Úvod do jazyka Java, procedurální programování

1

Úvod

Programovací jazyk Java patří mezi nejpopulárnější programovací jazyky současnosti. Za jeho popularitou stojí zejména jednoduchost a srozumitelnost, dostupnost praktických nástrojů usnadňujících programování (vývojových prostředí) a velké množství tzv. knihoven, které nabízí již hotová řešení problémů, na které může programátor narazit. Ty umožňují programátorovi soustředit se pouze na konkrétní problém, který má řešit, a mohou mu tak ušetřit mnoho času.

Poznámka 1 *V posledních letech vzniklo několik programovavích jazyků, zmiňme třeba Kotlin nebo Scala, které vychází z jazyka a platformy Java, přidávají novou, stručnější, syntaxi nebo nové prvky jazyka pro snažší a rychlejší programování. To je bezesporu pozitivní vlastnost. Ve srovnání s těmito „moderními" jazyky může jazyk Java, se svou rozvleklou syntaxí a omezenější výbavou, působit příliš konzervativně, možná i zastarele. Není to však nezbytně ke škodě jazyka. U praktických aplikací se nemalé množství práce soustředí na údržbu aplikací, tzn. opravu chyb, přidávání nových vlastností nebo propojování s jinými aplikacemi. Pokud na projektu pracuje více lidí nebo se tým programátorů mění, je jednoduchý jazyk s omezenější sadou vlastností, s explicitní syntaxí, naopak velkou výhodou. A právě proto jazyk Java najdeme na řadě míst, kde je potřeba vyvíjet a udržovat aplikace po dlouhá léta, typicky v bankovních aplikacích nebo podnikových informačních systémech. Dá se proto se přepodkládat, že poptávka po programátorech v Javě ješte dlouho potrvá.*

Jazyk Java se řadí mezi *objektově orientované* programovací jazyky. To znamená, že se snaží přistupovat k programu a datům ve stejném duchu, jako lidé běžně uvažují o předmětech, se kterými se denně setkávájí. Vezměme si jako příklad automobil. Automobil můžeme chápat jako jeden objekt, který má svou úlohu, typicky dopravit svého majitele z místa A do místa B. Pokud bychom se pustili do jeho bližšího zkoumání, zjistili bychom, že se skládá opět z dalších objektů, kde každý objekt plní svou úlohu. Například motor má za úkol roztáčet nápravu, kola mají zajistit pohyb po silnici, volant ovládá nápravu, atp. V podobném duchu můžeme pokračovat dál, do větší hloubky. Motor se skládá z dalších objektů jako jsou písty, hřídel, atd. Postupně bychom se dostali až na úroveň jednotlivých šroubků a matiček, které již dále rozložit nelze.

Klíčové v této metafoře je, že stejně jako z jednoduchých součástek vznikají složitější díly, až z nich vznikne automobil, v objektovém programování postupně z jednoduchých částí programu, které mají svůj jasně definovaný účel, vytváříme složitější celky, až vznikne výsledný program.

Naše metafora má ještě dva důležité důsledky. (i) Stejně jako jednotlivé součástky automobilu mohou navrhovat a vyrábět různé továrny, stačí, aby mezi nimi platila domluva, co která součástka má dělat a jak do sebe ve finálním výrobku zapadnou, tak jednotlivé části programu mohou vytvářet různí programátoři. Stačí se jen domluvit, co daný kus programu (objekt) má dělat a jak se bude používat. (ii) Podobně jako jednotlivé základní součástky nachází uplatnění ve více různých výrobcích, tak můžeme jednotlivé jednodušší části programu opakovaně používat v různých programech. Objektově orientované programovaní se proto hojně používá na velkých projektech, jelikož umožňuje rozdělit programování mezi jednotlivé programátory (týmy programátorů) a usnadňuje opakované použití již vytvořeného kódu.

Je potřeba zmínit, že jazyk Java není čistě objektový programovací jazyk. Z praktických důvodů kombinuje objektové a procedurální programování.[1] Proto předtím, než se dostaneme k samotnému objektovému programování, si ukážeme základní prvky procedurálního programování v Javě.

1 Procedurální programování v jazyce Java

Procedurální programování je postaveno na několika základních kamenech, kterými jsou proměnné, výrazy, příkazy, podmínky a cykly. Postupně si je představíme.

