Primitivní typy, operace, výrazy

- Primitivní datové typy
- Operace, operátory
- Operace, vyhodnocení, priorita
- Výrazy, konstrukce, vyhodnocení
- Obalové třídy primitivních datových typů

Typy jazyka Java

- Základní datové typy (primitivní datové typy)
  - Celočíselné – `byte, short, int, long`
  - Reálné – `float, double`
  - Znakový – `char`
  - Logický – `boolean`

  - Základní datové typy nejsou třídy
  - Ke každému základnímu primitivnímu datovému typu existuje obalová třída, pomocí které lze hodnotu toho kterého typu převést na objekt a která obsahuje některé nástroje pro práci s hodnotami daného typu

- Referenční datové typy
  - Pole
  - Třídy – `class`
  - Výčet – `enum`
  - Rozhraní – `interface`

PRIMITIVNÍ DATOVÉ TYPY

Reálné typy

Typy

`float`	4 byte	±1.4E-45 .. ±3.4E+38	7 – 8 platných číslic
`double`	8 byte	±4.9E-324 .. ±1.7E+308	15 – 16 platných číslic

- Hodnoty reálných typů ukládány v paměti v unifikovaném formátu

  `<znaménko> <mantisa> . 10`$^{\texttt{<exponent>}}$

- Typy vyhovují mezinárodnímu standardu IEEE 754
  - typ `float` 4 byte = 32 bitů, 1 bit znaménko, 8 bitů exponent, 23 bitů mantisa – $\pm\ 1.mmm \times 2^{eee}$
- Reálná čísla – **nepřesné zobrazení**
- Zápis konstantních hodnot reálných typů
  - Znaky – znaménko, číslice, oddělovač des. míst, oddělovač exponentu
  - Oddělovač desetinných míst tečka `12.3` `-1.847`
  - Mohou začínat a končit tečkou `14.` `.85`
  - Oddělovač exponentu – `E, e` `8e2` `5E-3` `-4.846E+3`
  - Implicitně jsou konstanty typu `double`
  - Konstanty typu `float` končí na `F, f` `-14.3f` `1E-5F`

Celočíselné typy

Typy

`byte`	1B	-128 .. 127
`short`	2B	-32768 .. 32767
`int`	4B	-2147483648 .. 2147483647
`long`	8B	(viz například specifikace jazyka)
	řád hodnot 10^{18}	

- Hodnoty uloženy ve tvaru dvojkového doplňku (viz předmět CIP)
- Jazyk Java nemá neznaménkový celočíselný typ – jistá nevýhoda u jednobytových hodnot – technické aplikace, grafické aplikace
- Celá čísla (v daném rozsahu) – **přesné zobrazení**

Celočíselné konstanty

- V **desítkové** soustavě
  - Posloupnost číslic `0..9`, které nezačíná číslicí `0` (nula)

    `94, 76, 0, 1`
- V **osmičkové** soustavě
  - Posloupnost číslic `0..7`, která začíná číslicí `0` (nula)

    `07, 013`
- V **šestnáctkové** soustavě
  - Posloupnost, která začíná znaky `0x` nebo `0X` následovanými hexadecimálními číslicemi `0..9, a,b,c,d,e,f` nebo `A,B,C,D,E,F`

    `0xFF, 0X10`

Celočíselné konstanty jsou implicitně typu `int`

Konstanta typu `long` – ukončená znakem `L, l`

`9876543210123456L`

Logický typ

- Jediným logickým typem jazyka Java je typ **`boolean`**

- Typ `boolean` je typ, který obsahuje pouze 2 hodnoty
- Tyto hodnoty jsou představované (a v kódu zapisované) logickými konstantami `true` a `false`
  - `true` – logická 1, pravda
  - `false` – logická 0, nepravda

- V jazyce Java nejsou hodnoty logického typu převoditelné na hodnoty jiného datového typu

Znakový typ

- Typ **`char`**
- Proměnná typu `char` zabírá v paměti 2 byte
- Znaky jsou kódovány čísly – jazyk Java používá kódování Unicode

- Znakové konstanty – uzavřené do apostrofů
  - Jediný **znak v apostrofech** 'A'
  - **Posloupnost** '\uXXXX', kde XXXX jsou šestnáctkové číslice, představující kód znaku, použití například pro zápis akcentovaných znaků '\u0041'
  - **Escape sekvence** například '\n' pro znak „nová řádka“ (newline, linefeed, LF)
  - **Osmičkový zápis** '\ooo', kde ooo jsou osmičkové číslice představující kód čísla, vždy jsou nutné všechny tři číslice '\101'

