Proč Java

- jednotný, univerzální jazyk (ne však vhodný pro cokoli)
- rozsáhlý, otevřený, rozvíjející se softwarový systém a technologie
- právně chráněný
- normotvorný - zavádí řadu užitečných norem, uzancí a přístupů
- expanzivní, od května 1995 vykazuje nevídaný rozmach
- velmi efektivní při tvorbě software
- spolehlivý, robustní, přesně specifikovaný, dobře dokumentovaný
- portabilní, (Windows / Linux / Solaris ...)
- bezpečný
- deterministický - úlohy končí či havarují určitelným způsobem
- silně objektový - obsahuje však i neobjektové primitivní typy
- síťový
- internetový a internacionální a návazný na nativní platformu
- multithreadový - více vláken (úkolů) může běžet (jakoby) současně
- interpretovaný - s adaptací pro platformu
- zdarma

Java & IDE

- Tvůrcem a vlastníkem práv Javy je fa. **Sun Microsystems**, která vydává **JavaSE, JavaEE, JavaME, Netbeans** a další produkty po verzích - zdarma stáhnutelné z: http://java.sun.com/ resp. http://www.netbeans.org

  Verze 1.0.2 - byla první , 1.1 – rozšíření o delegační model událostí
  1.2. - zahrnuje JFC, 1.3, 1.4 - rozšíření **assert**,
  1.5 - z 2004: rozšíření jazyka **for, enum**, autoboxing, generics,
  1.6 u 7 - z 2006, 1.7 "Dolphin" – v plánu.
  jre - Java Runtime Environment (bez vývojových prostředků)
  Toto software neobsahuje ani IDE, databázi či aplikační server.

- S Javou lze pracovat velmi nepohodlně řádkově:
  j2sdk1.6/bin/javac A.java B.java ... kompilace
  j2sdk1.6/bin/java A běh

- **či velmi pohodlně a bezpečně pomocí vyspělých IDE:**
  Netbeans (Sun: v. **6.1** (prosinec 2007), 6.5 beta **) ,**
  JBuilder - Borland, JDeveloper - Oracle, Eclipse – IBM, JCreator, ...

Podpora Javy

Sun Microsystems velmi podporuje Javu tím, že vydává:

- **různé softwarové svazky – ke stažení a instalaci lze doporučit:**
  na http://java.sun.com/javase/downloads/index.jsp
  JDK 6 Update 7 with NetBeans 6.1

- **ke každé hlavní verzi dokumentaci ve formě html stránek. Ta obsahuje zejména popis API, který je radno připojit nerozzipovaný IDE. Viz:**
  http://java.sun.com/docs/ Nejnovější verze je: jdk-6-doc.zip .

- **řadu tématických tutoriálů ve formě html stránek.**
  Viz http://java.sun.com/docs/books/tutorial/index.html
  http://java.sun.com/docs/books/jls
  - Java specification 3rd edition popisuje i verze >= 1.5

Pecinovský, R: Java 5.0, CP Books, Brno, 2005, s. 152, ISBN 80-251-0615-2

Jména a konvence

Jméno tvoří libovolná neprázdná posloupnost písmen Unicodu (tedy i z národních abeced), znaků $ _ , tj. dolar a podtržítko (underscore) a číslic, nezačínající číslicí a jež není klíčovým slovem. Jména je radno volit mnemotechnická, pouze z písmen ASCII a číslic.
Dolarem a podtržítkem radno šetřit - neboť se užívají i jinak.
Tečka se užívá jako selektor položky.

Jmény se označuje:

package (balíček) - jen malá písmena a číslice tečkou, např:
abc def2.gh3 java.util.zip

class (třída) - a její konstruktory - podstatné jméno začínající velkým písmenem, např:
String FileInputStream BankAccount
Rectangle2D.Double - pro tzv. vnitřní třídy

Výjimky by měly mít sufix Exception, např: MySpecialException

Jména a konvence

interface (rozhraní) - přídavné jméno začínající velkým písmenem, např:

Comparable Serializable Cloneable

method (metoda) - sloveso začínající malým písmenem, např:

reset parseDouble compareTo

vrací-li typ boolean, hodí se prefix is nebo has

isLetterOrDigit hasNextElement

variable (proměnná) - začínající vždy malým písmenem, např:

j counter totalTax currentFrequency

final variable (konstanta) a návěští:

- jen velká písmena, číslice a podtržítka, např:

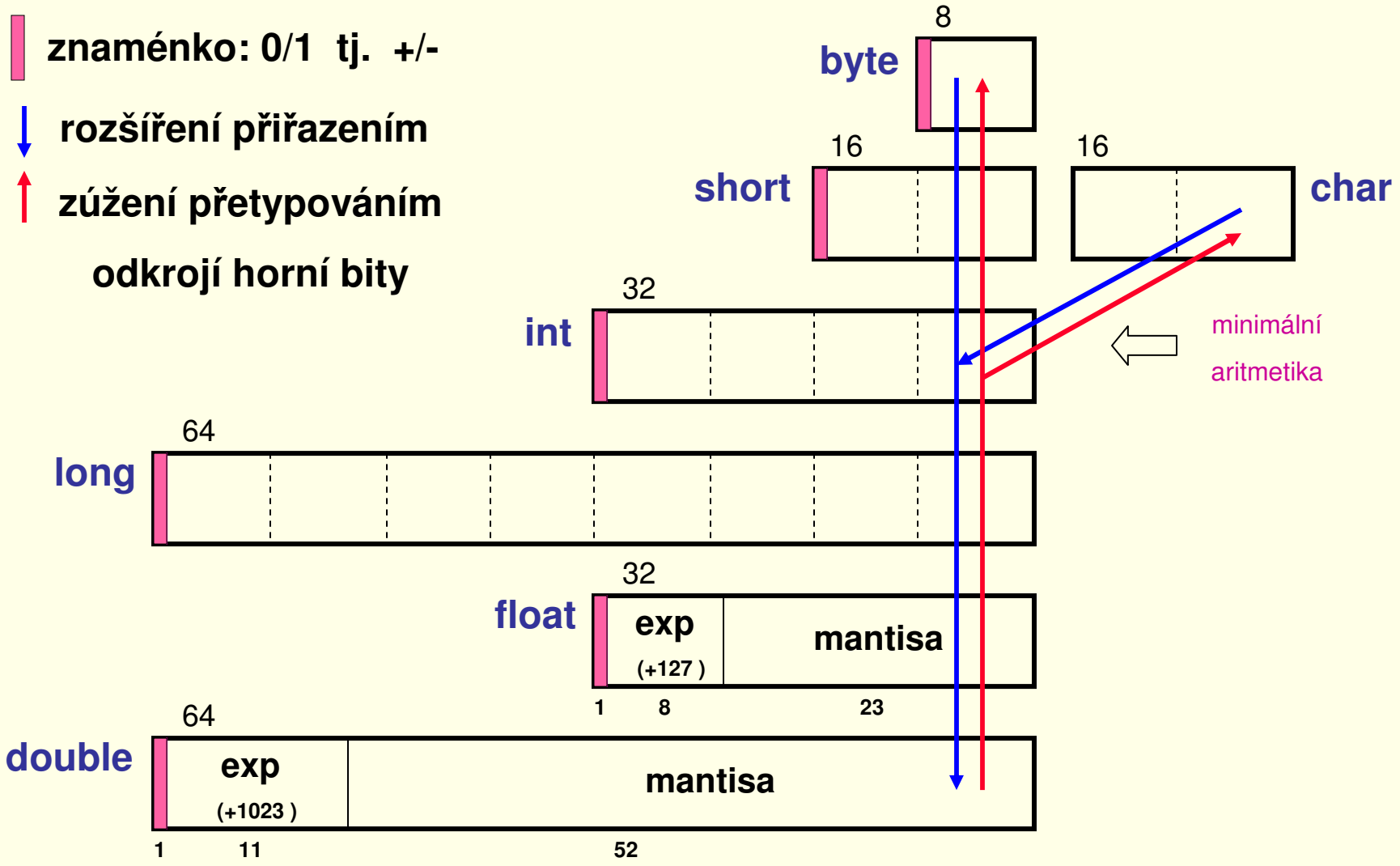
PI MAX_VALUE POINT_123 CYCLE3

Primitivní typy

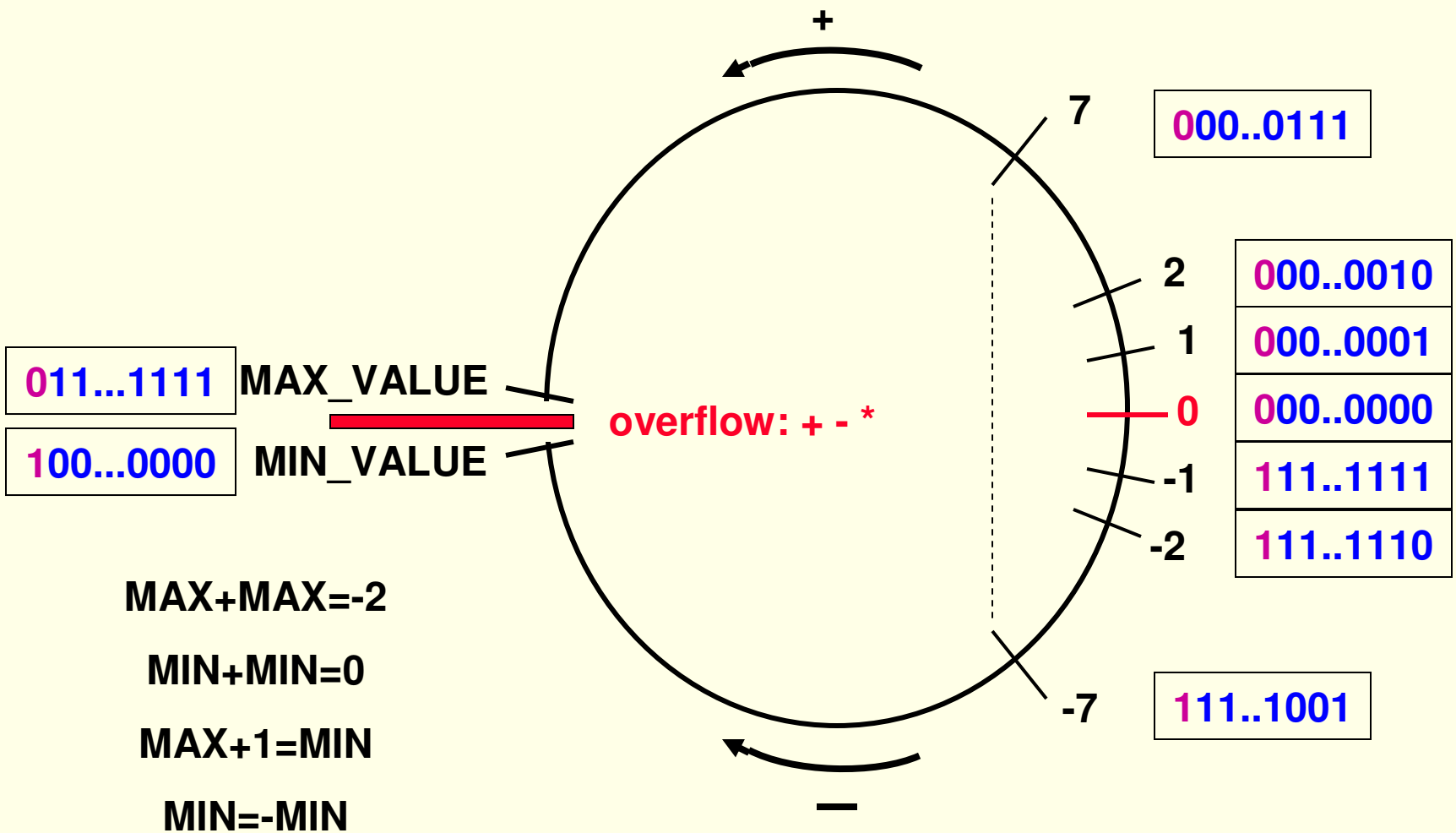
Typ	Formát	Obalové (wrap) třídy
	Celá čísla - dvojkový komplement se znaménkem	
byte	8-bit -128 … 127	**Byte**
short	16-bit -32768 … 32767	**Short**
int	32-bit -2147483648 … 2147483647	**Integer**
long	64-bit -9223372036854775808 … 9223372036854775807 (9.2E18 = 9.2 trilionů)	**Long**
char	16-bit **Unicode** '\u0000' to '\uffff' 0 ... 65535	**Character**
	"Reálná" čísla - dle normy IEEE 754	
float	32-bit 2^{-149} … $(2-2^{-23})*2^{127}$, NaN, pos/neg Infinity (1.4E-45 ... 3.4E38)	**Float**
double	64-bit 2^{-1074}… $(2-2^{-52})*2^{1023}$, NaN, pos/neg Infinity (4.9E-324 ... 1.7E308)	**Double**
boolean	**true false**	**Boolean**
void	**pomocný prázdný typ**	**Void**

