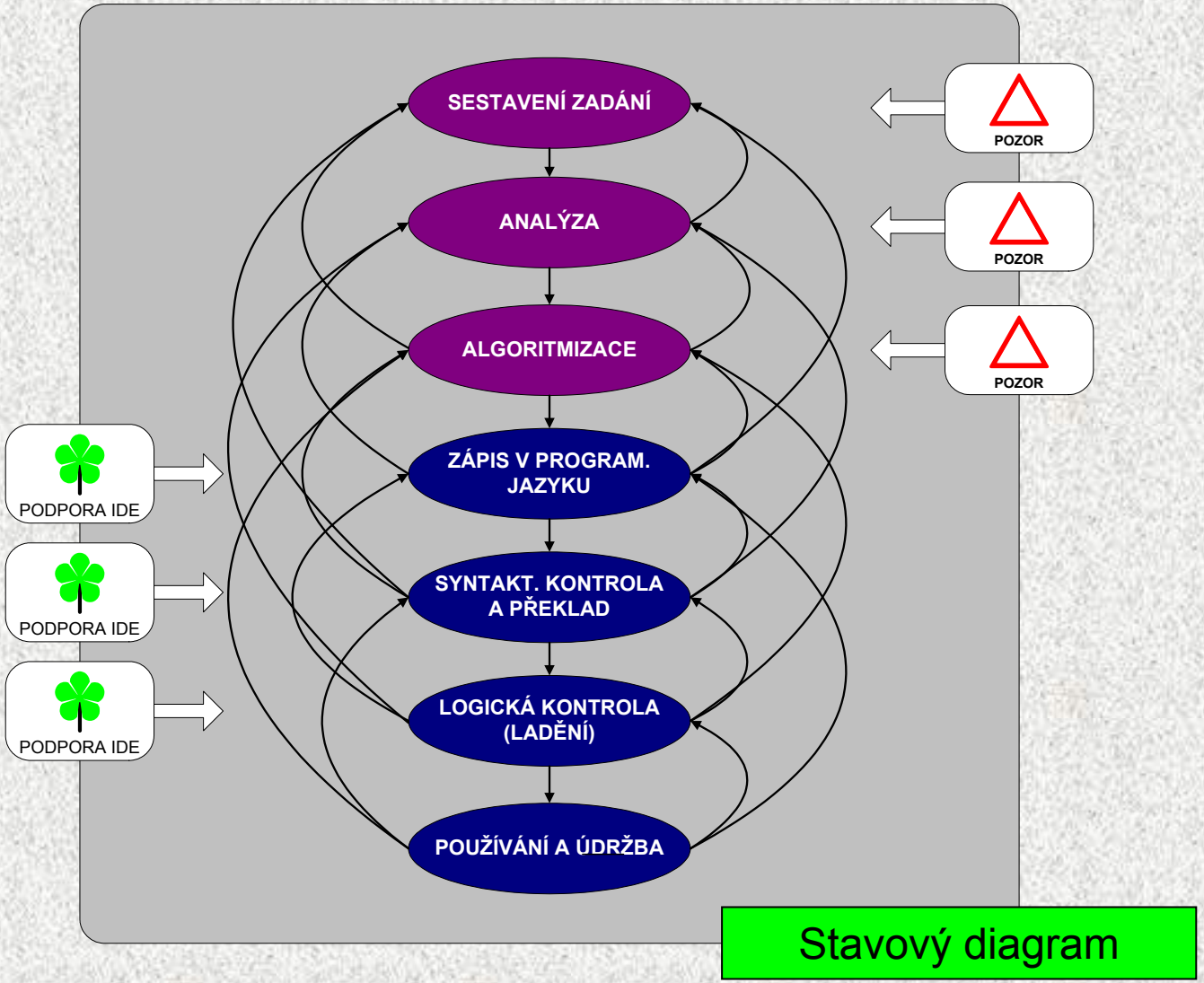
České vysoké učení technické Fakulta elektrotechnická