- Typ `char` je kompatibilní s typem `int` (konverze z `char` na `int` je implicitní)

- Escape sekvence

\b	\u0008	backspace BS
\t	\u0009	horizontal tab HT
\n	\u000a	linefeed LF
\f	\u000c	form feed FF
\r	\u000d	carriage return CR
\"	\u0022	double quote "
\'	\u0027	single quote '
\\	\u005c	backslash \

!!! Konstantní znakové hodnoty je nutné uzavřít v apostrofech

- Kódy některých akcentovaných znaků

Á	\uOOC1	á	\u00E1
Č	\u010C	č	\u010D
Ď	\u010E	ď	\u010F
É	\u00C9	é	\u00E9
Ě	\u011A	ě	\u011B
Í	\u00CD	í	\u00ED
Ň	\u0147	ň	\u0148
Ó	\u00D3	ó	\u00F3
Ř	\u0158	ř	\u0159
Š	\u0160	š	\u0161
Ť	\u0164	ť	\u0165
Ú	\u00DA	ú	\u00FA
Ů	\u016E	ů	\u016F
Ý	\u00DD	ý	\u00FD
Ž	\u017D	ž	\u017E

Řetězcové konstanty

- Konstantní textové řetězce – jedná se o instance třídy `String` (nepatří mezi primitivní datové typy)

- Sekvence konstantních znaků uzavřené do uvozovek
- V jednom jediném textovém řetězci je možné kombinovat různé zápisy znaků

```
"konstantni textovy retezec"
"Zadej stranu \u010Dtverce "
"a\nb\nc\n"
```

- Java umožňuje zřetězení textových řetězců pomocí operátoru +

```
"kratky text " + "kratky text je dlouhy text "

"Hodnota cisla pi je " + 3.14
"Mocnina dvou je " + 4
"Logicka 1 je " + true
```

Konverze mezi typy

- Java – jazyk se silnou typovou kontrolou

- Jednotlivé proměnné, konstanty, metody – musí mí specifikovaný typ – typ je určený při deklaraci (s typem souvisí paměťové místo – počet byte, formát uložení hodnoty do přiděleného paměťového místa, množina přípustných operací)

- Konverze mezi typy (hodnotami různých typů)
  - Implicitní konverze – rozšiřující konverze
  - Vynucená konverze – zužující konverze
  - Nepřípustné konverze

  - **Rozšiřující konverze** jazyk Java provádí implicitně, není nutné explicitní přetypování, při těchto konverzích nedochází ke ztrátě informace (poznámka: ke ztrátě informace může dojít při konverzi `long` na `double` nebo `int`, `long` na `float`)

    `byte` → `short` → `int` → `long` → `float` → `double`

    `char` → `int`

  - **Zužující konverze** se neprovádějí implicitně, pokud takovou konverzi potřebujeme je nutné využít operátor přetypování, může dojít ke ztrátě informace nebo změně hodnoty

    `double` → `float` → `long` → `int` → `short` → `byte`

    `int` → `char`

OPERACE, VÝRAZY

Operace, operátory

- **Unární** – jeden operand, operátor se zapisuje ve většině případů před operand, v některých případech následuje operátor až za operandem

  `<operator><operand>` nebo `<operand><operator>`

- **Binární** – dva operandy, jazyk Java používá **infixový** zápis, operátor je mezi oběma operandy, jiné možné způsoby jsou prefixový (operátor je zapsán před operandy, prefixový způsob zápisu používá např. jazyk Lisp) a sufixový respektive postfixový (operátor je zapsán za operandy)

  `<operand1><operator><operand2>`

- **Ternární** – má tři operandy, v jazyce Java je ternární logický operátor

  `<operand1><operator1c><operand2><operator2c><operand3>`

- Výsledkem operace je hodnota – typ výsledné hodnoty závisí obecně na příslušné operaci a na operandech (na typu operandů)
- Konstrukce výrazů – výraz po vyhodnocení má hodnotu
- Výraz ukončený středníkem je příkazem

Operátory přiřazení

- Operátor přiřazení je v jazyce Java reprezentován znakem **=** operaci přiřazení zapisujeme

