Interreg
EVROPSKÁ UNIE
Rakousko-Česká republika
Evropský fond pro regionální rozvoj

INFORMATIKA

Úvod do programování JAVA

UNIVERSITY OF APPLIED SCIENCES UPPER AUSTRIA
VŠTE
EVROPSKÁ UNIE

Obsah

1. ÚVOD DO PROGRAMOVÁNÍ V JAVA

Tvorba programů pro javovskou platformu probíhá ve dvou fázích. V prvním kroku napíšeme tzv. zdrojový kód. Zdrojový kód je zápisem programu. Ve druhém kroku je zdrojový kód přeložen do bajtkódu. Překlad zdrojového kódu do bajtkódu zajistí překladač (kompilátor, angl. compiler).

Bajtkód (angl. bytecode) jsou instrukce pro javovský virtuální stroj (angl. Java Virtual Machine, JVM).

1.1. Třídy

Program v Javě se skládá z jedné nebo více tříd. Třídu deklarujeme pomocí klíčového slova class. Třída může obsahovat metody. Každá třída i metoda má jméno (s výjimkou anonymních tříd, o kterých zde ovšem nebude řeč). Obsah třídy i metody je uzavřen ve složených závorkách { a }. Do metod zapisujeme příkazy. Každý program musí obsahovat metodu main, která má vždy stejnou hlavičku:

```
public static void main( String[] args )
```

Význam jednotlivých slov v této deklaraci probereme později. Pro začátek budeme zapisovat příkazy pouze do metody main. Tato metoda slouží jako vstupní bod do aplikace. Spuštění programu probíhá tak, že JVM zavolá tuto metodu. Příkazy v ní se provádějí jeden po druhém v pořadí, v němž jsou zapsány.

K výstupu na obrazovku slouží metoda System.out.println(). Program, který vytiskne jednoduchý pozdrav, vypadá takto:

```
public class Prvni {
   public static void main( String[] args ) {
      System.out.println( ”ahoj” );
   }
}
```

Tento zdrojový text deklaruje třídu Prvni, která obsahuje metodu main.

1.2. Javovský virtuální stroj

Javovský virtuální stroj (angl. Java Virtual Machine, JVM). JVM umožňuje spouštět javovské programy. Úkolem JVM je poskytovat programům vždy stejné prostředí. Stejné prostředí zajistí, že javovský program běží na počítači s procesorem SPARC a operačním systémem Solaris stejně jako na počítači s procesorem Intel a operačním systémem MS Windows.

Spuštění javovského programu probíhá ve dvou krocích: v prvním kroku spustíme JVM a ve druhém kroku JVM nahraje do paměti bajtkód a spustí jej (zavolá metodu main). Paměť programu je rozdělena na tři oblasti: zásobník (angl. stack), halda (angl. heap) a oblast metod (angl. method area). Oblast metod obsahuje program (bajtkód). Program je tvořen posloupností instrukcí JVM. Každá instrukce se skládá z jednobajtového operačního znaku a případně operandů. Např. instrukce goto má operační znak 167 a provede nepodmíněný skok na adresu určenou operandem.

Zásobník slouží pro alokaci lokálních proměnných, pro ukládání parametrů a výsledků instrukcí JVM a pro předávání parametrů a výsledků při volání metod. Paměť zásobníku se přiděluje po tzv. rámcích (angl. stack frames). Každé volání metody alokuje nový rámec. Rámec obsahuje paměť pro lokální proměnné a tzv. zásobník operandů (angl. operand stack). JVM udržuje ukazatel na tzv. vrchol zásobníku, což je naposledy vložená hodnota. Zásobník operandů se používá pro parametry a výsledky instrukcí JVM. Paměť běžícího programu Např. instrukce iadd vyzvedne dvě hodnoty z vrcholu zásobníku, tyto hodnoty sečte a výsledek umístí na vrchol zásobníku.

Každá položka operandového zásobníku má délku nejméně 4 bajty. Všechny aritmetické instrukce JVM tedy pracují s operandy délky nejméně 32 bitů. V oblasti proměnných se paměť přiděluje také po 4 bajtech. To znamená, že každá proměnná zabírá v JVM nejméně 32 bitů.

Halda je oblast paměti, v níž můžeme za běhu programu vytvářet objekty. Vytváření objektů na haldě se provádí pomocí klíčového slova new. Na rozdíl např. od jazyka C++ není potřeba objekty odstraňovat (tzv. dealokovat). O odstraňování objektů se v Javě stará tzv. garbage collector. Garbage collector sám rozpozná nepoužívané objekty a tyto objekty dealokuje. Po dealokaci objektu je paměť, kterou objekt zabíral, připravena k dalšímu použití.

Popis programovací jazyka rozdělujeme na dvě části: syntaxi a sémantiku. Syntaxe (angl. syntax) popisuje pravidla pro správný zápis. Např. říká, kde je nutné psát závorky. Je-li program syntakticky správně, lze jej přeložit a spustit. Sémantika (angl. semantics) definuje

význam jednotlivých jazykových konstrukcí. Např. říká, že operátor = provede přiřazení. Aby program dělal to, co po něm chceme, musí být sémanticky správně. V tomto textu se budeme zabývat syntaxí i sémantikou. Probereme postupně většinu syntaktických konstrukcí jazyka Java a u každé si řekneme její sémantiku. Při zápisu programu používáme tzv. klíčová slova (angl. keywords). Jsou to slova, která mají v jazyce speciální význam (dvě z nich, const a goto, zůstávají ovšem nepoužitá). Java rozlišuje malá a velká písmena. Např. Class je něco jiného než class. Klíčová slova jsou pouze z malých písmen.

1.3. BlueJ

BlueJ je volně šiřitelné multiplatformní vývojové prostředí vyvinuté speciálně pro výuku objektově orientovaného programování v jazyce Java. Umožňuje studentům navrhovat diagram tříd vyvíjené aplikace ve zjednodušené verzi jazyka UML. Hlavní výhodou BlueJ je jeho interaktivnost – umožňuje vytvářet instance jednotlivých tříd, zasílat jim zprávy a volat jejich metody. (Wikipedia)

1.4. Instalace

Nedříve je třeba nainstalovat Javu. Z oracle.com si stáhněte Balík Java JDK (Java Development Kit.)