v1.02

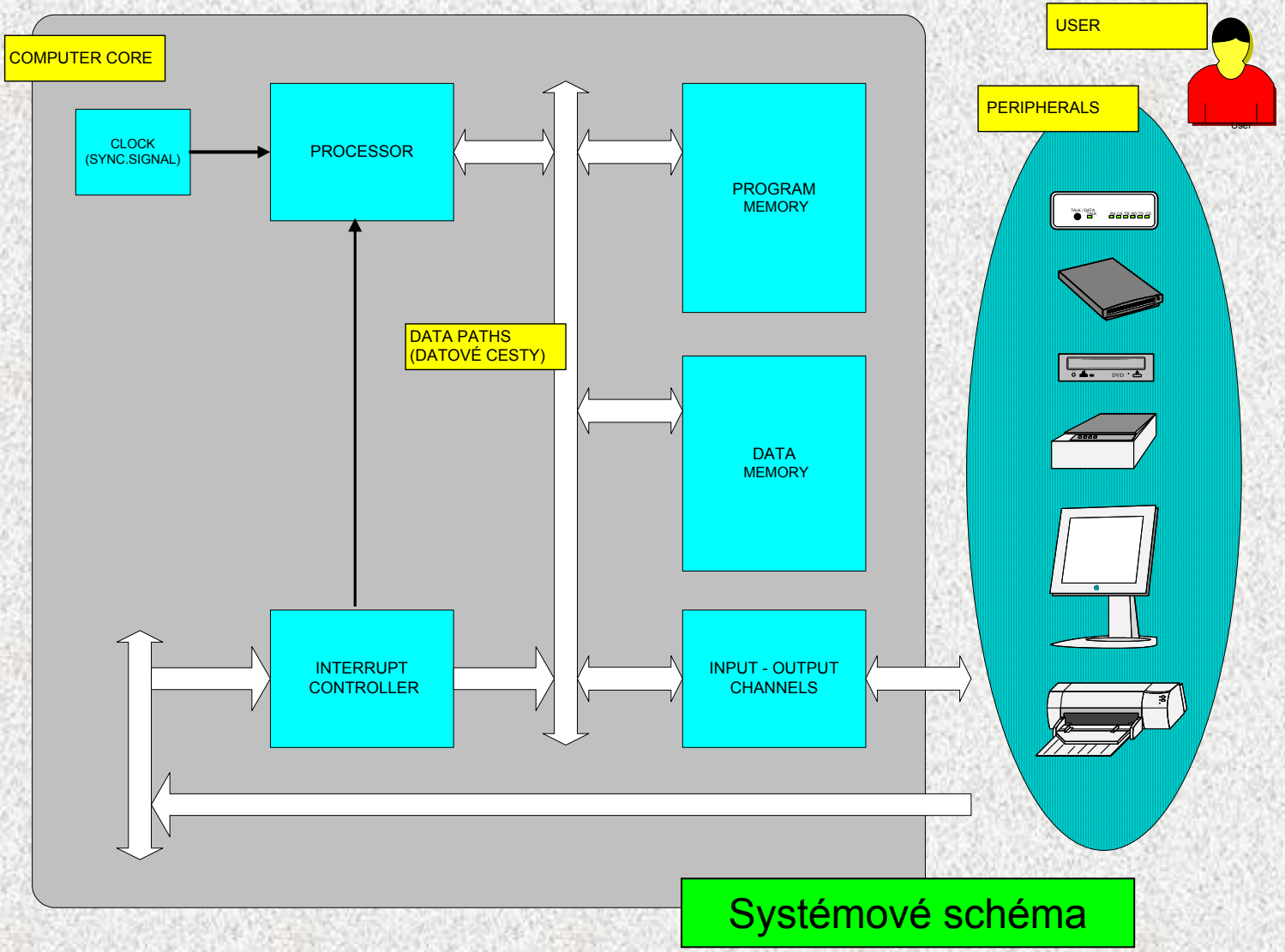
Obsah

- Postup při řešení problému počítačem
- Počítač, periferie a uživatel
- Výpočetní proces a paměť počítače
- Datové typy
- Proměnné a přiřazení
- Jednoduché datové typy
- Reprezentace dat v počítači
- Deklarace proměnných
- Literály a pojmenované konstanty
- Přiřazovací příkaz
- Typové konverze
- Příkazy výstupu – print(), println()
- Formátovaný výstup - printf()
- Příkaz vstupu - třída Scanner
- Výrazy
- Aritmetické operátory
- Relační operátory
- Logické operátory
- Matematické funkce
- Knihovní třída Math
- Další základní datové typy

Řešení problému pomocí počítače



Počítač, periferie a uživatel



Systémové schéma

Výpočetní proces a paměť počítače

- Výpočetní proces je posloupnost akcí nad daty uloženými v paměti počítače
- Data jsou v paměti reprezentována posloupnostmi bitů (bit = 0 nebo 1)

- Připomeňme:
  - paměť je tvořena řadou 8-mi bitových paměťových míst nazývaných bajt (z angl. byte, česky též slabika)
  - rozlišujeme vnitřní (operační) paměť a vnější paměť (např. disk)
  - každé paměťové místo vnitřní paměti má svou adresu (nezáporné celé číslo), která slouží pro jeho identifikaci
  - kapacita paměti se udává v MB (1 MB = 2^{20}B) nebo GB (1 GB = 2^{30}B)

- Instrukce strojového jazyka předepisují aritmetické, logické a jiné operace s posloupnostmi bitů bez ohledu na to, jaká data posloupnost bitů reprezentuje

Datové typy

- Při návrhu algoritmů a psaní programů ve vyšších programovacích jazycích abstrahujeme od binární podoby paměti počítače
- S daty pracujeme jako s hodnotami různých datových typů, které jsou uloženy v datových objektech
- Datový typ (zkráceně jen typ) specifikuje:
  - množinu hodnot
  - množinu operací, které lze s hodnotami daného typu provádět
- Příklad typu: celočíselný typ **`int`** v jazyku Java:
  - množinou hodnot jsou celá čísla z intervalu -2147483648 .. 2147483647
  - množinu operací tvoří
    - aritmetické operace +, -, *, /, jejichž výsledkem je hodnota typu *int*
    - relační operace ==, !=, >, >=, <, <=, jejichž výsledkem je hodnota typu **`boolean`**
    - a další
- Typ **`int`** je jednoduchý typ, jehož hodnoty jsou atomické (z hlediska operací dále nedělitelné)

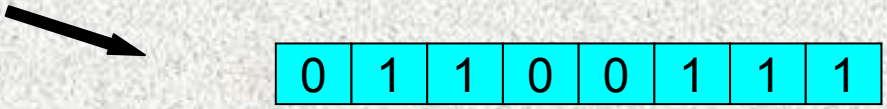
Primitivní či základní datové typy

Typ	Bitů	Rozsah		Obal.třída
Celočíselný typ				
byte	8	-128 … 127		Integer
short	16	-32768 … 32767		Short
int	32	-2147483648 … 2147483647		Int
long	64	-9223372036854775808 … 9223372036854775807		Long
Reálný typ, IEEE 754 (NaN, infinity)				
float	32	2^{-149} … $(2-2^{-23})*2^{127}$		Float
double	64	2^{-1074} … $(2-2^{-52})*2^{1023}$		Double
Znaky, UCS2				
char	16	'\u0000' to '\uffff‘ 0 ... 65535		Character
Logický typ				
boolean	1/8	true false		Boolean
Pomocný prázdný typ				
void				

Reprezentace dat v počítači

- **bit** (**bi**nary digi**t** – dvojková číslice; angl. bit – drobek, kousek)
  - základní a nejmenší jednotky informace
  - nabývá hodnot 0 nebo 1

- **byte** (též bajt, česky - slabika) – uspořádaná osmice bitů



  - Násobky
  - 1 kB = 1 000 bajtů (dekadický kilobajt, malé „k“)
  - 1 KB = 1 024 bajtů = 2^{10} (binární kilobajt, velké „K“)

  - 1 MB = 1 048 576 bajtů = 2^{20}
  - 1 GB = 1 073 741 824 bajtů = 2^{30}

Reprezentace dat v počítači

- Polyadické číselné soustavy

$$X_d = \sum_{i=-n}^{i=m} a_i z^i, \quad a \text{ - číslice soustavy, } z \text{ - základ soustavy}$$

- Desítková soustava (decimal numeral system)
  - a = „0“, „1“, „2“,… „9“; z = 10
  - Př: $125_d = 1*10^2 + 2*10^1 + 5*10^0 = 100 + 20 + 5$

