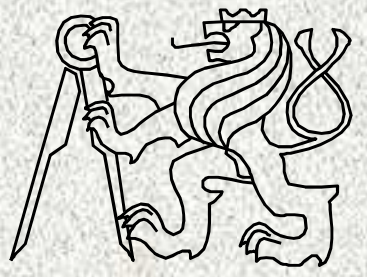
Java – reprezentace dat, výrazy



A0B36PR1-Programování 1

Fakulta elektrotechnická
České vysoké učení technické

Dva základní přístupy k imperativnímu programování

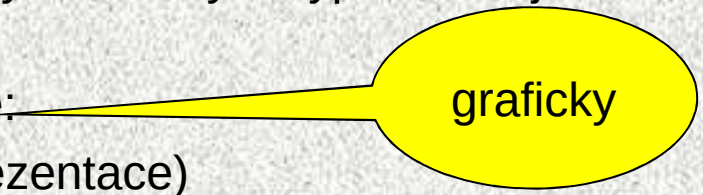
- Strukturované – procedurální
- Objektové

- V PR1 + PR2 byl zvolen postup od strukturovaného k objektovému
  - Těžiště PR1 je ve strukturovaném přístupu, PR2 v objektovém
    - Jazyk C nemá objektové možnosti
  - Princip strukturovaného přístupu:

Program = Data + Algoritmus

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 (+ způsob reprezentace)
  - 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)



Reprezentace dat v počítači

- V počítači není u každé datové položky určeno, jaký typ dat uchovává
- V jazyce Java musíme deklarovat s jakými údaji (typy dat) budeme pracovat!!
- Překladač jazyka Java tuto deklaraci hlídá!!

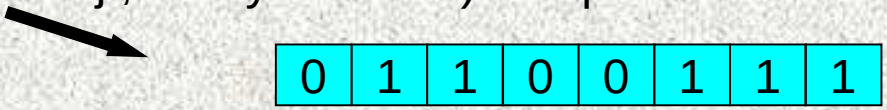
Příklad:

$(0100\ 0001)_2 = (41)_{16}$ nebo $(65)_{10}$ nebo znak A

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}

Pro info

Reprezentace celých čísel

- číselné soustavy - polyadické

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

- Desítková soustava

  $138{,}24 = \mathbf{1}*10^{\mathbf{2}}+\mathbf{3}*10^{\mathbf{1}}+\mathbf{8}*10^{\mathbf{0}}+\mathbf{2}*10^{\mathbf{-1}}+\mathbf{4}*10^{\mathbf{-2}}$
  $=1*100+3*10+8*1+2*0{,}1+4*0{,}01$

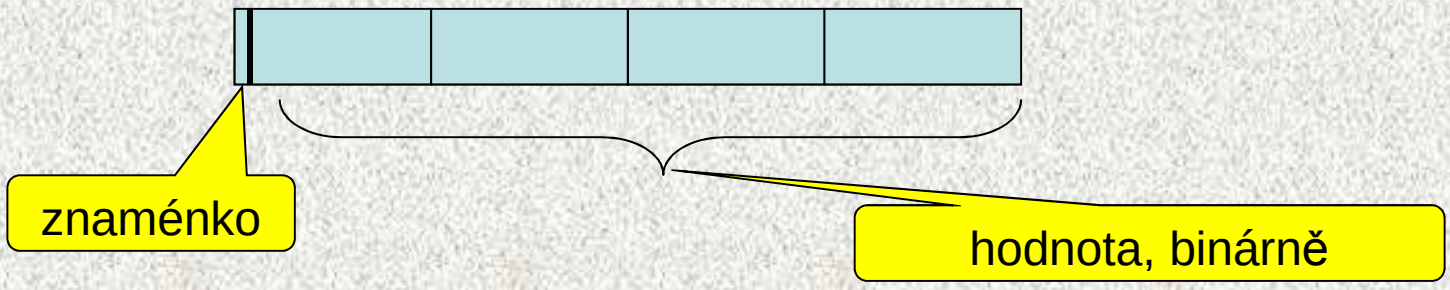
- Dvojková soustava

  $11010{,}01_b = \mathbf{1}*2^{\mathbf{4}}+\mathbf{1}*2^{\mathbf{3}}+\mathbf{0}*2^{\mathbf{2}}+\mathbf{1}*2^{\mathbf{1}}+\mathbf{0}*2^{\mathbf{0}}+\mathbf{0}*2^{\mathbf{-1}}+\mathbf{1}*2^{\mathbf{-2}} =$
  $1*16+1*8+0*4+1*2+0*1+0*1/2+1*1/4= 26{,}25$