BlueJ je možné stáhnout na adrese BlueJ.org. Obojí si nainstalujte. Návod na používání programu BlueJ naleznete například na této adrese: https://www.bluej.org/tutorial/tutorial-czech.pdf

1.5. Tvorba proměnných

Do proměnných se ukládají hodnoty. Proměnná je pojmenované místo v paměti. V rámci tvorby proměnné – její deklarace, v programovacím jazyce Java musí být u každé proměnné nejdříve řečeno, jaký datový typ bude obsahovat viz příklad 1 v našem případě slova int, boolean, char. Jejich význam si vysvětlíme v dalším odstavci.

Následuje název proměnné. Názvy proměnných píšeme bez diakritiky a mezer. Mezery mezi slovy se prostě vynechávají a každé další slovo začíná velkým písmenem. Výraz uzavíráme středníkem.

Příklad 1

```
int a;
boolean IsTrue;
char CarConsumption;
```

V příkladu 1 jsme vytvořili proměnné, ale nepřiřadili jsme jim žádné hodnoty. V příkladu 2 jsme vytvořili proměnné, kterým jsme přiřadili konkrétní hodnoty. První přiřazení do proměnné nazýváme inicializace.

Příklad 2

```
int a=2;
boolean IsTrue =true;
char CarConsumption ='A';
```

Chceme-li deklarovat více proměnných najednou, musíme je oddělit čárkou:

```
int a, b;
```

Budeme rozlišovat mezi výrazem a příkazem. Výraz má vždy nějakou hodnotu. Tuto hodnotu získáme vyhodnocením výrazu. Např. 42 a x + 1 jsou výrazy. Příkaz je kód, který něco provede. Např. x = 1; je příkaz, který přiřadí hodnotu 1 do proměnné x. Máme-li výraz, při jehož vyhodnocení se něco provede, můžeme z něj obvykle udělat příkaz tak, že za něj napíšeme středník. V takovém případě říkáme, že vyhodnocení tohoto výrazu má vedlejší efekt.

2. DATOVÉ TYPY

2.1. Celo číselné

Int – Z anglického *integer* – celé číslo. Můžeme tedy zadávat pouze celá čísla v rozsahu od -2,147,483,648 do 2,147,483,647, včetně nuly. Integer může reprezentovat například počet operací počítače za sekundu.

Další celočíselné datové typy jsou vypsány v tabulce:

Typ	**Počet bajtů**	**Rozsah**	
byte	1	-128	127
short	2	-32 768	32 767
int	4	-2147 483 648	2147 483 647
long	8	-9,22 x 1018	9,22 x 1018

Stane-li se, že výsledek operace neleží v přípustném intervalu daného datového typu, nastává přetečení (angl. overflow). V takovém případě má výsledek opačné znaménko než výsledek matematické operace. V Javě nezpůsobí přetečení chybu, je však třeba s ním počítat. Např. pokud máme v proměnné typu byte hodnotu 127 a přičteme k ní 1, bude v proměnné −128.

2.2. Reálná čísla

Pro práci s reálnými čísly má Java dva primitivní datové typy: float a double. Oba používají stejný způsob reprezentace: reálné číslo je uloženo jako trojice znaménko, mantisa a exponent. Čísla tak ukládáme pouze přibližně. Typ float používá 32 bitů: 1 bit zabírá znaménko, 8 bitů exponent a 23 bitů mantisa. Typ double používá 64 bitů: 1 bit pro znaménko, 11 bitů pro exponent a 52 bitů pro mantisu.

2.3. Konverze

Konverze (přetypování) je převod hodnoty na hodnotu jiného datového typu. Např. převod hodnoty typu double na hodnotu typu int. Některé konverze se provádějí automaticky, např.

z typu int na typu double, jiné si musíme vyžádat, např. z typu long na typ int. Automatické konverze není nutné zapisovat:

```
int a = 100;
long a = b;  // Automatic conversion from int to long
```

Je-li třeba konverzi předepsat, uvádíme cílový typ v závorkách před konvertovanou hodnotou. Např. konverzi z double na int zapíšeme takto:

```
double d1 = 5.85;
int i1 = (int) d1;
```

Výsledkem této konverze bude hodnota, která je v zápise původního čísla před desetinnou tečkou (pro nezáporné hodnoty je to celá část čísla). Tj. v proměnné i2 bude hodnota 5.

```
double d2 = -4.99;
int i2 = (int) d2;
```

V proměnné i2 bude hodnota −4. Připomeňme, že to není celá část čísla, protože celá část čísla −4.99 je −5.

2.4. Logické

Boolean - Obsahuje pouze výrazy true a false (Pravda a nepravda). Je také nazývána jako logická proměnná. Pokud bychom porovnávali dvě číselné proměnné ve výrazu a<b. Nazvěme je *a* a *b*. pokud je *a* skutečně menší než *b* pak je výraz a<b pravdivý a do proměnné c se uloží hodnota true. Pokud výraz pravdivý není do c se uloží false

```
int a = 5;
int b = 6;
boolean c = a<b;
System.out.println(c);
```

- V tomto příkladu systém vypíše „true“.
- Hodnota false je vnitřně reprezentována jako 0, hodnota true jako 1.

2.5. Textové

Char - Z anglického character – znak či symbol. Můžeme do něj ukládat jeden znak. Před tímto a za tímto znakem musí být apostrofy (jednoduché uvozovky). Viz příklad 2. Vypisovat můžeme znaky z tabulky Unicode. Hodnotu typu char lze chápat jako index (pořadí) znaku v tabulce Unicode.

```
char c ='H';
System.out.println(c) ;
```

2.6. Textové řetězce

Pro práci s řetězci používáme v Javě typ String. Řetězcové konstanty zapisujeme do uvozovek.

```
String s = " Hi how are you?";
```

Řetězce můžeme spojovat pomocí operátoru +.

```
String s1 = "jdk", s2 = "7.0";
String s3 = s1 + s2;  // Creates a string "jdk7.0"
```

K řetězci lze přičíst i hodnotu jiného typu. V takovém případě se hodnota nejprve převede na řetězec a poté se oba řetězce spojí.

```
int x = 42;
String s = "answer is " + x;
```

Příklad vytvoří řetězec "odpověď je 42".