- Šestnáctková soustava (hexadecimal numeral system)
  - a = „0“, „1“, „2“,… „9“, „A“, „B“,… „F“; z = 16
  - Př: $125_d = 0*16^2 + 7*16^1 + D*16^0 = 0 + 112 + 13 = 07D_h$

- Dvojková soustava (binary numeral system)
  - a = „0“, „1“; z = 2
  - Př: $125_d = 0*2^7+1*2^6+1*2^5+1*2^4+1*2^3+1*2^2+0*2^1+1*2^0 =$
    $= 0 + 64 + 32 + 16 + 8 + 4 + 0 + 1 = 01111101_b$

Reprezentace dat v počítači

- Celá kladná čísla – přímý binární kód
  - Př: $125_d = 0*2^7+1*2^6+1*2^5+1*2^4+1*2^3+1*2^2+0*2^1+1*2^0 =$
    $= 0 + 64 + 32 + 16 + 8 + 4 + 0 + 1 = 01111101_b$

- Celá záporná čísla – dvojkový doplněk (two's complement)
  - Př: $-125_d = 10000011_b$

    Výpočet:

    Kladné číslo: $x = +125_d = 0111\ 1101_b$

    $NOT\ x = 1000\ 0010_b$ (jednotkový doplněk)

    $+1$

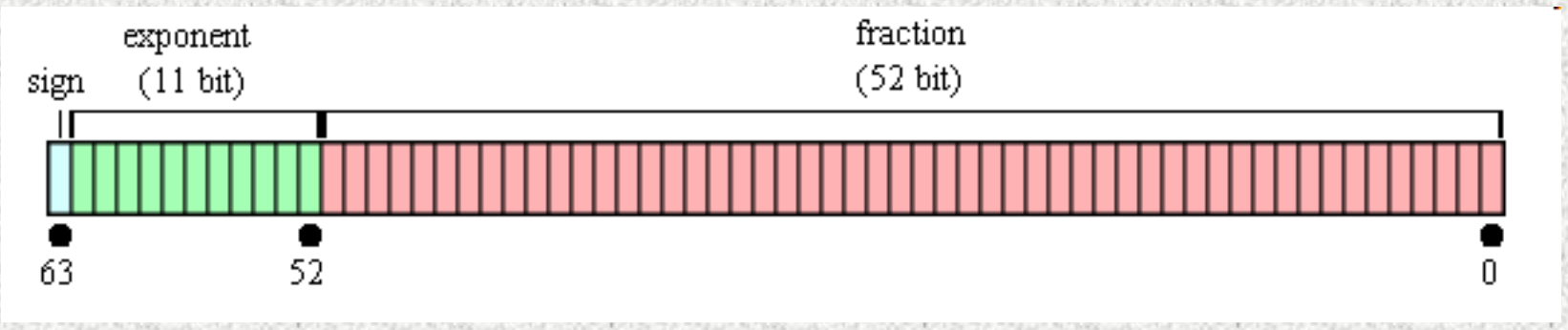
    Záporné číslo: $-x = -125_d = 1000\ 0011_b$ (dvojkový doplněk)

    znaménkový bit

    velikost čísla (není to přímý binární kód!)

Reprezentace dat v počítači

- Reálná čísla (čísla necelá)
- Norma IEEE 754
  - Př: typ double – 64 bitů



- $Xd = (-1^{sign}) * 2^{(exponent - 1023)} * 1.fraction$

Charakteristika

Mantisa

Znaménkový bit

Implicitní bit – neuchovává se v paměti

- Reálné číslo je vždy po výpočtu upraveno (normalizováno) tak, aby hodnota mantisy byla v rozsahu 1.0 – 1.99….

Reprezentace dat v počítači

- Reálná čísla (čísla necelá)
- Př: Zápis čísla 178.25_d ve formátu IEEE 754 (ve float = 32 bitů)
  a) $178.25_d = 10100010.001_b$
  b) Normalizace:

$$10100010.001 = 1.0100010001 * 2^{7d} = 1.0100010001 * 2^{111b}$$

Charakteristika = e = 7_d

  c) Formát IEEE 754 – float = single precision

$$Xd = (-1^{sign}) * 2^{(exponent - 127)} * 1.fraction$$

$$e = exponent - 127 \Rightarrow exponent = e + 127 = 7 + 127 = 134_d = 10000110_b$$

  d) Zápis čísla 178.25_d ve formátu float

Znaménkový bit

Implicitní bit – neuchovává se v paměti

$$0\ 10000110\ 1.01000100010000000000000_b = 0x43222000$$

Charakteristika

Mantisa

Celá čísla v Javě – typ int

int je reprezentováno 32 bity

nejmenší číslo *1000...000* = $-2^{31} = -2147483648$

největší číslo *0111...111* = $2^{31} - 1 = -2147483647$

Poznámka:

Pozor na důsledky počítání (v doplňkovém kódu).:

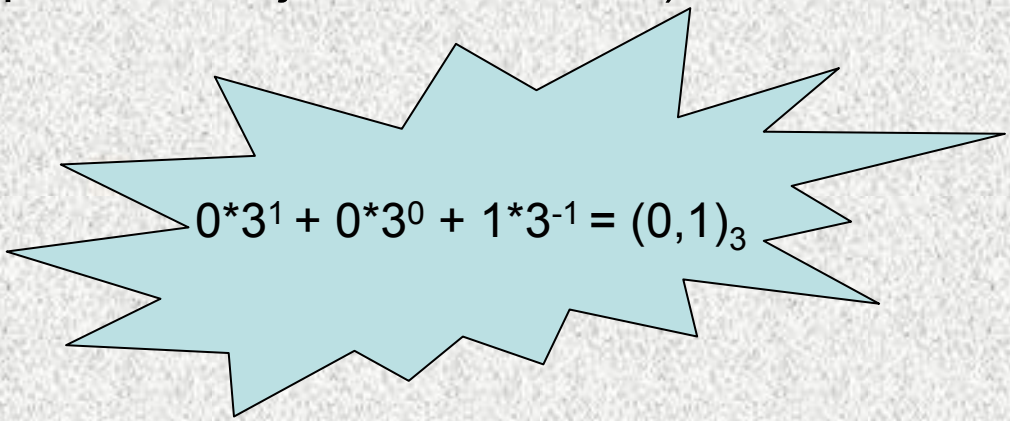
2 000 000 000 + 1 000 000 000 = -1 294 967 296