  <proměnná> = <výraz>

  - <proměnná> první operand musí být identifikátor proměnné
  - <výraz> – konstantní hodnota, proměnná, nebo výraz, hodnota výrazu na pravé straně přiřazení musí být **typově kompatibilní** vzhledem k přiřazení s typem proměnné, která je na levé straně operátoru přiřazení

- Operace přiřazení se vyhodnocuje **zprava doleva**
- Operace přiřazení má výslednou hodnotu rovnou přiřazené hodnotě
- **Příkaz přiřazení** – ukončení operace středníkem

- Příklady příkazů přiřazení (obsahují operaci přiřazení ukončenou středníkem)

```
int a, b, c;
a = 4;
b = 2 * a;
c = b = a;
```

```
int a, b, c;
a = 12;
b = 20;
c = a; a = b; b = c;
```

- Další operátory přiřazení
  - **+=**
  - **-=**
  - ***=**
  - **/=**
  - **%=**
  - **&=**
  - **^=**
  - **|=**
  - **>>=**
  - **<<=**
  - **>>>=**

```
<prom> += <výraz>
// má význam jako
<prom> = <prom> + <výraz>
```

Obdobně ostatní operátory přiřazení

Aritmetické operátory

Binární aritmetické operátory

- Použitelné na operandy číselných typů
  - **+** součet
  - **-** rozdíl
  - ***** násobení
  - **/** dělení reálné, dělení celočíselné
  - **%** zbytek po celočíselném dělení, dělení modulo
- Operandy jsou v obecném případě výrazy s hodnotou číselného typu, výsledná hodnota je dána odpovídající aritmetickou operací
- Poznámky
  - Typ výsledné hodnoty operací `+, -, *, /` **závisí na typu operandů**, pokud je jeden z operandů reálný, potom je i výsledek reálný, v opačném případě je výsledek celočíselný
  - Při dělení nulou v oboru celých čísel chyba
  - Dělení nulou v oboru reálných čísel – přípustná operace (NaN)
  - Aritmetika reálných čísel – nepřesná – plyne z nepřesného zobrazení reálných čísel v paměti počítače

- Příklady

```
float b = 3.2f;
float a = 25;
float c, d;
c = a – b;
d = a / b;

int m = 25 / 6;
int n = 25 % 6;

d = 31 / 7;

d = a / 6;
```

Aritmetické operátory

Unární aritmetické operátory

- Použitelné na operandy číselných typů
  - **+** unární plus
  - **-** unární mínus
  - **++** inkrementace hodnoty operandu, prefixový i postfixový zápis
  - **--** dekrementace hodnoty operandu, prefixový i postfixový zápis
    - Operandem inkrementace, dekrementace může být pouze proměnná – některého z číselných nebo znakového typu
    - Prefixový a infixový zápis má smysl rozlišovat ve složitějších výrazech
      - Prefixový zápis – hodnota proměnné je inkrementována/dekrementována, poté použita ve výrazu
      - Postfixový zápis – hodnota proměnné je použita ve výraze, poté je inkrementována/dekrementována
    - Výsledkem operace je hodnota, vedlejším efektem přiřazení
    - `++i;  i = i + 1;`
    - `--i;  i = i – 1;`

- Příklady

```
int a = +5;
int min = -164;
char c = 'a';
c++;
```

```
int k = 7;
++k;
int l = k++;
l = ++k;
```

- Co bude na výstupu?

```
int n = 1;
System.out.println(n);
System.out.println(++n);
System.out.println(n++);
System.out.println(n);
System.out.println(++n + n);
System.out.println(n++ + n);
System.out.println(++n + ++n);
System.out.println(++n + n++);
System.out.println(n++ + ++n);
System.out.println(n);
System.out.println(-++n);
System.out.println(---n);
```

Relační operátory

- Relační operátory jsou binární operátory, slouží k porovnání hodnot operandů
- Konstrukce logických výrazů
- Operandy mohou být libovolného typu, oba operandy musí být typově kompatibilní
- Výsledkem operace je hodnota logického typu (`true` nebo `false`)
- Operátory
  - **==** porovnání rovnosti
  - **!=** nerovnost
  - **<** menší
  - **<=** menší nebo rovno
  - **>** větší
  - **>=** větší nebo rovno

- Pozor na porovnávání ostré rovnosti, nerovnosti reálných čísel, porovnáváme „přibližnou“ rovnost

```
4 == 5  // false
4 != 5  // true
```

```
4 < 5   // true
4 >= 5  // false
```