2.7. Komentáře

V Javě jsou tři druhy komentářů:

- jednořádkový - začíná znaky // a pokračuje do konce řádky
- víceřádkový - začíná znaky /* a končí znaky */
- dokumentační – začíná znaky /** a končí znaky */

Dokumentační komentáře jsou určeny pro zpracování programem javadoc, který generuje dokumentaci ve formátu HTML.

```
/**
* Main program class.
*/

public class Main {
      public static void main(String[] args) {
            /* Prints the answer */
            System.out.println(42); // answer is 42
      }
}
```

Prostřednictvím komentářů vkládáme do zdrojového textu doplňující informace. Překladač komentáře přeskakuje, tudíž na běh programu nemají žádný vliv. Komentovat bychom měli např. význam proměnných, neobvyklé postupy a netradiční algoritmy. Tedy místa, jejichž význam nemusí být pro člověka čtoucího kód na první pohled zřejmý.

2.8. Terminálový vstup a výstup

V této kapitole si ukážeme, jak lze v Javě zapisovat na obrazovku a číst z klávesnice. K výstupu na obrazovku slouží tzv. výstupní proud (angl. output stream) System.out. Používat budeme tři jeho metody: print(), println() a printf(). Volání níže vypíše: Ahoj Babi

```
System.out.println( ” Hi Baby!” );
```

Metody print() a println() vytisknou hodnotu, kterou uvedeme v závorkách za jménem metody (této hodnotě říkáme parametr). Liší se tím, že println() k výstupu navíc připojí přechod na nový řádek a tudíž následující volání metody print() nebo println() bude tisknout od začátku řádky. Při tisku lze využít spojování řetězců pomocí operátoru +.

```
int v = 200;
System.out.println( ”Car moved ” + v + ” km/h” );
```

Elegantnější výstup nabízí metoda printf() (tzv. formátovaný výstup), která je obdobou stejnojmenné funkce jazyka C.

```
int m = 6;
System.out.printf( ” African elephant weighs %d tons”, m );
```

Parametry metody printf() jsou formátovací řetězec a seznam hodnot. Formátovací řetězec může obsahovat tzv. výstupní konverze, které určují, v jakém tvaru se vytiskne příslušná hodnota. Každá konverze začíná znakem %. Např. %d znamená výstup celého čísla v desítkové soustavě. Při provádění příkazu se místo konverze dosadí příslušná hodnota ze seznamu hodnot. Pokud hodnota chybí nebo neodpovídá výstupní konverzi, nastane chyba.

Zvláštním případem je konverze %n, které neodpovídá žádná hodnota v seznamu hodnot. Na výstupu se tato konverze projeví přechodem na nový řádek. V MS Windows se pro přechod na nový řádek používá dvojice znaků \r (carriage return) a \n (line feed). Unixové systémy používají znak \n. Konverze %n zajistí, že se použije správný přechod na nový řádek, tj. \r\n na MS Windows a \n na Unixu.

Pro reálná čísla máme konverzi %f. Můžeme u ní stanovit počet číslic za desetinnou tečkou (implicitní hodnota je 6). Např. %.2f vytiskne dvě číslice za desetinnou tečkou.

```
System.out.printf( ” Euler’s constant is about %.2f%n”,
Math.E );
```

Znaky tiskneme pomocí konverze %c.

```
char c = ’@’;
int i = c;
System.out.printf( ”Character ’%c’ Is in the Unicode table
in position %d%n”, c, i );
```

Pro řetězce používáme konverzi %s.

```
String Java = ”Java”;
System.out.printf( ” Our favorite programming language
is %s%n”, Java );
```

Pro čtení z klávesnice máme v Javě tzv. vstupní proud (angl. input stream) System.in. Většinou jej nepoužíváme přímo, ale např. přes třídu Scanner. Tato třída nabízí metody pro čtení primitivních typů a řetězců. Používáme-li třídu Scanner, náš program obvykle začíná příkazem import, kterým překladači říkáme, kde má třídu Scanner hledat. Před prvním čtením vytvoříme instanci třídy Scanner pomocí klíčového slova new. Načtení hodnoty typu int provedeme voláním metody nextInt() na této instanci.

```
import Java.util.Scanner;
public class Read {
      public static void main( String[] args ) {
            Scanner sc = new Scanner( System.in );
            int x = sc.nextInt();
            System.out.printf( ” Readed value is: %d%n”, x );
      }
}
```

Načtení hodnoty typu double zajistí metoda nextDouble(). Řetězec načteme metodou next().

```
Scanner sc = new Scanner( System.in );
double d = sc.nextDouble();
System.out.printf( ”Readed number is: %f%n”, d );
String s = sc.next();
System.out.printf( ”Readed string is: %s%n”, s );
```

3. OPERÁTORY

S čísly lze v Javě provádět běžné aritmetické operace: sčítání, odčítání, násobení, dělení, modulo (zbytek po celočíselném dělení). K zápisu operací slouží **operátory**. Např. sčítání zapisujeme pomocí operátoru + a modulo pomocí operátoru %. Zápis x + 2 je tedy zápisem operace sčítání. Každý operátor pracuje s jednou či více hodnotami nebo proměnnými, kterým říkáme **operandy**. Podle počtu operandů můžeme operátory rozdělit na unární (s jedním operandem), binární (se dvěma operandy) a ternární (se třemi operandy). Unárním operátorem je např. ++ (inkrementace) a binárním operátorem je např. = (přiřazení). Jediným ternárním operátorem je ?: (podmíněný operátor). Výsledkem provedení operátoru je hodnota (říkáme, že operátor vrací hodnotu). Např. binární operátor + (sčítání) vrací součet svých operandů. Typ výsledku závisí na operátoru a někdy i na operandech. U operátorů, které vrací celočíselnou hodnotu, je výsledek buď int nebo long. Např. sečteme-li dvě hodnoty typu int, výsledek bude int, sečteme-li dvě hodnoty typu long, výsledek bude long, a sečteme-li dvě hodnoty typu byte, bude výsledek int. Operátor / (děleno) vrací pro celočíselné operandy celočíselný podíl. Např. 15 / 4 je 3 a −15 / 4 je −3.