Math.abs(-2147483648) = -2147483648 !! záporné číslo

Nepřesnost v zobrazení čísel

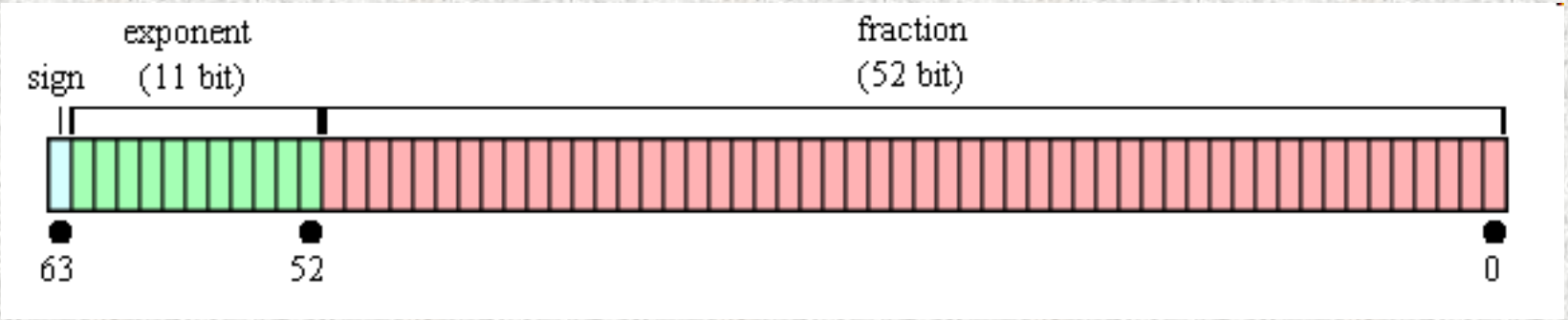
- pokusme se v dekadické soustavě zapsat číslo 1/3 (jedna třetina)
  - 0,3
  - přesněji 0,33
  - lépe 0,333
  - ...
  - 0,33333333333333333333333333333333333

- jelikož máme omezený paměťový prostor, nikdy neuložíme číslo 1/3 přesně (Jak by číslo 1/3 vypadalo v trojkové soustavě?)



Necelá čísla v Javě – typ double

double je reprezentován 64 bity, norma IEEE 754



znaménkový bit (s), exponent, mantisa

nejmenší čísla v normalizovaném tvaru (v abs. hodnotě)

$\pm 2^{-1022} \approx \pm 2.2250738585072020 \times 10^{-308}$

největší čísla

$\pm(1-(1/2)^{53})2^{1024} \approx \pm 1.7976931348623157 \times 10^{308}$

Pozor!! 1E100+5000000000000L = 1.0E100

Nepřesnost v zobrazení čísel

- nepřesnosti způsobují

1. Iracionální **čísla** (e ,$\sqrt{2}$, π)
2. Čísla, jež mají v dané soustavě periodický rozvoj (1/3 ve dvojkové či dekadické soustavě, 1/10 ve dvojkové)
3. Čísla, která mají příliš dlouhý zápis:

   double

   – 1bit znaménko – dvě možnosti, +,-
   – 11 bitů exponent – 2048 možností
   – 52 bitů – 4 503 599 627 370 496 možností, tedy asi 4,5 biliardy

   - není možné v typu double přesně uložit čísla se zápisem delším než 52 bitů!

   - čím větší exponent tím větší „mezery“ mezi sousedními čísly

Pro info

Model reprezentace reálných čísel

Reálná čísla se zobrazují jako aproximace daných rozsahem paměťového místa
Reálná čísla se zobrazují ve tvaru:

X= mantisa*základ exponent = m*z exponent

Mantisa musí být normalizována:

0,1<=m<1, důvod: jednoznačnost zobrazení

Model:

- délka mantisy 3 pozice + znaménko
- délka exponentu 2 pozice + znaménko

Příklad

X= 77.5 = +0.775*z+02

- zakódujeme znaménko +=0, - = 1
- z je 10, 16, 2,..