Konverze primitivních typů

Primitivní typy jsou vzájemně nekompatibilní, ale jejich hodnoty lze převádět.



Integrální aritmetika int a long



Numerické literály

Obsahuje-li literál jen číslice a příp. znaménko ukládá se do typu int.
např.: -123, 123, +123

Pro uložení v typu long se sufixem l či L
např.: 1230000000000000l nebo 1230000000000000L

Obsahuje-li literál tečku nebo exponent, ukládá se do typu double
např.: 123., +123.45, -123E2, -123e2, 12300e-2

Pro uložení v typu float se sufixem f či F
např.: -123.0f , +123.45E+3F

Oktalový literál se vyznačuje prefixem 0 (nula)
např.: 077 (dekadicky 63), 023131545646 46L

Hexadecimální literál se vyznačuje prefixem 0x (nula iks)
např.: 0x77 (dekadicky 119), 0x1DADA2CAFE3FEEL

Standardní aritmetiky

- **Integrální aritmetika pro celá čísla int a long s těmito operátory:**

  + - * / %

  << >> >>>

  & | ^ ~

  Pozor: Přetečení se nijak nehlásí !

  Pouze dělení nulou vyhazuje výjimku java.lang.ArithmeticException.

- **"Reálná" aritmetika pro float a double čísla má operátory:**

  + - * / %

  Nevyhazuje výjimky, avšak zahrnuje tři speciální hodnoty:

  NaN - Not a Number, tj. nečíslo nebo neurčitý výraz
  POSITIVE_INFINITY
  NEGATIVE_INFINITY

  S těmito hodnotami lze provádět další operace, avšak již nelze získat opravdové číslo. Krom x != NaN je každé porovnání s NaN vždy false.

Použití obalových tříd

- `int iMax = Integer.MAX_VALUE;` // zjištění max. hodnoty
- `int k = Integer.parseInt( "123" );` // konverze řetěz - číslo
- `String s = Integer.toHexString(k);` // konverze do hexa
- `Integer I = new Integer( "123" );` // konverze řetěz - číslo

```
// obdobně pro byte, short, long
```

- `double dMax = Double.MAX_VALUE;` // zjištění max. hodnoty
- `double x = Double.parseDouble( "-123.45" );` // konverze řetěz - číslo
- `boolean b = Double.isNaN(x);` // test na nečíslo

```
// obdobně pro typ float
```

- `char c = Character.toLowerCase( 'Z' );` // převod znaku
- `boolean b = Character.isLetterOrDigit( '@' );` // je znak písmeno / číslice?
- `int i = Character.digit( "f" , 16 );` // hexadec. hodnota znaku

Java 1.5 zavedla zjednodušený zápis tzv. autoboxing, který automaticky zabaluje/rozbaluje primitivní typy do/z příslušných obalových objektů.

Matematické operace

K dispozici jsou třídy:

- **Pro počítání v typech int, long, float a double:**

  java.lang.Math

  java.lang.StrictMath je identická, strictfp
  – požaduje přesné dodržování normy IEEE 754, byť hardware umožňuje více

- **Pro čísla s libovolnou přesností existují třídy:**

  java.math.BigInteger - pro celá čísla

  java.math.BigDecimal - pro desetinná reálná čísla

  Výpočty provádějí metodami - nikoli operátory.