Je-li hodnota druhého operandu 0, nastane chyba. Je-li alespoň jeden operand reálný (tj. float nebo double), je i druhý operand převeden na reálný, a operátor děleno vrátí podíl obou operandů. Např. 4.5 / 3 je 1.5. Je-li druhým operandem 0, bude výsledkem reálného dělení některá z těchto hodnot: plus nekonečno, je-li první operand kladný, mínus nekonečno, je-li druhý operand záporný, nebo NaN (Not a Number), je-li první operand kladné nekonečno, záporné nekonečno, NaN či 0. Operátor % (modulo) vrací zbytek po celočíselném dělení. Např. 17 % 4 je 1. Tento operátor je definován i pro záporné hodnoty, ovšem jinak než tomu bývá v matematice. Znaménko výsledku je vždy stejné jako znaménko prvního operandu: −17 % 4 je −1, 17 % −4 je 1 a −17 % −4 je −1.

Operátory lze řetězit, tj. lze zapisovat výrazy, které obsahují více operátorů. Např. x + 2 * y je výraz, který obsahuje operátory sčítaní a násobení. Pořadí vyhodnocování operátorů určuje **priorita** operátorů (angl. precedence). Např. ve výrazu x + 2 * y se provede nejprve násobení a pak sčítání, protože operátor násobení má vyšší prioritu než operátor sčítání. Jiné pořadí vyhodnocení lze předepsat závorkami: (x + 2) * y. Použijeme-li ve výrazu více operátorů se stejnou prioritou, rozhoduje o pořadí vyhodnocování **asociativita** operátoru (angl. asociativity). Např. ve výrazu x − y − z se provede nejprve x − y a poté se od výsledku odečte z. Říkáme, že operátor odčítání asociuje zleva doprava. Výraz x − y − z má tedy stejnou hodnotu jako výraz (x − y) − z.

Některé operátory mají asociativitu opačnou, tj. zprava doleva. Příkladem je operátor přiřazení. Návratovou hodnotou tohoto operátoru je hodnota levého operandu po přiřazení. Ve výrazu x = y = 1 se provede nejprve přiřazení y = 1 a poté se návratová hodnota operátoru (v tomto případě 1) přiřadí do x. Výraz x = y = 1 se tedy vyhodnocuje stejně jako x = (y = 1).

3.1. Operátory inkrementace a dekrementace

Operátor inkrementace (++) způsobí zvětšení hodnoty proměnné o jedničku. Lze jej zapisovat dvěma způsoby: prefixově a postfixově. V prefixové notaci operátor předchází svůj operand, v postfixové jej následuje. V obou případech dojde k inkrementaci proměnné, rozdíl je však v návratové hodnotě operátoru. Prefixový operátor vrací hodnotu proměnné po zvětšení, postfixový operátor hodnotu před zvětšením.

```
int x = 1;
int y = ++x;
```

Ve výrazu y = ++x se provede nejprve operátor inkrementace, protože má vyšší prioritu než operátor přiřazení. Dojde tedy ke zvýšení hodnoty x o jedničku. Návratovou hodnotou tohoto operátoru je hodnota x po zvětšení, tj. 2. Ta se použije jako operand operátoru přiřazení. Do proměnné y se tedy uloží hodnota 2.

```
int x = 1;
int y = x++;
```

Ve výrazu y = x++ se provede nejprve operátor inkrementace. Jeho návratovou hodnotou je hodnota proměnné x před zvětšením, tj. 1. Do proměnné y se tedy uloží hodnota 1.

Operátor dekrementace (−−) způsobí snížení hodnoty proměnné o jedničku. Používá se obdobně jako operátor inkrementace.

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

3.2. Logické operátory

Logické výrazy můžeme spojovat dohromady pomocí logických operátorů. Logický operátor má operandy typu boolean a vrací hodnotu typu boolean. Existují 2 logické operátory .

Operátor **logického součinu**, nazývaný též *a zároveň* anglicky *and* se zapisuje && a vrací true tehdy a jen tehdy, pokud mají oba jeho operandy hodnotu true.

Příklad:

```
if( x == 0 && y == 0 ) {
      system.out.println( ”x and y are equal to zero” );
}
```

Operátor logického součtu , nazývaný též *a nebo,* anglicky or se zapisuje || a vrací true tehdy a jen tehdy, pokud alespoň jeden jeho operand má hodnotu true.

```
if( x == 0 || y == 0 ) {
      System.out.println( ” At least one of the numbers x,
      y is equal 0” );
}
```

Oba tyto operátory používají tzv. **zkrácené vyhodnocování**, tj. druhý operand se vyhodnotí pouze v případě, že po vyhodnocení prvního operandu není známa hodnota celého výrazu. Např. má-li v logickém součinu první operand hodnotu false, druhý operand se nevyhodnocuje a hodnota celého výrazu je false. Protože se tyto operátory používají často v podmínkách, říká se jim podmínkové.

Kromě podmínkových operátorů má Java také operátory, které vyhodnocují vždy oba operandy. Zapisují se & (logický součin) a | (logický součet). Lze je použít na stejném místě jako podmínkové operátory.

```
boolean b1 = x > 0 & y == 1;
boolean b2 = x <= 0 | y <= 0;
```

Kombinujeme-li logický součin s logickým součtem, je potřeba mít na paměti, že logický součin má vyšší prioritu než logický součet:

```
if( x == 0 || y > 0 && z > 0 ) {
      System.out.println( "x is zero or y and z are positive " );
}
```

3.3. Přiřazovací operátory

Jeden z přiřazovacích operátorů jsme již poznali, operátor =. Další přiřazovací operátory nám umožňují provést s proměnnou nějakou aritmetickou operaci. Např. operátor += přičte k proměnné hodnotu druhého operandu.

```
x += 5;
```

Dalšími přiřazovacími operátory jsou −=, *=, /=, %=. Každý z nich je zápisem příslušné operace s proměnnou na levé straně. Např. operátor %= provede operaci modulo:

```
x %= 6;  // stejné jako x = x % 6
```

3.4. Priorita provádění operací

Všechny operátory vrací hodnotu, která je výsledkem příslušné operace. Mají stejnou prioritu a asociují zprava doleva. V příkazu

```
int y = x + = 1;
```

se nejprve provede operátor += a teprve pak =. Operátor += vrátí hodnotu x po přičtení jedničky. Tato hodnota se použije jako hodnota druhého operandu operátoru =.