+	775	+	02

1	775	1	02

Model reprezentace reálných čísel

Maximální zobrazitelné kladné číslo

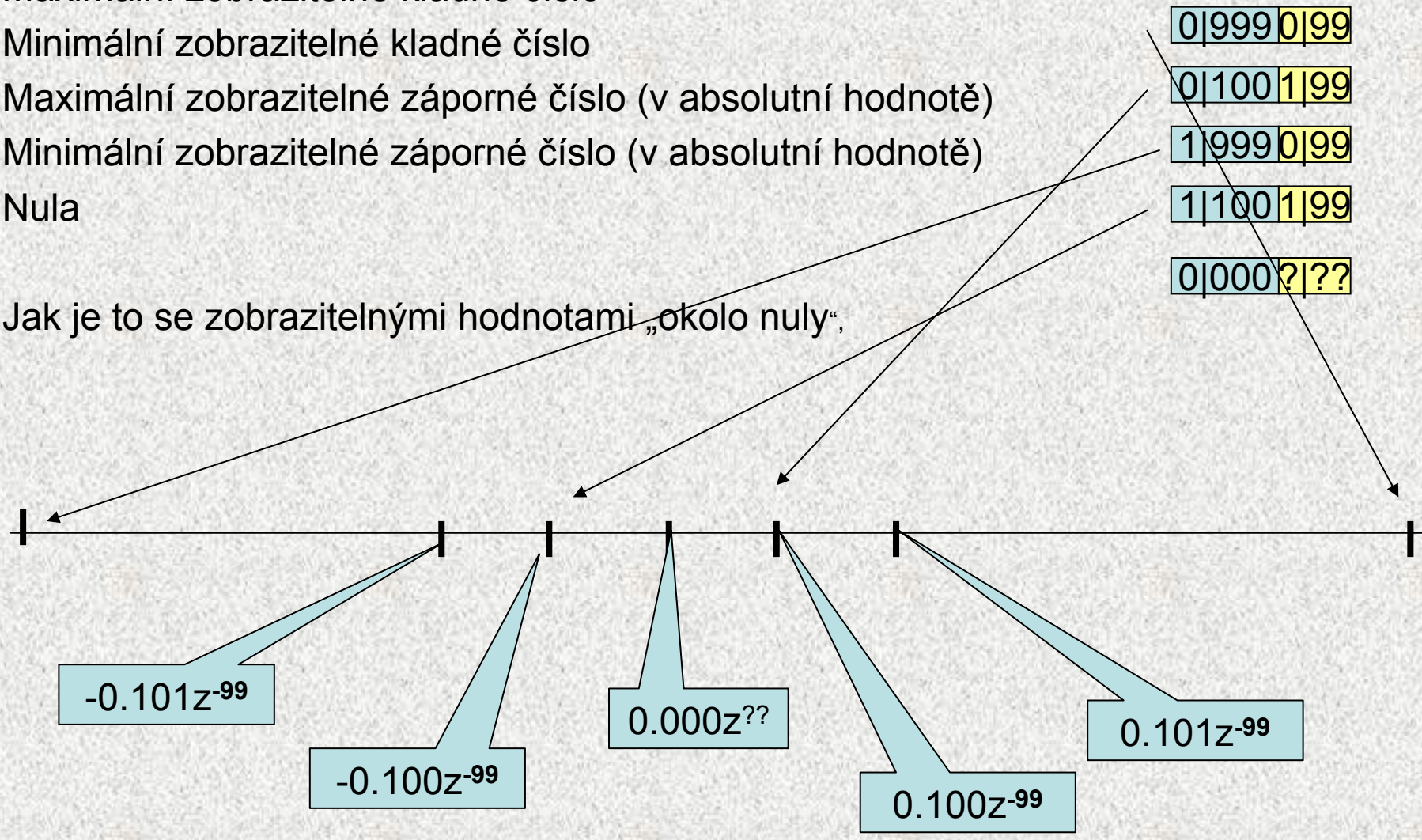
Minimální zobrazitelné kladné číslo

Maximální zobrazitelné záporné číslo (v absolutní hodnotě)

Minimální zobrazitelné záporné číslo (v absolutní hodnotě)

Nula

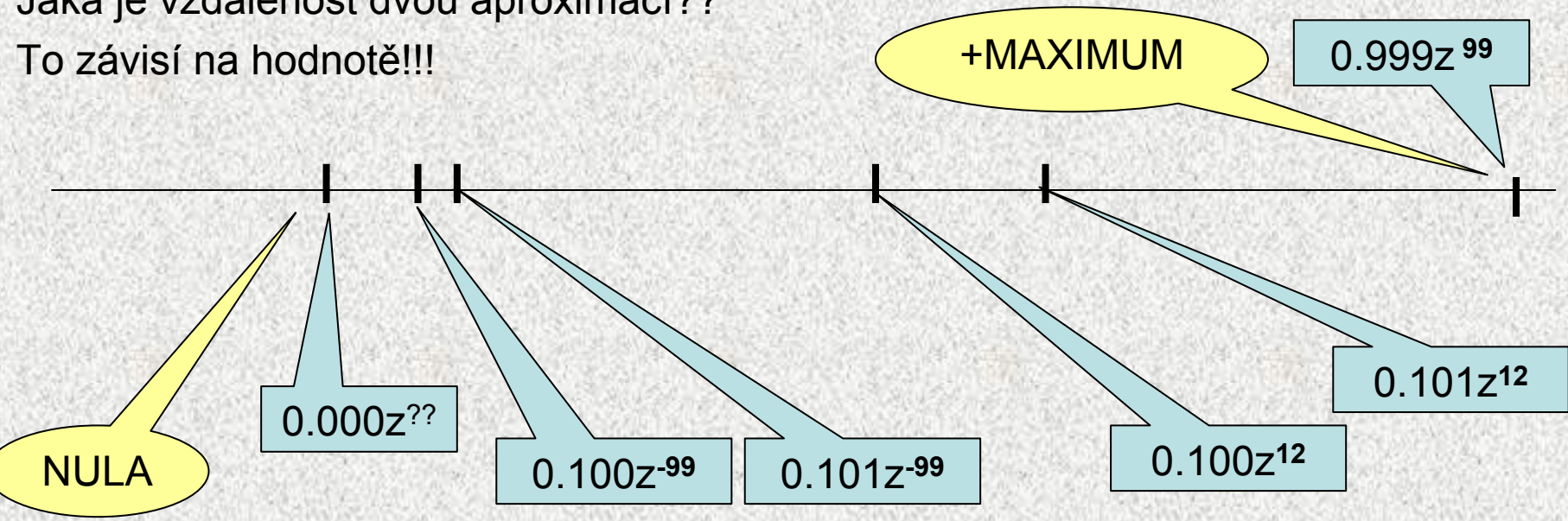
Jak je to se zobrazitelnými hodnotami „okolo nuly“,



Model reprezentace reálných čísel

Jaká je vzdálenost dvou aproximací??

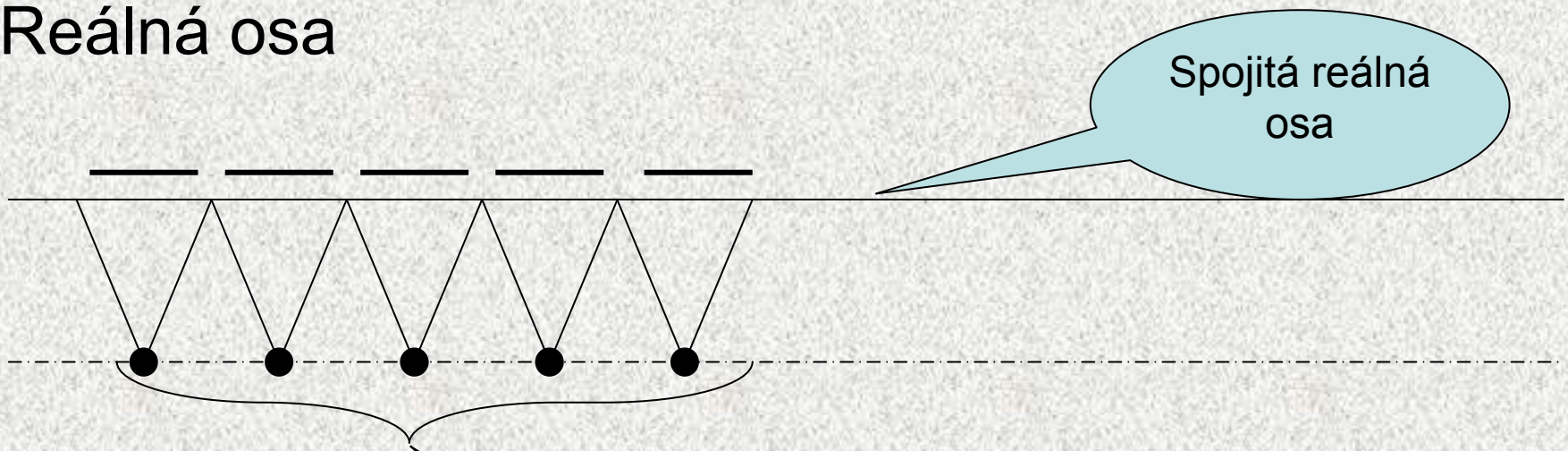
To závisí na hodnotě!!!