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

Celá čísla v Javě – typ int

int je reprezentováno 32 bity



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

Pro zobrazování záporných čísel – doplňkový kód

nejmenší číslo v doplňkovém kódu *1000...000* = -2^{31} = -2147483648

Reprezentace záporných celých čísel

- Doplňkový kód – D(x)
  - předpokládejme reprezentaci čísel pomocí 8 bitů, lze reprezentovat $2^8 = 256$ čísel = r (rozsah)

$$D(x) = \begin{cases} x, \text{ pro } & 0 <= x < r/2 \\ r+x, \text{ pro } & 0 > x >= -r/2 \end{cases}$$

Pozor na důsledky počítání (v doplňkovém kódu) pro **`int`***:*
2 000 000 000 + 1 000 000 000 = -1 294 967 296

desítkově	**doplňkový kód**
0 – 127	*00000000 – 01111111*
128	nelze zobrazit na 8 bitů v d.k.
-128	D(-128) = -128+256 = 128 = *10000000*
-1	D(-1) = -1+256 = 255 = *11111111*
-4	D(-4) = -4+256 = 252 = *11111100*

Pro zájemce

Nepřesnost v zobrazení reálných 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?)

$0*3^1 + 0*3^0 + 1*3^{-1} = (0,1)_3$

Nepřesnost v zobrazení čísel

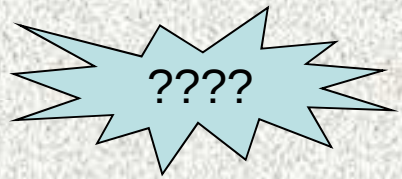
Nepřesnosti způsobují

1. Iracionální čísla (e , π, $\sqrt{2}$,….)
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

Implementace

`double` (na 64 bity)

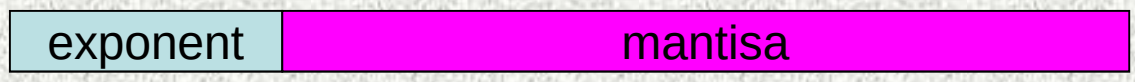
- 1bit znaménko – dvě možnosti, +,-
- 11 bitů exponent – 2048 možností
- 52 bitů – 4 503 599 627 370 496 možností, tedy asi 4,5 biliardy
  - není možné v typu double přesně uložit čísla se zápisem delším než 52 bitů!
  - čím větší exponent tím větší „mezery“ mezi sousedními aproximacemi!!!



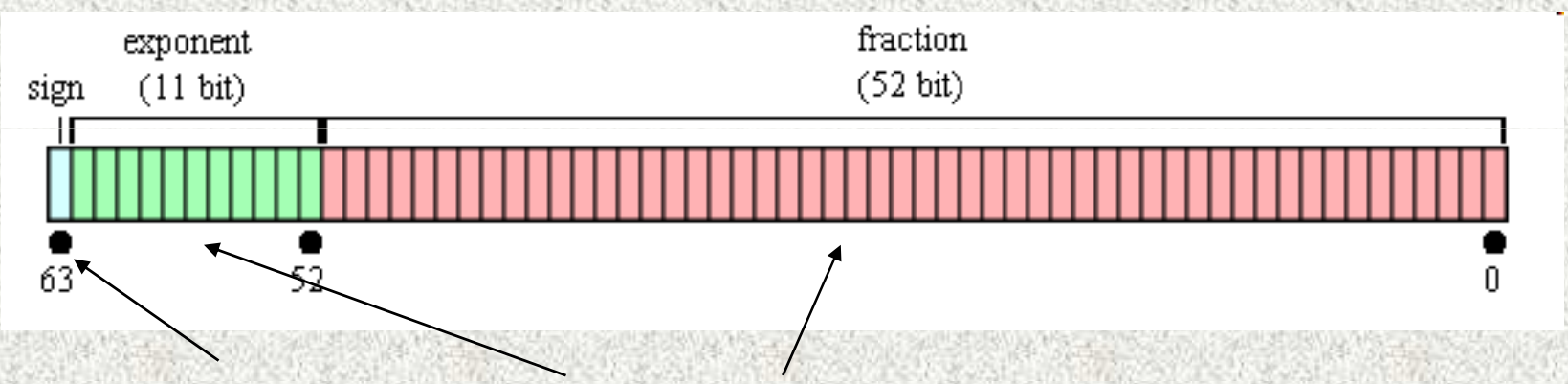
Necelá čísla v Javě – typ double

Reálná čísla X se zobrazují ve tvaru:

X= mantisa*základ $^{\text{exponent}}$



double je reprezentován 64 bity, norma IEEE 754



znaménkový bit (s), exponent, mantisa

Nejmenší čísla v normalizovaném tvaru

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

Největší čísla

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

Model reprezentace reálných čísel

Reálná čísla se zobrazují jako aproximace daných rozsahem paměťového místa

Reálná čísla se zobrazují ve tvaru:

X= mantisa*základ $^{\text{exponent}}$ = m*z $^{\text{exponent}}$

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}

+	775	+	02
1	775	1	02

- zakódujeme znaménko "+" = 0," –" = 1
- z je pro názornost 10

Model reprezentace reálných čísel

Maximální zobrazitelné kladné číslo

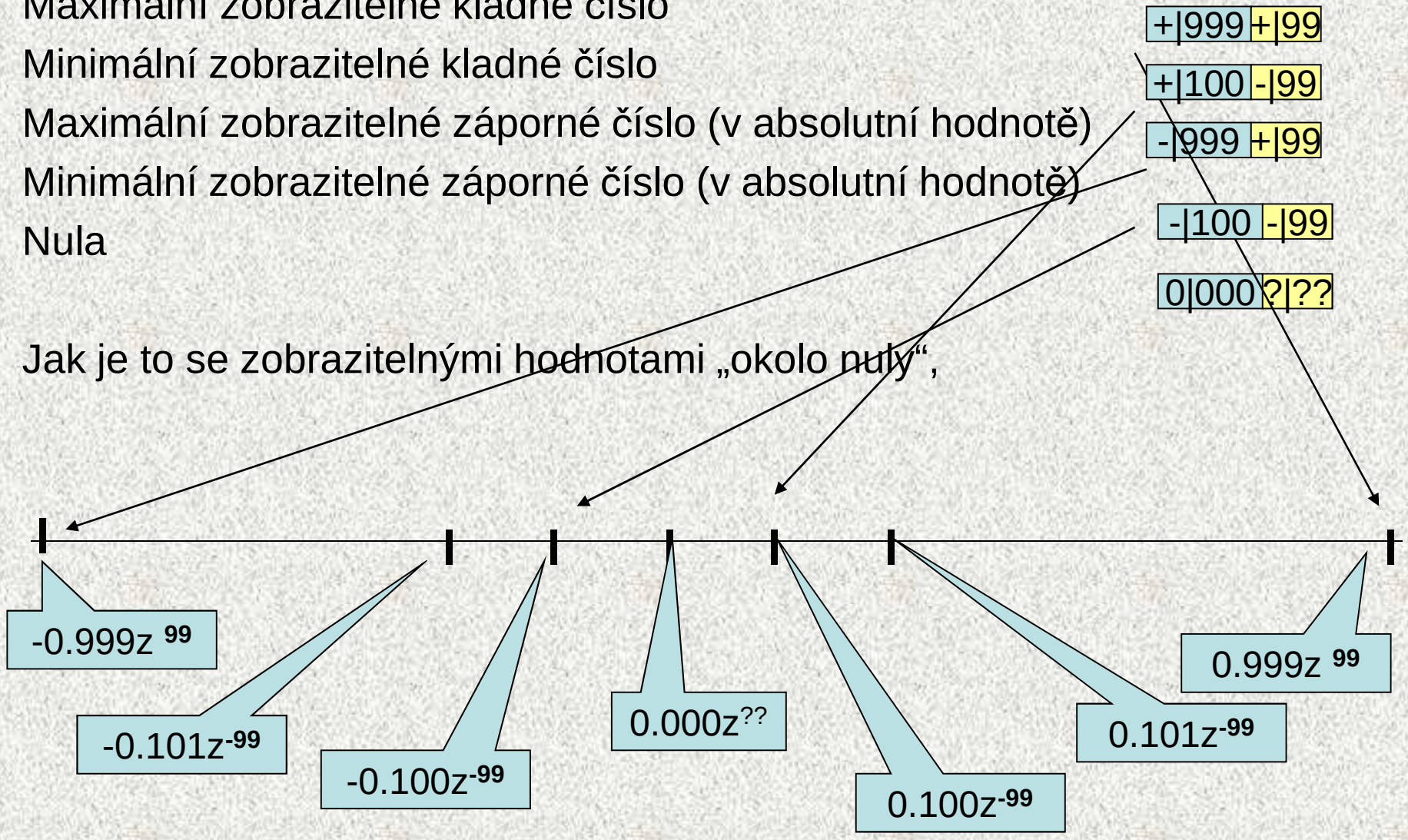
Minimální zobrazitelné kladné číslo

Maximální zobrazitelné záporné číslo (v absolutní hodnotě)