Probrané operátory si můžeme seřadit podle priority (od nejvyšší k nejnižší):

- inkrementace (++), dekrementace (−−)
- přetypování
- násobení (*), dělení (/), modulo (%)
- sčítání (+), odčítání (−)
- přiřazovací operátory (=, +=, −=, *=, /=, %=)

Priorita nám říká, jak silně se operátory váží na operandy. Např. přetypování má vyšší prioritu než násobení, proto se ve výrazu (int) d * 2 provede nejprve přetypování a až potom násobení.

```
double d = 5.8;
int i = (int) d * 2;  // same like ((int) d) * 2
System.out.println( i );
```

Tento kód tedy vytiskne hodnotu 10.

Přiřazovací operátory asociují zprava doleva a aritmetické operátory (sčítání, odčítání, násobení, dělení, modulo) zleva doprava.

Všechny binární operátory vyhodnocují operandy ve stejném pořadí: nejprve levý a pak pravý. Toto pořadí je významné, pokud má vyhodnocení operandů vedlejší efekt.

```
int x = 0;
int y = x + x++;
```

Na druhém řádku se nejprve vyhodnotí operátor +. Jeho levý operand má hodnotu 0, pravý operand má také hodnotu 0, operátor tedy vrátí 0 a ta se přiřadí do y. Při vyhodnocování pravého operandu dojde ke zvýšení proměnné x o 1 (vyhodnocení má vedlejší efekt). Prohodíme-li pořadí operandů, přiřadí se do y hodnota 1.

```
int x = 0;
int y = x++ + x;
```

3.5. Relační operátory a operátory rovnosti

Pro zápis podmínky používáme tzv. **relační operátory** a **operátory rovnosti**. Relační operátory jsou:

- menší než (<)
- větší než (>)
- menší nebo rovno (<=)
- větší nebo rovno (>=)

Operátory rovnosti jsou:

- rovná se (==)
- nerovná se (!=)

4. ZÁKLADNÍ PROGRAMOVACÍ STRUKTURY

4.1. If

K větvení programu slouží příkazy if a switch. Příkaz if umožňuje větvit program na základě nějaké podmínky. Začíná klíčovým slovem if, za nímž je v závorkách výraz typu boolean. Výraz typu boolean je výraz, jehož hodnota je true nebo false. Dále následuje příkaz. Při provádění se vyhodnotí výraz v závorkách a má-li hodnotu true (podmínka je splněna), provede se příkaz. V příkladu níže se testuje zda-li je proměnná x rovna nule. V případě, že bude rovna nule, přiřadí se proměnné x hodnota 2.

```
if (x==0) x=2;
```

Pro zápis podmínky používáme relační operátory a operátory rovnosti.

Příkaz if může obsahovat větev else, která se provede, pokud podmínka za if není splněna. Pokud podmínka není splněna a větev else chybí, neprovede se nic (bude se pokračovat za příkazem if).

```
if( x == 0 )
      System.out.println( ” Can not be divided by zero!” );
else
      z=5/x;
```

Chceme-li zapsat za podmínku více příkazů, použijeme blok. Blok začíná otevírací složenou závorkou { a končí zavírací složenou závorkou }. Blok je v Javě příkaz, takže jej můžeme použít všude tam, kde se může vyskytovat příkaz.

```
if( x == y ) {
      x++;
      y--;
}
```

Blok omezuje platnost deklarace proměnné. Každá deklarace je platná jen do konce bloku, v němž je uvedena. Říkáme, že je v tomto bloku **lokální**.

```
if( x == y ) {
    int z = y;
 }  // tato závorka ukončuje platnost deklarace proměnné z
 // zde již z nelze použít
```

V závorkách za if můžeme použít proměnnou typu boolean.

```
// In the month variable we have the serial number of the month
     boolean isMay = (month == 5);
     if( isMay ) {
          System.out.println( ”time for love” );
}
```

Pro zápis opačné podmínky používáme operátor logické negace (!).

```
boolean isHere = true;
if( ! isHere ) {
     System.out.println( ”Is not here” );
}
```

Potřebujeme-li rozvětvit program např. podle hodnoty celočíselné proměnné, můžeme použít zřetězení příkazů if.

```
if( x == 1 ) {
      System.out.println( ”one” );
} else if( x == 2 ) {
      System.out.println( ”two” );
} else if( x == 3 ) {
      System.out.println( ”three” );
}
```

4.2. Switch

Takto zřetězené if (jako v minulém příkladu) můžeme někdy nahradit příkazem switch. Jeho zápis začíná klíčovým slovem switch. Za ním je v závorkách výraz typu int nebo String (nebo typu, který lze zkonvertovat na int) a dále blok, v němž je libovolný počet návěští case. Za každým case je uvedena konstanta (nebo konstantní výraz, což je výraz, jehož hodnota je známa při překladu), dvojtečka a posloupnost příkazů.

```
switch( x ) {
      case 1:
            System.out.println( ”one” );
            break;
      case 2:
            System.out.println( ”two” );
            break;
      case 3:
            System.out.println( ”three” );
}
```

Při provádění se vyhodnotí výraz za klíčovým slovem switch a jeho hodnota se začne srovnávat s hodnotami uvedenými za case (v pořadí, v němž jsou uvedeny). Jakmile dojde ke shodě, začnou se provádět příkazy. Provádění příkazů končí příkazem break. Pokud příkaz break chybí, provedou se všechny příkazy až do konce příkazu switch. Příkaz switch může obsahovat větev default, která se provede, pokud nedošlo ke shodě u žádného návěští case.

5. CYKLY

Cykly slouží k opakovanému provádění příkazů.

5.1. While

Cyklus while začíná klíčovým slovem while, za kterým následuje v závorkách podmínka a dále tělo cyklu. Tělem cyklu je buď příkaz, nebo blok. V rámci cyklu while se vyhodnotí se podmínka a je-li splněna, provede se tělo cyklu. Poté se znovu vyhodnotí podmínka a je-li splněna, opět se provede tělo cyklu, atd. Pokud podmínka není splněna, pokračuje se příkazy za cyklem. Jestliže na začátku není podmínka splněna, tělo cyklu se neprovede ani jednou. Cyklus while je tedy cyklus s počtem opakování 0 nebo více.

Příklad syntaxe:

```
While(condition) {
      // body
}
```

Konkrétní příklad

```
int x = 5;
while( x > 0 ) {  // Do until x is greater than zero
      System.out.println( x );
      x --;
}
```

Tento příklad vypíše čísla od 5 do 1. Proběhne tedy 5x.