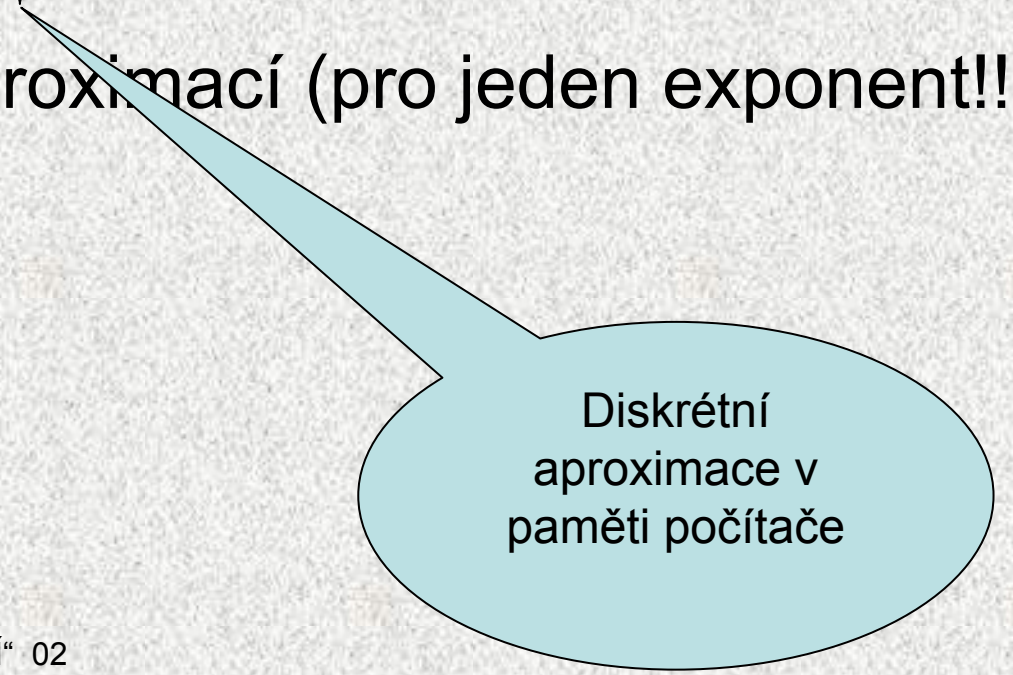
Aproximace reálných čísel: nejsou na číselné ose rovnoměrně rozložené!!

Reprezentace reálných čísel

Reálná osa



Zobrazení aproximací (pro jeden exponent!!)



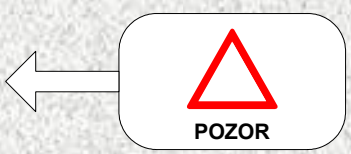
Deklarace proměnných

- Proměnné se zavádějí deklaracemi
- Příklady deklarací proměnných:

```
int i;          // deklarace proměnné i typu int
double x;       // deklarace proměnné x typu double
```

- Proměnná deklarovaná uvnitř funkce (lokální proměnná) nemá deklarací definovanou hodnotu
- Použití proměnné s nedefinovanou hodnotou v jazyku Java je chyba při překladu
- Příklad:

```
int x, y;
x = y + 2;      // chyba při překladu
```



- Deklaraci proměnné lze doplnit o inicializaci proměnné:

```
int x = 10;     // deklarovaná proměnná má hodnotu 10
```

- Deklarací lze zavést několik proměnných stejného typu:

```
int x, z;
```

Literály a pojmenované konstanty

- Přímý zápis hodnoty v programu se nazývá literál
  - **`int`** 34 45
  - **`double`** 25.3 1.2E-3
  - **`boolean`** **`false true`**
  - **`char`** **`'a' '1' '+'`**
- Dalším často potřebným literálem je literál typu **`String`** (není to jednoduchý typ): **`"abcd" "Nazdar"`**
- Kromě literálů lze v jazyku Java používat pojmenované konstanty, které se deklarují podobně jako inicializované proměnné, v deklaraci je navíc klíčové slovo **`final`**
- Příklad deklarace konstant:

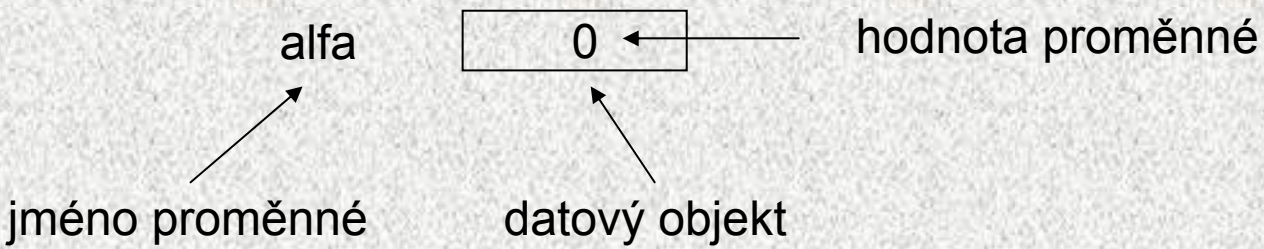
```
final int MAX = 100;
final String NAZEV_PREDMETU = "Programovani";
```