Minimální zobrazitelné záporné číslo (v absolutní hodnotě)

Nula

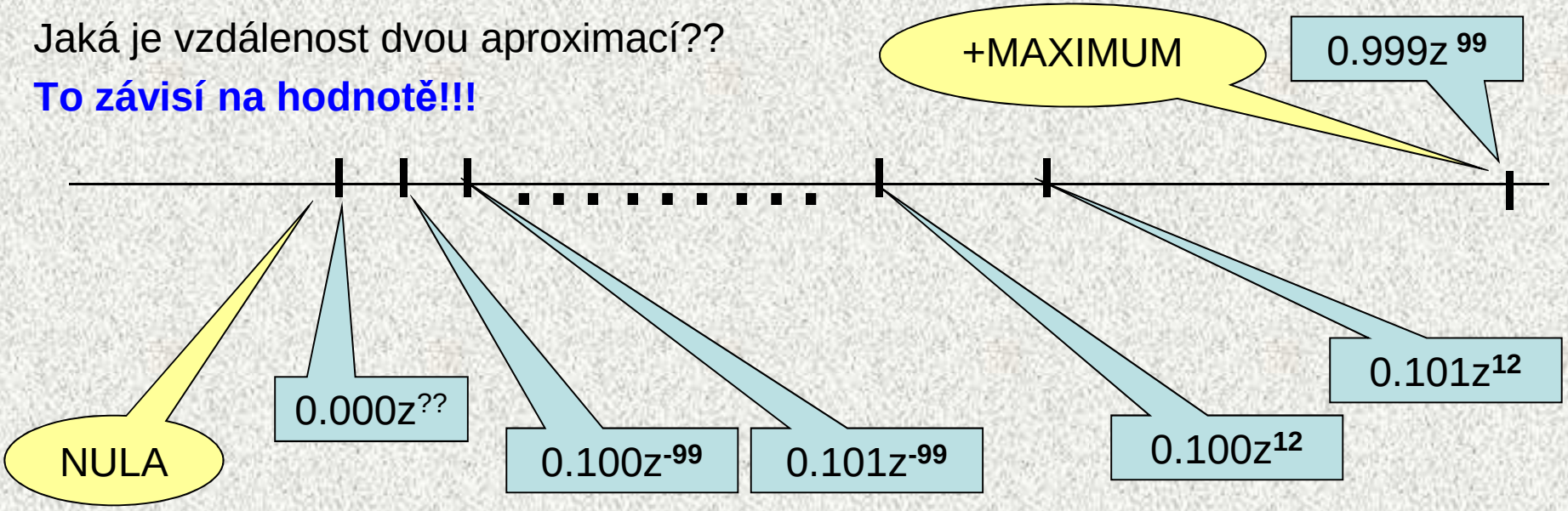
Jak je to se zobrazitelnými hodnotami „okolo nuly“,



Model reprezentace reálných čísel

Jaká je vzdálenost dvou aproximací??