5.2. Do while

Cyklus do while začíná klíčovým slovem do, za kterým je tělo cyklu, následuje klíčové slovo while a podmínka viz příklad syntaxe. Provádí se takto: nejprve se provede tělo cyklu, pak se vyhodnotí podmínka a je-li splněna, znovu se provede tělo cyklu, vyhodnotí se podmínka, atd. Není-li podmínka splněna, pokračuje se za cyklem. Tělo cyklu se tedy provede vždy alespoň jednou. Počet opakování je tedy 1 nebo více.

Příklad syntaxe:

```
do {
      // body
} while (condition);
```

Konkrétní příklad

```
do {
      System.out.println( x );
      x --;
} while (x > 0);
```

5.3. For

Cyklus for má tvar: for *(řídící proměnná cyklu; podminka; prikaz)* Provádí se takto: nejprve se provede inicializace řídící proměnné cyklu, pak se vyhodnotí podmínka a je-li splněna, provede se tělo cyklu. Poté se provede příkaz, znovu se vyhodnotí podmínka a je-li splněna, opět se provede tělo cyklu, atd. Inicializace řídící proměnné cyklu se tedy provede pouze jednou na začátku. Pokud na začátku není podmínka splněna, tělo cyklu se neprovede ani jednou.

Příklad syntaxe:

```
for (cycle variable; condition; command){
      // body
}
```

Konkrétní příklad

```
int a;
for( a = 1; a < 10; a++ ) {
      System.out.println( a );
}
```

Řídící proměnná cyklu může být v rámci cyklu nově deklarována. Tato deklarace je platná pouze v daném cyklu (tj. v hlavičce a těle cyklu). Říkáme, že proměnná je v tomto cyklu lokální.

```
for( int a = 1; a <= 10; a++ ) {
      System.out.println( a * a );
}
```

Libovolná z částí řídící proměnná cyklu, podmínka, příkaz může chybět. Chybí-li podmínka, jde o nekonečný cyklus (podmínka je stále splněna). Pokud není uvedena řídící proměnná cyklu nebo příkaz, neprovede se v daný okamžik žádný příkaz.

5.4. Break a continue

V těle cyklu můžeme používat příkazy break a continue. Příkaz break okamžitě ukončí provádění cyklu. Pokračovat se bude příkazy za cyklem.

```
int s = 100;
while( s > 0 ) {
      int n = sc.nextInt();
      if( n == 0 ) {
            break;
      }
      s -= n;
      System.out.println( s );
}
// Here will continue after the break executed
```

Příkaz continue ukončí provádění těla cyklu a přejde na vyhodnocení podmínky cyklu.

```
int s = 0;
do {
      int n = sc.nextInt();
      if( n == 0 ) {
            continue;
      }
      s += n;
      System.out.println( s );
      // here goes continue
} while( s < 100 );
```

6. STATICKÉ METODY

Doposud jsme zapisovali celý program do metody main. Pokud bylo třeba provést stejnou posloupnost příkazů na více místech, museli jsme tyto příkazy zopakovat. Tomu se lze vyhnout. Metody nám umožňují členit kód do logických celků a tyto celky opakovaně využívat. V této kapitole budeme pod pojmem metoda rozumět statickou metodu. Jednu statickou metodu již známe. Je to metoda main. Kromě metody main můžeme ve třídě deklarovat i další metody, např. metodu pro tisk informací o programu.

```
class MainClass {
      static void printInfo() {
            System.out.println( "Version: 1.0" );
            System.out.println( " Autor: 007" );
      }
      public static void main( String[] args ) {
            printInfo ();
      }
}
```

Deklarace statické metody začíná klíčovým slovem static. (viz příklad výše) Za ním následuje návratový typ a jméno metody. Návratovým typem může být libovolný javovský typ. Pokud metoda nevrací žádnou hodnotu, uvedeme jako návratový typ void. Jméno metody obvykle začíná malým písmenem. Pokud se jméno skládá z více slov, oddělujeme slova tak, že první písmena dalších slov píšeme velká. Např. spoctiPolomerKruzniceOpsane. Není zvykem používat ve jméně metody podtržítka. Za jménem metody je v závorkách seznam parametrů. Seznam parametrů je tvořen deklaracemi proměnných. Oddělovačem deklarací v seznamu parametrů je čárka.

6.1. Volání metody

V místě, kde chceme metodu provést, zapíšeme volání metody. Volání metody se skládá ze jména metody a seznamu parametrů. Návratovou hodnotou metody bude hodnota výrazu, který je uveden za příkazem return.

Na pořadí, v němž metody deklarujeme, nezáleží. Lze volat i metodu, která je deklarována později.

```
class MainClass {
      public static void main( String[] args ) {
            Scanner sc = new Scanner( System.in );
            int x = sc.nextInt();
            long factorial = countFactorial( x );
            System.out.printf( "%d! = %d%n", x, faktorial );
      }
      static long countFactorial ( int n ) {
            long fact = 1;
            for( ; n > 1; n-- ) {
            fact *= n;
      }
      return fact;
      }
}
```

V metodě typu void lze použít příkaz return bez parametrů. Toho se využívá pro předčasné ukončení metody.

```
// print rectangle a x b from @
static void printRectangle ( int a, int b ) {
      // The minimum rectangle side size is 2
      if( a < 2 || b < 2 ) {
            return;
      }
      for( ; a > 0; a-- ) {
            for( int i = 0; i < b; i++ ) {
                  System.out.print( '@' );
            }
            System.out.pritln();
      }
}
```

Příkaz return se může v metodě vyskytovat vícekrát. Provede se však vždy jen jednou jako poslední příkaz metody. Jeho provedení způsobí okamžité ukončení metody.

```
static boolean isPrimeNumber ( int n ) {
      if( n == 2 ) {  // 2 prime number
            return true;
      }
      if( n % 2 == 0 ) {
            // Even number is not a prime number (except 2)
            return false;
      }
      int sqrt = (int) Math.sqrt( n );
      for( int i = 3; i <= sqrt; i += 2 ) {
            if( n % i == 0 ) {
                  // We found a divisor, so n is not a
                  prime number
                  return false;
            }
      }
      return true;
}
```

7. INSTANČNÍ PROMĚNNÉ

Instančním proměnným říkáme instanční atributy nebo zkráceně atributy (angl. instance attributes nebo fields). Statické metody již známe. Deklarují se pomocí klíčového slova static.