- Konvence (dohoda): Jména konstant píšeme velkými písmeny
- Konstantě nelze změnit hodnotu přiřazovacím příkazem

```
MAX = 20;              // chyba při překladu
```

Proměnné a přiřazení

- Proměnná je datový objekt, který je označen jménem a je v něm uložena hodnota nějakého typu, která se může měnit



- V jazyku Java zavedeme výše uvedenou proměnnou deklarací

```
int alfa = 0;
```

- Hodnotu proměnné lze změnit přiřazovacím příkazem


```
alfa = 37;
```

Přiřazovací příkaz

- Slouží pro přiřazení hodnoty proměnné
- Tvar přiřazovacího příkazu:

  <*proměnná*> = <*výraz*>;
- Příklad:

```
x = y + z;
```

  proměnné *x* se přiřadí součet hodnot proměnných *y* a *z*

```
x = x + 1;
```

  hodnota proměnné *x* se zvětší o 1
- Proměnné v jazyku Java lze přiřadit pouze hodnotu jejího typu
- Příklady nedovolených přiřazovacích příkazů:

```
boolean b; int i; double d;
b = 1;
i = 1.4;
d = true;
```

Přiřazovací příkaz

- přířazovací operátory:

<proměnná> = <proměnná> <OP> <výraz> ~ *<proměnná> <OP>= <výraz>*

Př.:

```
j += 5; ~ j = j + 5;
```

- přiřazení je výraz !!!:

Příklad:

```
int x, y;
x = 7;              // má hodnotu 7 a lze tedy opět přiřadit!!
y = x = x + 6;      // vyhodnotí se jako y = (x = (x + 6));
```

Typové konverze

- Typová konverze je operace, která hodnotu nějakého typu převede na hodnotu jiného typu
- Typová konverze může být *implicitní* (vyvolá se automaticky) nebo *explicitní* (v programu je třeba ji explicitně předepsat)
- Konverze typu **`int`** na **`double`** je v jazyku Java implicitní:
  - kde se očekává hodnota typu **`double`**, může být uvedena hodnota typu **`int`**, která se automaticky převede na hodnotu typu *double*
- Příklad:

```
double x; int i = 1;
x = i;    // hodnota 1 typu int se automaticky převede na
          // hodnotu 1.0 typu double
```

- Převod hodnoty typu **`double`** na **`int`** (odseknutím necelé části) je třeba explicitně předepsat
- Příklad:

```
double x = 1.2; int i;
i = (int)x; // hodnota 1.2 typu double se odseknutím
            // necelé části převede na hodnotu 1 typu int
```

Příkazy výstupu

- Pro výpis dat na obrazovku se v Javě nejčastěji používá příkaz

  ```
  System.out.println(parametr);
  ```

- Data daná parametrem se vypíší na aktuální řádek a přejde se na další řádek
- Pro výpis dat na aktuální řádek bez přechodu na další řádek lze použít příkaz

  ```
  System.out.print(parametr);
  ```

- Parametrem musí být výraz typu *String* (řetězec)
- Příklad výpisu textu:

  ```
  System.out.println("Nazdar");
  ```

- Pro každý jednoduchý typ existuje implicitní konverze na typ *String*, tzn. parametrem příkazu výstupu může být proměnná (výraz) jednoduchého typu
- Příklad výpisu hodnoty proměnné *n* typu *int*:

  ```
  System.out.println(n);
  ```

- Řetězce lze spojovat pomocí operátoru +; je tedy dovolen např. tento příkaz:

  ```
  System.out.println("hodnota proměnné n = " + n);
  ```

Příkazy výstupu

Př.

```
package pri02;

public class DruhyProgram {
  public static void main(String[] args) {
    int x = 10, y;
    y = x + 20;
    System.out.println("hodnota proměnné x je "+x);
    System.out.println("hodnota proměnné y je "+y);
  }
}
```

- Program po překladu a spuštění vypíše:

```
hodnota proměnné x je 10
hodnota proměnné y je 30
```

Příkaz výstupu - poznámka

- Co vypíší následující příkazy?:

příkazy	výstup
`int x = 1, y = 2;` `System.out.println(x+y);`	`3`
`int x = 1, y = 2;` `System.out.println("součet je "+(x+y));`	`součet je 3`
`int x = 1, y = 2;` `System.out.println("součet je "+x+y);`	`součet je 12`

- Příčinou „podivného“ výstupu posledního programu je, že operátor + (a většina dalších) je asociativní zleva, tzn. výraz `"součet je " + x + y`
  se vyhodnotí takto: `("součet je " + x) + y`

Formátovaný výstup - printf( )

- **`System.out.printf("Cislo Pi = %6.3f %n", Math.PI);`**

  Výstup na obrazovku: Cislo Pi = 3.142

- Specifikace formátu
  `%[$indexParametru][modifikátor][šířka][.přesnost]konverze`
- **konverze** - povinný parametr
  - typ celé číslo d,o,x - dekadicky, oktalově a hexadecimálně
  - typ double f je desetinný zápis; e,E vědecký s exponentem
- **šířka** - počet sázených míst, zarovnání vpravo
- **.přesnost** - počet desetinných míst
- **modifikátor** - v závislosti na typu konverze určuje další vlastnosti, například pro konverzi f (typ double)
  - symbol **+** určuje, že má být vždy sázeno znaménko,
  - symbol **-** určuje zarovnání vlevo,
  - symbol **0** doplnění čísla zleva nulami.
- **%n** přechod na další řádek
- a další (formátování data, měny, …)

Příkaz vstupu - třída Scanner

- V dalších příkladech budeme potřebovat příkazy pro vstup číselných dat zadaných na klávesnici

  Příklad:

```
package pri02;
import java.util.*; // Scanner je v knihovně java.util

public class TretiProgram {
   static {Locale.setDefault(Locale.US);}

   public static void main(String[] args) {
     Scanner sc = new Scanner(System.in);
     int x, y, z;
     System.out.println("Zadejte dvě celá čísla");
     x = sc.nextInt();
     y = sc.nextInt();
     z = x + y;
     System.out.println("Součet čísel:"+x+" + "+y+" = "+z);
     }
}
```