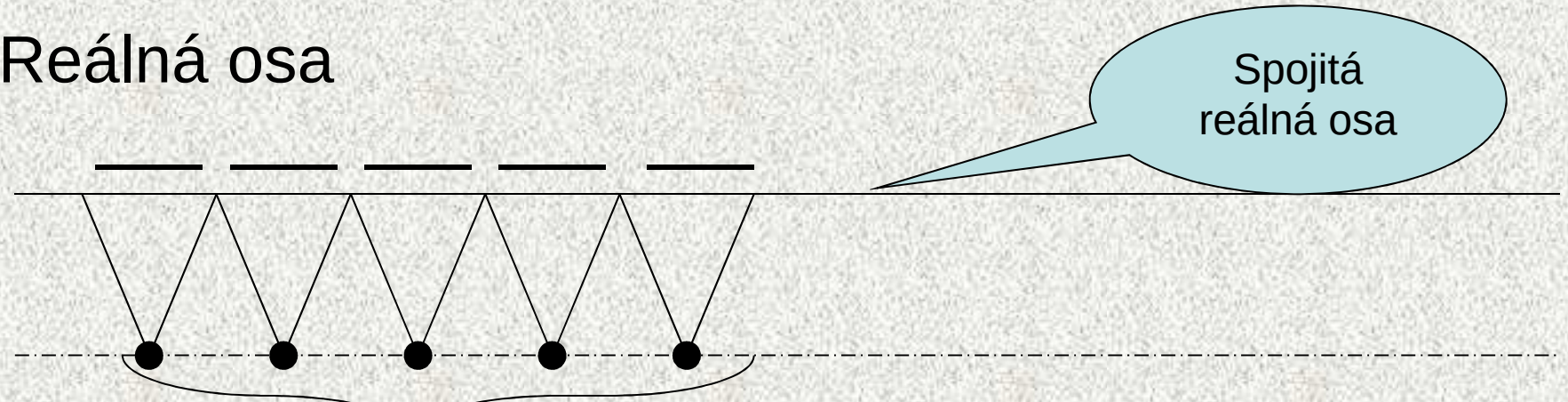
To závisí na hodnotě!!!



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

Reprezentace reálných čísel

Reálná osa



Zobrazení aproximací (pro jeden exponent!!)

Reálná čísla se zobrazují jako aproximace daných rozsahem paměťového místa

Diskrétní
aproximace v
paměti počítače

Reprezentace znaků

Java používá typ **char**, znaky jsou kódovány 16 bitovým kódem Unicode (UTF-16, podmnožina UCS-2).

- Odstraňuje problémy češtiny
- Programy v Javě provádí konverzi znaků mezi vnitřní reprezentaci v programech a OS, většinou automaticky podle informací OS (Locale)

- Poznámky
  - Různé OS různá kódování
    - sedmibitové ASCII (American Standard Code for Information Interchange), osmibitová Win1250, ISO 8859-2, kód s proměnnou délkou znaku - UTF8(UCS Transformation Format), .
  - Java – UCS2
    - Universal Character Set - univerzální znaková sada definována normou ISO 10646

Reprezentace znaků, část reprezentace

Dec	Hex	Znak	Význam
0	00	NUL	
1	01	☺	Start of Header (SOH)
2	02	☻	Start of Text (STX)
3	03	♥	End of Text (ETX)
4	04	♦	End of Transmission (EOT)
5	05	♣	Enquiry (ENQ)
6	06	♠	Acknowledge (ACK)
7	07	•	Bell (BEL)
8	08	◘	Backspace (BS)
9	09	○	Horizontal Tab (HT)
10	0a	◙	Line Feed (LF)
11	0b	VT	Vertical Tab
12	0c	FF	Form Feed
13	0d	CR	Carriage Return
14	0e	SO	Shift Out

Dec	Hex	Znak
32	20	SPC
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)
42	2a	*
43	2b	+
44	2c	,
45	2d	-
46	2e	.

Dec	Hex	Znak
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4a	J
75	4b	K
76	4c	L
77	4d	M
78	4e	N

Dec	Hex	Znak
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6a	j
107	6b	k
108	6c	l
109	6d	m
110	6e	n

Primitivní či základní datové typy

Typ	Bitů	Rozsah		Obal.třída
Celočíselný typ				
byte	8	-128 … 127		Byte
short	16	-32768 … 32767		Short
int	32	-2147483648 … 2147483647		Integer
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				

Výrazy

- Výraz předepisuje výpočet hodnoty určitého typu
- Příklad výrazu: {14.6 + sin(3.14)} * a_{100}
- Výraz může obsahovat:
  - proměnné
  - konstanty
  - binární operátory
  - volání funkcí
  - 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

Vyhodnocení výrazů, příklady

124+4*8 = 124 + (4*8) = 124 + 32 = 156
// násobení má vyšší prioritu

36/2*9 = (36/2)*9 = 18*9 = 162
// násobení i dělení mají stejnou prioritu, asociativita je L - zleva

Asociativita a priorita operací

- Binární operace na množině *S* **asociativní**, jestliže platí
  - (*x* * *y*) * *z* = *x* * (y * z), pro každé *x*, *y* a *z* v *S*.
- U neasociativních operací je tedy třeba buď důsledně závorkovat, nebo se dohodnout na implicitním pořadí provádění operací – pak se někdy mluví o operacích *asociativních zleva* či *asociativních zprava*.
- **Priorita binárních operací** vyjadřuje pořadí, v jakém se provádějí binární operace
- Příklady:
  - Odčítání levě asociativní,
    - výraz 10 − 5 − 3 se chápe jako (10 − 5) − 3,
  - Umocňování je asociativní zprava $2^{3^4} = 2^{(3^4)}$