Instanční metody se deklarují podobně jako statické. Na rozdíl od statických metod však jejich hlavičky neobsahují klíčové slovo static. Instanční metody často pracují s instančními atributy.

7.1. Pole

Pole umožňuje pracovat s více hodnotami stejného typu. Pro uložení např. dvaceti hodnot typu int můžeme buď zavést dvacet proměnných, nebo vytvořit pole o dvaceti prvcích. V mnoha případech se s polem pracuje snadněji. Typ pole zapisujeme v Javě pomocí hranatých závorek. Deklarace proměnné typu pole položek typu int vypadá takto:

```
int[] p;
```

Proměnná typu pole je tzv. reference. Bude obsahovat odkaz (referenci) na pole. Samotná deklarace pole nevytváří. Pole můžeme vytvořit pomocí klíčového slova new:

```
p = new int[6];
```

V tomto případě jsme vytvořili pole o šesti prvcích typu int. Potřebujeme-li počet prvků pole, použijeme názevPole.length. V našem případě tedy

```
p.length
```

Hodnota této proměnné je nastavena při vytvoření pole a nelze ji změnit (je pouze pro čtení).

```
System.out.printf( ”array p has %d elemets%n”, p.length );
```

K jednotlivým prvkům pole přistupujeme pomocí indexů, které zapisujeme do hranatých závorek:

```
p[1] = 5;
```

Indexy začínají vždy od nuly, První číslo bude mít index 0, druhé číslo bude mít index 1, třetí číslo bude mít index 2 atd Platnost indexu se kontroluje za běhu programu. Použití neplatného indexu způsobí běhovou chybu. Po vytvoření jsou prvky pole inicializovány na hodnoty, jejichž vnitřní reprezentace je 0. U číselných typů je to 0, u typu boolean je to false a u typu char je to znak na pozici 0 v tabulce Unicode. Pro načtení hodnot do pole používáme obvykle cyklus for:

```
int[] p = new int[10];
      for( int i = 0; i < a.length; i++ ) {
            p[i] = i;
}
```

Výpis prvků pole provedeme nejčastěji opět cyklem for:

```
for( int i = 0; i < a.length; i++ ) {
      System.out.println( a[i] );
}
```

Délka pole musí být nezáporná. Pokus vytvořit pole záporné délky způsobí chybu. Vytvoření pole lze spojit s inicializací. V takovém případě se nepoužívá new:

```
int[]numbers = { 3, 5, 6, 7};
```

Velikost pole je dána počtem hodnot ve složených závorkách. Pole může být parametrem metody a může být i návratovým typem.

7.2. Vícerozměrná pole

Doposud jsme pro určení prvku v poli používali jeden index. Takovému poli říkáme jednorozměrné. Java umožňuje deklarovat a vytvářet i vícerozměrná pole. Např. deklarace dvojrozměrného pole položek typu int vypadá takto:

```
int[][] p;
```

Vícerozměrné pole v Javě je pole polí. Proměnná p je reference na pole, jehož prvky jsou jednorozměrná pole celých čísel. Pole vytvoříme pomocí klíčového slova new:

```
p = new int[2][3];
```

K prvkům pole přistupujeme pomocí indexů:

```
p[0][1] = 1;
```

Počet prvků pole je v proměnné length daného pole.

```
System.out.printf( "pole p má d řádkůn", p.length );
System.out.printf( "první řádek má d sloupcůn", p[0].length );
```

Při práci s vícerozměrnými poli používáme obvykle vnořené cykly for. Např. inicializaci prvků pole p na hodnotu 1 můžeme provést takto:

```
for( int i = 0; i < p.length; i++ ) {
  for( int j = 0; j < p[i].length; j++ ) {
       p[i][j] = 1;
  }
}
```

Vícerozměrná pole je možné vytvářet postupně. Dílčí pole pak mohou mít různý počet prvků. Dvojrozměrné pole tedy nemusí být nutně obdélníkové.

8. TŘÍDY

Třída je forma, která popisuje jednoznačně ohraničený soubor (typů) dat a operací nad nimi. Pomocí třídy pak vytváříme instance, jednotlivé objekty, které obsahují samotná data (nad kterými jsou volány příslušné operace).

Příklad: Mějme třídu Auto, tato třída popisuje, že auto má značku, typ, stáří a najeto a obsahuje operaci informaceOVozidle(), která vypíše zadané značku typ, stáří a nejeto . Pomocí této šablony – třídy – pak vytváříme jednotlivé instance, v reálném životě bychom je popsali jako konkrétní vozidla.

Každé nově vytvořené instanci (vozidlu) v programu přiřadíme data (značka, typ, stáří a najeto). Když později zavoláme operaci informaceOVozidle () nad tímto objektem (instancí), tak nám vypíše hlášku specifickou pro dané vozidlo.

8.1. Deklarace třídy

Pro deklaraci třídy používáme klíčové slovo class, před kterým je specifikátor přístupu, a za kterým následuje název třídy. Samotné tělo třídy je uzavřeno v bloku (složené závorky). Pokud se název skládá z více slov, zpravidla první písmeno každého slova píšeme velké (např. VypisInfo). Nepoužíváme podtržítka. Ve třídě můžeme deklarovat proměnné a metody. Proměnné i metody mohou být buď statické, nebo instanční.

```
class Cat {
  int weight;  // instance atribut
  int age;
  void showInfo() {  // Instance method
       System.out.println( info );
  }
}
```

Od dané třídy můžeme vytvářet instance (říkáme jim též objekty, angl. instances či objects). Třída je šablona, která říká, jak budou objekty vypadat, tj. jaké budou mít atributy a metody. V našem případě bude mít každá instance třídy Kočka dvě proměnné typu int. Deklarací proměnné typu Kočka zavedeme proměnnou, do níž můžeme uložit referenci (odkaz) na instanci třídy MojePrvni.

```
Cat  v;
```