Příkaz vstupu - třída Scanner

- Využití služeb třídy *Scanner* je třeba deklarovat příkazem
  `import java.util.*;`
- Je třeba vytvořit objekt třídy Scanner a napojit jej na standardní vstupní proud
  `Scanner sc = new Scanner(System.in);`
- Třída *Scanner* poskytuje tyto služby:
  `sc.nextInt()`
    - přečte celé číslo z řádku zadaného klávesnicí (řádek je zakončen klávesou *Enter,* číslo je zakončeno mezerou nebo *Enter*) a vrátí je jako funkční hodnotu typu *int*

  `sc.nextDouble()`
    - přečte číslo z řádku zadaného klávesnicí a vrátí je jako funkční hodnotu typu *double,* jako oddělovač použijte . (tečku) nebo čárku v závislosti na definovaném jazyku OS. Lze změnit před vytvořením Scanneru pomocí
      `static {Locale.setDefault(Locale.US);}`

  `sc.nextLine()`
    - přečte zbytek řádku zadaného klávesnicí a vrátí je jako funkční hodnotu typu *String*

Výrazy

- Výraz předepisuje výpočet hodnoty určitého typu
- Výraz může obsahovat:
  - proměnné
  - konstanty
  - volání funkcí
  - binární operátory
  - unární operátory
  - závorky
- Pořadí operací předepsaných výrazem je dáno:
  - prioritou operátorů
  - asociativitou operátorů
- Příklad:

výraz	pořadí operací	zdůvodnění
`x + y * z`	`x + (y * z)`	* má vyšší prioritu než +
`x + y + z`	`(x + y) + z`	+ je asociativní zleva

Aritmetické operátory

- Pro operandy typu **`int`** a **`double`** budeme požívat tyto aritmetické operátory (seřazeno sestupně podle priority):
  - unární **`-`** (změna znaménka)
  - binární **`*, /, %`** (násobení, dělení a zbytek po dělení)
  - binární **`+`** a **`-`** (sčítání a odčítání)
- Jsou-li oba operandy stejného typu, výsledek aritmetické operace je téhož typu
- Jsou-li operandy různého typu, operand typu *int* se implicitní konverzí převede na hodnotu typu *double* a výsledkem operace je hodnota typu *double*
- Výsledkem dělení operandů typu *int* je celá část podílu
  např. **`7/3`** je 2 **`-7/3`** je -2

- Pro zbytek po dělení platí: ***`x % y = x - (x / y) * y`***
  např: **`7%3`** je 1; **`-7%3`** je -1; **`7%-3`** je 1; **`-7%-3`** je -1

- speciální inkrementační a dekrementační operátory:
  `++x, x++, --x, x--`

Relační operátory

- Hodnoty všech jednoduchých typů jsou uspořádané a lze je porovnávat relačními operátory
- Budeme používat tyto relační operátory (priorita je menší než priorita aritmetických operátorů):
  - **`>, <, >=, <=`** (větší než, menší než, větší nebo rovno, menší nebo rovno)
  - **`==, !=`** (rovná se, nerovná se)

- Výsledek relační operace je typ **`boolean`** (**`true`**, když relace označená operátorem platí, **`false`** v opačném případě)
- Jestliže při porovnávání číselných hodnot jsou operandy různého typu, operand typu *int* se implicitní konverzí převede na hodnotu typu *double*
- Relační operátory mají menší prioritu, než aritmetické operátory
- Příklady relačních výrazů (relací):

```
int i = 10; double x = 12.3; boolean b;
System.out.println(i == 10);  // vypíše true
System.out.println(i+1 == 10);// vypíše false
b = i > x;                    // proměnné b se přiřadí false
```

Logické operátory

- Logické operátory jsou definovány pro hodnoty typu *boolean*
  - unární ! (negace)
  - binární && resp. & (konjunkce, logický součin)
  - binární || resp. | (disjunkce, nevýhradní logický součet, OR)
  - binární ^ (disjunkce, výhradní logický součet, XOR)

x	y	!x	x && y	x || y	x ^ y
false	false	true	false	false	false
false	true	true	false	true	true
true	false	false	false	true	true
true	true	false	true	true	false

- Negace má stejnou prioritu, jako změna znaménka, logický součin má nižší prioritu než relační operátory
- Operace && a || se vyhodnocují zkráceným způsobem, tj. druhý operand se nevyhodnocuje, jestliže lze výsledek určit již z prvního operandu
- Příklady:

```
int n = 10; boolean b1 = false, b2 = true;
System.out.println(1 <= n && n <= 20);   // vypíše se true
System.out.println(b1 || !b2);           // vypíše se false
if (y != 0 && x/y < z)                   // zkrácené vyhodnocení
```

Matematické funkce

- Při číselných výpočtech často potřebujeme matematické funkce jako *abs, sin, cos, sqrt* (druhá odmocnina), *log* (přirozený logaritmus), *log10* atd.
- Tyto funkce poskytuje knihovní třída *Math*
- Příklad:

```
import java.util.*;

public class Prepona {
  public static void main(String[] args) {
   Scanner sc = new Scanner(System.in);
   System.out.println("Zadejte odvesny pravouhl.trojúhelníka");
   double x = sc.nextDouble();
   double y = sc.nextDouble();
   double z = Math.sqrt(x*x+y*y);
   System.out.println("Délka přepony je "+z);
  }
}
```

- Knihovní třídu *Math* není třeba importovat příkazem *import*

Knihovní třída Math

- Některé z funkcí poskytovaných třídou *Math*

```
public static int abs(int a)
public static double abs(double a)

public static int max(int a, int b)
public static double max(double a, double b)
public static int min(int a, int b)
public static double min(double a, double b)

public static double sqrt(double a)
public static double sin(double a)

public static double random()
```

  - vrátí pseudonáhodné číslo větší nebo rovno 0.0 a menší než 1.0

- Třída poskytuje též dvě konstanty: E a PI

Knihovní třída Math

Příklad:

```
// Tridu Math neni treba importovat
import java.util.*;

   public class ObvodKruhu {
     public static void main(String[] args) {
       Scanner sc = new Scanner(System.in);
       System.out.println("zadejte poloměr kruhu");
       double r = sc.nextInt();
       System.out.println("obvod kruhu je "+2*Math.PI*r);
     }
   }
```

A0B36PRI - PROGRAMOVÁNÏ

Datové objekty, výrazy vstup, výstup

Jazyk JAVA

Konec

České vysoké učení technické Fakulta elektrotechnická