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
```

Operátory a jejich priorita

priorita	operátor	typ operandu	asociativita	operace
1	++	aritmetický	P	pre/post inkremetace
	--	aritmetický	P	pre/post dekrementace
	-	aritmetický	P	unarní plus/minus
	~	celočíselný	P	bitová inverze
	!	logický	P	logická negace
	(typ)	libovolný	P	přetypování
2	*, / ,%	aritmetický	L	násobení, dělení, zbytek
3	-	aritmetický	L	odečítání
	+	aritmetický, řetězový	L	sčítání, zřetězení

Operátory a jejich priorita II

priorita	operátor	typ operandu	asociativita	operace
4	**<<**	celočíselný	L	posun vlevo
	>>	celočíselný	L	posun vpravo
	>>>	celočíselný	L	posun vpravo s doplňováním nuly
	<,>, <=, >=	aritmetický	L	porovnání
	instanceof	objekt	L	test třídy
	==, !=	primitivní	L	rovno, nerovno
7	**&**	celočíselný nebo logický	L	bitové nebo logické AND
8	**^**	-"-	L	bitové nebo logické XOR
9	**|**	-"-	L	bitové nebo logické OR

Operátory a jejich priorita III

priorita	operátor	typ operandu	asociativita	operace
10	**&&**	logický	L	logické AND vyhodnocované zkráceně
11	**||**	logický	L	logické OR vyhodnocované zkráceně
12	**? :**	logický	P	podmíněný operátor
13	**=, -=, *=, /=, %=, ==, &=, ^=, |=**	libovolný	P	přiřazení

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;
```

Deklarace proměnných

- každá proměnná má definovaný **typ**
- Java zná 8 primitivních datových typů
  - ostatní typy jsou referenční – objekty, pole, řetězce,..)

Příklad

```
int a;
```

- definuje proměnnou typu int
- lze do ní přiřadit pouze hodnoty typu int a hodnoty užší(tedy hodnoty, které se dají rozšířit na int, tedy byte, short a char
- lze s ní provádět operace definované pro int

Jak spolu souvisí typ int a typ „celá čísla“??

Deklarace proměnných

Proměnné se zavádějí deklaracemi

```
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

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

- 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;
```

Přiřazovací příkaz

```
int a, b = 10;
```

- hodnota a není definována, hodnota b je 10

```
a = b;
```

- a je 10, b je 10

```
b = 12 + 5;
```

- a je 10, b je 17

```
a = a+3;
```

- vezmi hodnotu z a (10), zvětši ji o 3 a výsledek ulož zpět do a, a je 13
- Pozor, proč lze psát:

```
y = x = x + 6;
```

- Přířazovací příkaz je výrazem - vyhodnotí se jako

```
y = (x = (x + 6));
```

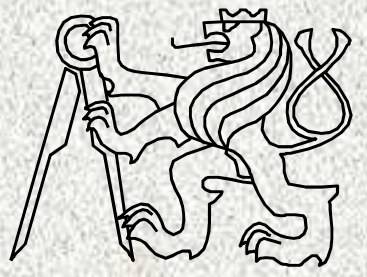
Jména a konvence

- **balíček - `package` -** jen malá písmena, i několik jmen oddělených tečkou, např: **`prog1, java.util`**
- **třída, abstraktní třída, jejich konstruktory, rozhraní - `class, abstract class, interface`**
  - jméno začínající velkým písmenem, např: **`String, MyFirstClass, Serializable, Comparable`**
  - Výjimky by měly mít sufix Exception, např: **`MySpecialException`**
- **metoda** - sloveso začínající malým písmenem, další slovo začíná velkým písmenem, např: **`setBorder, isEmpty, getNumber`**
- **proměnné -** začínající vždy malým písmenem další slovo začíná velkým písmenem, např: **`diskriminant, totalCount`**
- **konstanta -** jen velká písmena, jednotlivá slova oddělena podtržítkem, např. **`MAX_COUNT, RED`**

Ukázka vývojového prostředí

toto je nejlepší ukázat v NetBeans .....

Java – reprezentace dat
konec



A0B36PR1-Programování 1

Fakulta elektrotechnická
České vysoké učení technické