Logické operátory

- Z logických operátorů je jediný operátor unární ostatní jsou binární
- Operandy jsou logického typu, výsledek logického typu (`true`, `false`)
- Konstrukce logických výrazů
  - **!** negace, NOT, unární operátor, zapisuje se před operand
  - **&&** logický součin AND se zkráceným vyhodnocováním
  - **||** logický součet OR se zkráceným vyhodnocováním
  - **&** logický součin AND s úplným vyhodnocováním
  - **|** logický součet OR s úplným vyhodnocováním
  - **^** logický výlučný součet XOR

- Operátory `&, | , ^` se používají i jako bitové operátory
- Pravdivostní tabulka jednotlivých funkcí – viz dále
- Příklady

```
(a > 3) && (a < 10)
(x > -3.4)  && (x <= 11.2)
c >= ‘0' && c <= '9'
c >= 'A' && c<= 'Z' || c >='a' && c<= 'z‘
((c >= 'A‘) && (c<= 'Z‘)) || ((c >='a‘) && (c<= 'z‘))
b = !b
```

Logické operátory

- Pravdivostní tabulka logických funkcí

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

- **Úplné a neúplné vyhodnocování** logických výrazů

  `(x > 4) & (x < 8) & (y > 3) & (y < 6)` – v případě, že není splněna první část této podmínky (pro `x <= 4`) má celý výraz hodnotu `false`, není nutno vyhodnocovat další tři základní podmínky výrazu

  `(x < 6) | (y > 12)` – v případě, že první část podmínky (`x < 6`) má hodnotu `true`, celý výraz má hodnotu `true` bez ohledu na hodnotu druhé části výrazu

  - Jazyk Java má k dispozici dva druhy logických operátorů pro operace logického součtu a logického součinu, které zajistí buď úplné nebo neúplné vyhodnocování obdobných logických výrazů

Bitové operace

- Při bitových operacích se provádějí operace s jednotlivými bity operandů
- Operandy nesmějí být typu `float` nebo `double`
  - **&** bitový součin AND
  - **|** bitový součet OR
  - **^** bitový exkluzivní součet XOR
  - **<<** bitový posun vlevo
  - **>>** bitový posun vpravo
  - **>>>** neznaménkový posun vpravo
  - **~** jedničkový doplněk, unární operátor

Přetypování

- Přetypování využijeme máme-li hodnotu nějakého typu a potřebujeme z ní „vytvořit“ hodnotu jiného typu (například z `float` na `int`)

  `(<typ>) <výraz>`

- Výsledkem je hodnota výrazu `<výraz>` reprezentovaná jako hodnota typu `<typ>`

- Přetypování má nejvyšší prioritu. Pokud chceme přetypovat výslednou hodnotu výrazu, musíme výraz vložit do závorek.

```
double d = 15.76E-3;
float f = (double) d;
float x = (double) (2 * f);
```

- **Rozšiřující konverze** jazyk Java provádí implicitně, není nutné expl. přetypování

  `byte` → `short` → `int` → `long` → `float` → `double`

  `char` → `int`

  Při těchto konverzích nedochází ke ztrátě informace (poznámka: ke ztrátě informace může dojít při konverzi `long` na `double` nebo `int`, `long` na `float`)

- **Zužující konverze** se neprovádějí implicitně, pokud takovou konverzi potřebujeme je nutné využít operátor přetypování

  `double` → `float` → `long` → `int` → `short` → `byte`

  `int` → `char`

  Při těchto konverzích může dojít ke ztrátě informace nebo změně hodnoty

```
float f= 13.74f;
System.out.println((int)f);    //13
int i = 129;
System.out.println((byte) i);  //-127
```

Konstrukce výrazů

- Výraz – série operací
- Obsahuje konstantní hodnoty, proměnné, operátory, volání metod
- Pro výpočet matematických funkcí ve výrazech voláme (používáme) metody třídy `Math` (popřípadě `StrictMath`)
  Třída `Math` je umístěna v baličku `java.lang`
  Např. pro výpočet odmocniny ze 2 použijeme: `Math.sqrt(2)`
- Úplný výčet metod a jejich popis – viz dokumentace jazyka Java