Znakové a řetězové literály

- **Znakový literál představuje právě jeden znak v Unicodu a uvádí se v apostrofech např.: 'A' 'Ř' '@' . Lze ho vyjádřit čtyřmi hexadecimálními číslicemi s prefixem \u např.: '\uCafe' .**

  Leč nelze použít \u000A či \u000a je to přechod na novou řádku.

  Inu trapné, užijte tedy \n či \r .

- **Řetězový literál je posloupnost znaků Unicodu (i prázdná) v uvozovkách např.: "Abc123" "Řeřicha" "@" "" .**

  Lze vložit znaky Unicodu, např.: "My \uCafe & Tea" .

Řetězový literály se ukládají při kompilaci do poolu literálů.

Řetěz není primitivní typ – je to objekt referenčního typu java.lang.String, který je imutabilní.

Na řetězovém literálu lze volat všechny metody třídy String.

Operátory

= přiřazení

+ - sčítání / odčítání aritmetické nebo unární plus / minus

++ -- inkrement / dekrement pre a post

***** násobení

/ dělení, integrální dělení jsou-li oba operandy integrálních typů

% modulo, tj. zbytek po dělení

<< >> >>> posun vlevo/vpravo aritmetický resp. logický (nuly zleva) – jen int a long. Posun o k%32 resp. k%64 pro int resp. long.

+ zřetězení – je-li alespoň jeden operand řetěz

instanceof - test referenčního typu

(type) - přetypování

Složené přiřazovací operátory

= += -= *= /= %= <<= >>= >>>= &= ^= |=

jsou zkratky zápisu, např. pro int resp. String s lze napsat výraz:

i += 2 místo i = i + 2 resp. s += “XYZ” místo s = s + “XYZ”;

Složené přiřazení udělá automatické přetypování.

Kompilátor nepřipustí i = i + 2.9, kdežto i += 2.9 je jako i += 2.

- Přiřazovací operátory jsou zprava asociativní, tedy výraz
  a = b = c je ekvivalentní a = (b = c)
- Na levé straně přiřazovacího operátoru musí být proměnná.
- Výsledkem přiřazení je hodnota proměnné po provedeném přiřazení a proto je následující příklad chybný:

```
double d;
int i = d = 10;   // Error: possible loss of precision: double, required int
```

Ternární podmíněný výraz

- Operátor ? : vrací hodnotu jednoho nebo druhého výrazu podle hodnoty boolského výrazu:

```
int abs(int i) { return i >= 0 ? i : – i; }
```

- Podmíněný operátor je zprava asociativní, takže

  a ? b : c ? d : e ? f : g

  znamená toto:

  a ? b : (c ? d : (e ? f : g))

- První operand musí být typu boolean.
- Druhý a třetí operand musejí být kompatibilního typu.

  Výsledek zkonvertován na "obecnější" z obou typů:

```
    double d;
    boolean zaokrouhlit;
    …
    return  zaokrouhlit ? Math.rint(d) : d;
// vraci hodnotu typu double
```

Komparační operátory

Výsledkem porovnání je logická hodnota (true, false)

- **Porovnání primitivních typů**

==	!=	rovno, nerovno
>	>=	větší než, větší nebo rovno
<	<=	menší než, menší nebo rovno

- **Porovnání referenčních typů**

  == != porovnávají se pouze reference nikoli obsah objektů

  Porovnatelné objekty se porovnávají zpravidla metodami

  ```
  boolean equals( Object o )
  int  compareTo( Object o )
  int  compare( Object o1, Object o )
  ```

Bitové a logické operátory

Bitové a logické operátory sestupně podle priority:

! unární negace pro **boolean**

~ bitový komplement celého čísla

- **Oba operandy musejí být buď celočíselné anebo boolean.**

& AND

^ XOR

| OR

- **Operátory pro zkrácené vyhodnocení typů boolean:**

&& AND

|| OR

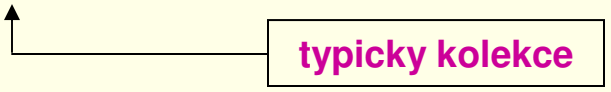
- **Je-li výsledek dán prvním operandem, zbytek výrazu se nevyhodnocuje.**
- **Pozor na vedlejší efekty, je-li dalším operandem např. volání metody.**
- **Pozor na nižší priority && a || :** false && true | true dá false