1.1 Proměnné

Proměnné nám umožňují uchovávat hodnoty, se kterými program pracuje. Každá proměnná má své jméno a přiřazené místo v paměti počítače, kde je hodnota proměnné uložena. V Javě musí být každá proměnná před svým použitím *deklarována*, tj. musí být v programu uveden název proměnné a musí být určeno, jakých hodnot může proměnná nabývat. To se označuje jako *typ* proměnné. Typická deklarace proměnné má některý z následujících tvarů:

```
<typ> <promenna>;
<typ> <promenna> = <hodnota> ;
final <typ> <promenna> = <hodnota> ;
```

V kódu deklarace proměnné vypadá následovně:

```
// deklarace proměnné bez počáteční hodnoty
int a;

// proměnná s počáteční hodnotou rovnou jedné
int b = 1;

// konstanta
final int c = 42;
```

[1] V Javě se dá setkat i s prvky funkcionálního programování. Tomu se budeme věnovat v některém z pozdějších seminářů.

V prvním případě deklarujeme proměnnou `a`, která může obsahovat pouze celá čísla (typ `int`). Proměnné `b` i `c` mohou nabývat taktéž celých čísel, avšak obě jsou inicializovány nějakou hodnotou. Při deklaraci je možné použít klíčové slovo `final`, které značí, že hodnota proměnné nepůjde změnit; bude to tedy konstanta.

Pokud lze typ proměnné odvodit z hodnoty přiřazené při její deklaraci, můžeme použít klíčové slovo `var` a nechat určení typu na překladači, jak ukazuje následující příklad.

```
// v obou případech se z hodnot odvodí typ int
var b = 1;
final var c = 42;
```

Poznámka 2 *Tento zápis může být pro programátora jednodušší, avšak my jej v našich příkladech nebudeme používat, aby bylo jasně vidět, jakého typu jednotlivé proměnné jsou.*

V našich ukázkách jsme vedle deklarace proměnných použili ještě komentáře, které se používají k vysvětlení, co daná část programu dělá, a jsou určeny, čistě pro programátory. Z pohledu provádění programu je cokoliv za `//` ignorováno. Vedle toho se dá setkat s víceřádkovými komentáři:

```
/*
 * Víceřádkový komentář.
 * Kde je vše od začátku do konce komentáře ignorováno.
 * Hvězdičky na jednotlivých řádcích nejsou nutnost ale pouhá dekorace.
 */
```

1.2 Primitivní datové typy

V jazyce Java rozlišujeme dva druhy datových typů (i) *primitivní datové typy* sloužící k reprezentaci čísel, pravdivostních hodnot a znaků a (2) *třídy* a *pole*, těm se budeme věnovat později. Zastavme se nyní u primitivních datových typů, jejichž kompletní výčet najdete v Tabulce 1. Charakteristickou vlastností hodnot primitivních datových typů je, že jsou nedělitelné a neměnitelné, např. hodnota `42` představuje právě jednu hodnotu, kterou dále nelze rozdělit ani nijak změnit, vždy to bude hodnota `42`.

Poznámka 3 *Pro reprezentaci celých čísel máme k dispozici hned čtyři datové typy, ale většinou se používá typ* `int`*, protože poskytuje dostatečný rozsah hodnot a práce s ním je v současných počítačích dostatečně efektivní.*

1.3 Výrazy

S primitivními hodnotami se pracuje pomocí operátorů,[2] které odpovídají běžným matematickým operacím a pomocí nichž je možné vytvářet složitější výrazy. Stejně jako v běžném matematickém zápisu můžeme použít i závorky.

[2]Tyto operátory vychází z jazyka C a lze je nalézt ve většině dnes běžně používaných jazyků.

typ	přípustné hodnoty (pozn.)
`byte`	celá čísla -2^7 až $2^7 - 1$ (-128 až 127)
`short`	celá čísla -2^{15} až $2^{15} - 1$ (-32.768 až 32.767)
`int`	celá čísla -2^{31} až $2^{31} - 1$ (cca $\pm$ 2,1 mld.)
`long`	celá čísla -2^{63} až $2^{63} - 1$
`boolean`	`true` nebo `false` (pravda, nepravda)
`float`	čísla s plovoucí řádovou (desetinnou) čárkou
`double`	čísla s plovoucí řád. čárkou (dvojitá přesnost)
`char`	znaky, zapisují se jako `'a'`

Tabulka 1: Přehled primitivních datových typů

```
int a = 40;

// do proměnné "b" uloží hodnotu proměnné "a"
// zvýšenou o 2
int b = a + 2;

// do proměnné "o" uloží hodnotu odpovídající
// obvodu obdélníku o stranách "a", "b"
int o = (a + b) * 2;

int r = 10;

// přiřadí do proměnné "s" přibližnou
// velikost plochy kruhu o poloměru "r"
double s = 3.14 * r * r;

// přiřadí do proměnné smallCircle hodnotu true,
// pokud je hodnota "s" menší než "b",
// jinak do proměnné přiřadí hodnotu false
boolean smallCircle = s < b;
```

Základní přehled operátorů je uveden v tabulkách 2, 3 a 4.