- Některé metody třídy `Math, StrictMath`

`random` – generování náhodného čísla v rozsahu 0 – 1
`ceil` – celá část reálného čísla, která je menší než parametr (parametr i výsledek `double`)
`floor` – celá část reálného čísla, která je větší než parametr
`rint` – celé číslo nejbližší předanému argumentu
`round` – zaokrouhlená hodnota reálného čísla – výsledek je typu `long`
`signum` – vrací 0.0, -1.0, 1.0 dle znaménka argumentu
`getExponent` – vrací exponent reálného čísla

`E` – konstanta typu `double`, základ přirozeného logaritmu
`PI` – konstanta typu `double`, Ludolfovo číslo
`abs` – absolutní hodnota čísla
`max` – větší ze dvou hodnot
`min` – menší ze dvou hodnot
`pow` – obecná mocnina čísla
`sqrt` – druhá odmocnina čísla
`exp` – exponenciální funkce
`log` – logaritmická funkce, přirozený logaritmus (základ e)
`log10` – dekadický logaritmus
`sin` – goniometrická funkce sin – parametr v radiánech
`cos` – goniometrická funkce cos
`tan` – goniometrická funkce tg
`asin` – inverzní funkce k funkci sin – výsledek v radiánech
`acos`
`atan`
`sinh` – hyperbolický sinus
`cosh`
`tanh`
`toDegrees` – převod radiánů na stupně
`toRadians` – převod stupňů na radiány

Použití metod třídy `Math`

- Metody vrací výslednou hodnotu deklarovaného typu svým jménem – volání metod použijeme obecně ve výrazech
- Každá z uvedených metod (kromě metody `random`) má parametry – pořadí a typ parametrů musí odpovídat deklaraci – podrobnější popis v dokumentaci jazyka Java
- Některé z uvedených metod jsou k dispozici ve více verzích – například metoda `abs` – tyto verze se liší typem argumentu, typem vracené hodnoty – jedná se o tzv. *přetížené metody* – při volání je potom podle typu skutečného argumentu rozhodnuto, která verze metody bude použita
- Bližší popis metod v **dokumentaci jazyka Java**

```
float r, obvod;
r = (float) Math.random()*100; // náhodné reálné číslo z intervalu <0, 100)
obvod = (float) (2 * Math.PI * r);

int n = 10 + (int) (Math.random() * 90); // náhodné celé dvoumístné číslo
                                            tj. číslo od 10 do 99
double alfa = 60;              // úhel ve stupních
double sinAlfa = Math.sin(Math.toRadians(alfa));
sinAlfa = Math.sin(alfa * Math.PI/180);
```

Vyhodnocení výrazů

- Operace přiřazení jsou vyhodnocovány zprava doleva

  `a = b = c = d`

- Ostatní operace jsou vyhodnocovány zleva doprava

  `a + b + c + d`

- Operace ve výraze jsou vyhodnocovány s ohledem na prioritu – operace s vyšší prioritou je vyhodnocena dříve

  `a + b * c + d`

- Chceme-li změnit implicitní pořadí vyhodnocování použijeme okrouhlé závorky
- Okrouhlé závorky musí být párové a mohou být libovolně vnořené

Priorita operátorů

Uvedeno **od nejvyšší priority k nejnižší**

`(typ)  []  .`	přetypování, indexování, reference
`!  ++  --` `-  +  ~`	unární operátory
`*  /  %`	multiplikativní – násobení a dělení
`+  -`	aditivní – sčítání a odčítání
`<<  >>  >>>`	bitový posun
`<<  >>  >>>`	bitový posun
`<  >  <=  >=`	relační operátory
`==  !=`	relační operátory rovno, nerovno
`&`	logický i bitový součin AND
`^`	logický i bitový XOR
`|`	logický i bitový součet OR
`&&`	logický zkrácený součin
`||`	logický zkrácený součet
`?:`	logický ternární operátor
`=  +=  -=  *=`	operátory přiřazení
`,`	operátor čárka v příkazu `for`

OBALOVÉ TŘÍDY PRIMITIVNÍCH TYPŮ

Obalové třídy primitivních datových typů

- Java objektový jazyk
- Primitivní datové typy – `float`, `double`, `byte`, `short`, `int`, `long`, `char`, `boolean` – nejedná se o třídy, proměnné primitivních datových typů nejsou objekty
- Ke každému primitivnímu datovému typu má jazyk Java *obalovou třídu* (*wrapper class*, obalující/obalovou třída)