Protože proměnné typu třída odkazují na objekty, nazývají se referenční proměnné. Každá referenční proměnná zabírá v paměti stejný prostor: 32 bitů v 32-bitové JVM a obvykle 64 bitů v 64-bitové JVM. Naproti tomu objekty různých typů mají zpravidla různou velikost. Velikost objektu je dána jeho atributy. Objekty vytváříme pomocí klíčového slova new:

```
V = new Cat();
```

Po provedení tohoto příkazu bude proměnná v obsahovat odkaz na instanci třídy Kocka. K atributům a metodám přistupujeme pomocí tečky. Volání metody zapisujeme pomocí jména metody a kulatých závorek:

```
v.info = 1;
v. showInfo ();
```

Od jedné třídy můžeme vytvořit libovolné množství instancí. Tyto instance jsou na sobě nezávislé.

```
Cat v2 = new Cat();
v2. showInfo = 2;
v2. showInfo ();
```

Instanční metody slouží k provádění operací nad objekty daného typu. Např. ve třídě MojePrvni můžeme deklarovat metodu, která zvýší hodnotu atributu x o 1:

```
class My {
  int x;
  void IncreaseA () {
        a++;
  }
}
```

Instanční metody, stejně jako metody třídní, mohou mít parametry a vracet hodnotu. Parametry a návratovou hodnotu stanovíme v deklaraci metody.

```
class My {
  int x;
  // Adds dx to x and returns a new x value
  int moveX( int dx ) {
       x += dx;
     return x;
  }
}
```

9. SPECIFIKÁTORY PŘÍSTUPU

Pro specifikace práv přístupu k jednotlivým třídám, jejich operacím a proměnným používáme specifikátory přístupu. Jejich význam je především v zakrývání implementačních detailů, které nemá (nesmí) uživatel vidět a případně je používat.

Public	Z jakékoliv třídy.
Private	Pouze uvnitř dané třídy, žádný přístup z vnějšku.
Protected	Z jakékoliv třídy stejného balíku, případně z potomka třídy kdekoliv.
žádný (package friendly)	Z kterékoliv třídy stejného balíku.

9.1. Konstruktory

Vytvoříme-li novou instanci třídy Kočka, budou všechny její instanční proměnné inicializovány na nulové hodnoty. Pokud budeme chtít u nově vytvořené kočky zadat její stáří, musíme použít speciální metodu: konstruktor, která se jmenuje stejně jako třída a u níž se neuvádí typ návratové hodnoty. V konstruktoru můžeme nastavit počáteční hodnoty instančních proměnných.

Jako příklad máme zaměstnance, který má svůj věk a plat. Obsahuje dále metodu IntroduceYourself.

```
class Employee {
  public Employee (int age, int wage) {
     this.age = age;
     this.wage = wage;
  }
  private int age = 1;
  public int getAge () { return age; }
  public void setAge(int age) { this. age = age; }
  private int wage = 1;

  public int getWage() { return wage; }
  public void setWage(int wage) { this.wage = wage; }
  public void introduceYourself(){
       System.out.println("My age a wage are "
       + age + "years"+ wage + "Euros");
  }
  public static void main(String[] args) {
    Employee employee = new employee (30,100);
  }
}
```

9.2. Dědičnost

Pomocí dědičnosti můžeme vytvářet hierarchie tříd, ve kterých můžeme o libovolném uzlu říct, že je speciálním případem libovolného ze svých předků. V normálním světě bychom řekli, že židle je typem nábytku, letadlo je typem dopravního prostředku a dopravní prostředek je typem stroje. Pozor ale nemůžeme dědit židli od zvířat, protože mají také 4 nohy!

V Javě pro vytvoření podtypu v hlavičce třídy ihned po jejím názvu uvádíme klíčové slovo extends a název rozšiřované třídy. Takto vytvořená třída zdědí všechny nesoukromé (včetně package friendly, je-li rozšiřující třída ve stejném balíčku) metody a třídní proměnné předka, které mohou být znova deklarovány a překryty.

Naopak třída nezdědí soukromé a statické metody svého předchůdce, protože ty se vztahují pouze ke konkrétní třídě předka. Koncové (final) metody označené jako potomek, třída zdědí, ale nemůže je překrýt. Každý objekt potomka můžeme přetypovat na předka.

9.3. Super

Při voláních metod podtypů často narazíme na to, že nechceme celou metodu překrýt, pouze k ní chceme přidat další funkcionalitu. V tento okamžik můžeme zavolat super.jmenoMetody(), čímž zavoláme funkcionalitu předka. Také můžeme volat konstruktor předka voláním super() – toto volání musí být v rámci konstruktoru potomka uvedeno nejdříve.

Jako příklad jsme vytvořili třídu ředitel, která je odvozena od třídy zaměstananec. Ovšem to neznamená, že by tato třída měla automaticky přístup ke všem soukromým proměnným a metodám původní třídy. Viz. Specifikátory přístupu.

Volání konstruktoru nadřazené třídy se provádí pomocí klíčového slova super a musí být prvním příkazem v těle konstruktoru. Pokud, že konstruktor nadřazené třídy nezavoláme, zařadí se do programu jeho volání automaticky - v takovém případě se volá tzv. implicitní konstruktor, tedy konstruktor bez parametrů.

```
class Director extends Employee {
  public Director(int age, int wage)
  {
        super(age,wage);
  }
  public static void main(String[] args) {
        Employee k = new Director(30, 50);
  }
}
```

10. POLYMORFISMUS

Můžeme jakoukoliv metodu překrýt v potomkovi vlastní implementací. Když nad daným objektem tuto metodu zavoláme, dojde vždy k vykonání překrývajícího kódu. A to bez ohledu na to, jestli k metodě přistupujeme pomocí reference na objekt předka nebo na objekt potomka (jehož třída obsahuje ono překrytí).

Této vlastnosti je dosaženo pomocí pozdní vazby (late binding), kdy je typ objektu, na němž bude metoda volána, rozhodnut až za běhu programu, nikoliv během jeho kompilace.

Toto ovšem platí pouze pro překrývání metod, nikoliv pro jejich přetěžování. Pokud budeme mít na daném objektu dvě metody, které se budou lišit pouze parametrem (jedna pro předka, druhá pro potomka), tak bude volána vždy ta metoda, která má v parametru stejný typ, jako je aktuální reference na objekt. Volání přetížených metod je totiž rozhodováno v době překladu, kdy ještě nemusí být jasné, jestli bude reference ukazovat na objekt předka nebo potomka.