false && true || true dá true

(false && true) | true dá true

Základní konstrukty

```
[ public | protected | private ] [ static ] [ transient ] [ volatile ]  // pro atributy
[ final ] typ [ [ ] ] id [ = výraz ] , id [ = výraz ] , ... ;  // lokální  var. s inicializací

 { příkaz; blok; definice ... ; příkaz; ... ; }        // blok

 if  ( podmínka )  příkaz1 [ else  příkaz2 ]

 podmínka  ?  výraz1 : výraz2

[ návěští :]   while  ( podmínka ) příkaz
[ návěští :]   do  příkaz  while ( podmínka ) ;
[ návěští :]   for  ( [ inicializace ] ; [ podmínka ] ; [ modifikace ] ) příkaz
[ návěští :]   for  ( [ final ] typ id : ( iterable | pole ) ) příkaz

    continue [ návěští ];    // jen v cyklu
    break [ návěští ];        // jen v:  cyklu /  case / default
    return [ výraz ];
```



 - metasymboly

Základní konstrukty

```
- switch  ( index )  {          // index:  char, byte, short či int, enum,
                                // Character, Byte, Short, Integer - ne null
     [ case k1 :  [ příkazy ]   // k musí být různé a   int  literal  anebo
       case k2 :  [ příkazy ]   // int - kompatibilní final výraz  čili i
       case k3 :  [ příkazy ]   // enum konstanta.  Nikoli obalená hodnota.
       ...   ]
      [ default  : [ příkazy ] ]   // nezávisle volitelné a kdekoli v pořadí
      [ case k4 : [ příkazy ]
       ...   ]
    }                           // typ indexu indukuje typ konstant  k

 - try { [ příkazy ] }
     [ catch ( typVyjímky1   parm )  { [ příkazy ] }
       catch ( typVyjímky2   parm )  { [ příkazy ] }
     ...   ]
     [ finally { [ příkazy ] } ]    // nelze vynechat catch i finally

- throw  Throwable                  // vyhoditelný objekt typu Throwable
```

Základní konstrukty

- Class [<T>] = T.class // popisný objekt typu

- synchronized (výraz) blok // výraz musí referovat objekt

- assert podmínka [: výraz] ; // kontrola podmínky od v. 1.4

- enum jméno { // výčtový typ od v. 1.5 :
  ABC [(atributy)] [{ metody }] ,
  DEF [(atributy)] [{ metody }] ,
  GHI [(atributy)] [{ metody }] ;
  ...
  // další složky
  }

Syntax metod (bez generiky)

```
[ public | protected | private ] [ static ] [ final ] [ synchronized ]
(   [ native | strictfp ]                    //  pro konkrétní
abstract  [ public | protected ]              //  pro abstraktní
)

( returnTyp | void )  jmenoMetody
(  [ [ final ] Typ parm,  ...  [ [ final ] Typ ... varargs ] ]  )
[ throws Ex1, Ex2, ... ]                     // ← vyznačení vyjímek

(  { ... }  )     // ← tělo pro konkrétní metody
                  //      pro returnTyp void může být tělo i prázdné
     ;            // ← středník pro abstraktní a nativní metody
```

Příklad volání: `jmenoMetody( x, y,  z1, z2, z3 );` possible autoboxing

Vrácená hodnota musí být kompatibilní s návratovým typem metody.

Klíčová slova a reservované literály

literály: **logické**: **false true**
referenční: **null**

typy: **primitivní(8)**: **boolean byte short int long char float double**

řídící příkazy: **if else for do while break continue**
switch case default return
pro výjimky a chyby: **throw try catch finally assert**

operátory: **instanceof new this super**

strukturální: **package import import static**
class enum interface extends implements

modifikátory: **přístupu**: **public protected** *friend se nepíše* **private**
atributů, metod a tříd: **final static**
atributů: **transient volatile**
metod: **synchronized native void throws**
metod, tříd, interfejsů : **abstract strictfp**

nefunkční: **const goto**

Pozn: toto rozdělení neodpovídá úplně přesně specifikaci Javy.