Typ	Obalová třída
`float`	**`Float`**
`double`	**`Double`**
`byte`	**`Byte`**
`short`	**`Short`**
`int`	**`Integer`**
`long`	**`Long`**
`char`	**`Character`**
`boolean`	**`Boolean`**

- *Obalové třídy*
  - Umožňují konverzi hodnoty příslušného primitivního typu na objekt,
  - Obsahují předdefinované konstanty příslušného primitivního typu,
  - Obsahují metody pro manipulaci s hodnotami příslušného typu
  - Úplný přehled konstant, metod, konstruktorů jednotlivých tříd – viz **dokumentace jazyka Java**

Třídy `Float` a `Double`

Vybrané konstanty

`MIN_VALUE`
`MAX_VALUE`
`MIN_EXPONENT`
`MAX_EXPONENT`
`NEGATIVE_INFINITY` – hodnota, která je výsledkem dělení záporného čísla nulou
`POSITIVE_INFINITY`
`NaN` – tato hodnota je výsledkem operace dělení nuly nulou
`SIZE` – velikost hodnoty typu v bitech

Použití

```
float min = Float.MAX_VALUE;
float max = -Float.MAX_VALUE;
```

Vybrané statické metody třídy `Double` (Třída `Float` má metody obdobné)

`compare(d1, d2)` – výsledek typu `int`
`isInfinite(d)` – výsledek typu `boolean`
`isNaN(d)` – výsledek typu `boolean`

`parseDouble(s)` – výsledek typu `double`
`toString(d)` – výsledek typu `String`

`valueOf(s)` – výsledek typu `Double`
`valueOf(d)` – výsledek typu `Double`

Použití

```
v = sum / n;
if (Float.isInfinite(v)) …;
else …;
```

Třídy `Byte`, `Short`, `Integer` a `Long`

Vybrané konstanty

`MIN_VALUE`
`MAX_VALUE`

`SIZE` – velikost hodnoty typu v bitech

Použití

```
int max = Integer.MIN_VALUE;
int min = Integer.MAX_VALUE;
```

Některé statické metody třídy `Integer`

`parseInt(s)` – výsledek typu `int`
`toString(i)` – výsledek typu `String`

`valueOf(s)` – výsledek typu obalové třídy
`valueOf(i)` – výsledek typu obalové třídy

Použití

```
int n = Integer.parseInt("126");
```

Třída `Character`

Vybrané konstanty

- Třída Character obsahuje řadu konstat

`MIN_VALUE`
`MAX_VALUE`
`SIZE` – velikost hodnoty typu v bitech

- Použití konstant a metod
  - před jménem statické metody třídy nebo konstanty musí být uveden identifikátor odpovídající třídy oddělený od identifikátoru metody resp. konstanty tečkou
  - před jménem instanční metody (konstanty, instanční proměnné) musí být uveden identifikátor proměnné odpovídajícího objektového typu oddělený od identifikátoru metody tečkou

Některé metody

`toString(c)` – výsledek typu `String`
`valueOf(c)` – výsledek typu obalové třídy

Testování znaků
`isLetter(c)`
`isLetter(i)`
`isLowerCase(c)`
`isUpperCase(c)`
`isSpaceChar(c)`
`isWhitespace(c)`

Konverze znaků
`toUpperCase(c)`
`toLowerCase(c)`

Třída Boolean

Vybrané konstanty

`FALSE` – konstanta typu `Boolean` deklarovaná ve třídě `Boolean`

`TRUE` – konstanta typu `Boolean`

Poznámka:

`false, true` – toto jsou rezervovaná slova jazyka Java představující konstantní hodnoty primitivního typu `boolean`

Některé metody

`parseBoolean(s)` – výsledek typu `boolean`

`toString(b)` – výsledek typu `String`

`valueOf(s)` – výsledek typu `Boolean`

`valueOf(b)` – výsledek typu `Boolean`

Instance obalových tříd

Vytvoření instance/objektu obalové třídy

- Použití metody konstruktoru
- Některé metody vracející hodnotu typu obalové třídy

```
Double dd;
dd = new Double(6.7);
dd = Double.valueOf(6.7);
dd = Double.valueOf(("126e-4");

dd = 6.7;
```

Konverze mezi primitivními datovými typy a obalovými třídami

- Novější verze Java – implicitní konverze
- Starší verze – využití konstruktoru nebo `valueOf()` metody

```
Double dd;
Double d;
dd = new Double(274.6);
d = dd;
d = 2.48;
dd = d;
```