Poznámka 4 *Operátory typu* `x += y` *a* `x++` *v sobě kombinují aritmetickou operaci a přiřazení, odpovídají tedy kombinacím* `x = x + y` *a* `x = x + 1`.

Abyste si mohli ověřit již získané znalosti, ukážeme si asi nejjednodušší možný příklad programu v Javě. Tento program na textový výstup vypíše text: `"Prvni program!"`.[3]

[3]Pro první seznámení s jazykem Java můžeme využít i terminálový nástroj `jshell`, který nám umožňuje zadávat výrazy v jazyce Java a výsledné hodnoty vypisuje na terminál. Jinými slovy umožňuje nám pracovat s jazykem Java formou Read-Eval-Print-Loop.

operátor	význam
`x = y`	do proměnné `x` přiřadí hodnotu `y`
`x + y`	součet dvou číselných hodnot nebo řetězců
`x - y`	rozdíl dvou číselných hodnot
`x * y`	součin dvou číselných hodnot
`x / y`	podíl dvou číselných hodnot
`x % y`	zbytek po celočíselném dělení
`x += y`	přičte k proměnné `x` hodnotu `y` a vrátí novou hodnotu (podobně `x -= y`, `x *= y`, atd.)

Tabulka 2: Binární operátory

operátor	význam
`-x`	opačná hodnota číselné hodnoty `x`
`!x`	negace pravdivostní hodnoty `x` (pro typ `boolean`)
`x++`	zvýší hodnotu proměnné `x` o 1 a vrátí novou hodnotu (analogicky `x--` sníží hodnotu o 1)
`++x`	zvýší hodnotu proměnné `x` o 1 a vrátí původní hodnotu (analogicky operátor `--x` sníží hodnotu o 1)

Tabulka 3: Unární operátory

operátor	význam
`x == y`	`true`, pokud je `x` shodné s `y`, jinak `false`
`x != y`	`true`, pokud je `x` různé od `y`, jinak `false`
`x < y`	`true`, pokud je `x` menší `y`, jinak `false`
`x <= y`	`true`, pokud je `x` menší nebo rovno `y`, jinak `false`
`x > y`	`true`, pokud je `x` větší `y`, jinak `false`
`x >= y`	`true`, pokud je `x` větší nebo rovno `y`, jinak `false`

Tabulka 4: Operátory porovnání

```
1 public class MyFirstJavaProgram {
2   public static void main(String[] args) {
3     System.out.println("Prvni program!");
4   }
5 }
```

V tomto příkladu jsou věci, které jsme si doposud nevysvětlili. Časem se k nim dostaneme. Teď je pro nás klíčový řádek číslo 3, který se postará o vypsání hodnoty na výstup programu. Ostatní řádky můžeme pro tento okamžik považovat za informace nutné pro spouštění programu.[4]

Pro uložení textu se v Javě používá datový typ `String`. Hodnotami tohoto typu jsou textové řetězce, které se v kódu programu zapisují do dvojitých uvozovek, tj. `"Textovy retezec"`.[5] Pro vypsání řetězce či jiné hodnoty se používá obrat `System.out.println(hodnota)`.[6]

Drobnou změnou programu, nahrazením třetího řádku jinou posloupností instrukcí, si můžete vyzkoušet, jak se chovají jednotlivé operátory, viz následující příklad.

```
1 public class MyFirstJavaProgram {
2   public static void main(String[] args) {
3     int x = 10;
4     int y = 20;
5     int o = (x + y) * 2;
6     System.out.println("Obdelnik o stranach "  + x + " a " + y + " ma obvod:");
7     System.out.println(o);
8   }
9 }
```

Všimněte si, že na řádku 6 se operátor + používá ke spojení řetězců a číselných hodnot, do jednoho řetězce, který je pak vypsán na výstup programu.

Poznámka 5 *Programy v Javě jsou sestaveny z tzv. tříd, podrobněji o nich budeme mluvit v příštím semináři. Zdrojové kódy tříd jsou uloženy v souborech s příponou* `*.java` *a název souboru musí být shodný s názvem třídy. Ten najdeme za klíčovými slovy* `public class`*. V našem případě máme tedy třídu* `MyFirstJavaProgram`*, která musí být uložena v souboru* `MyFirstJavaProgram.java`*.*

Poznámka 6 *Programy v Javě lze vytvářet v libovolném textovém editoru, a pak je přeložit a spustit pomocí nástrojů z Java Development Kitu (JDK), který lze stáhnout z webu společnosti Oracle. Tento postup ale není nejpohodlnější, a proto se v praxi používá jen ve speciálních případech. Většina programů v Javě vzniká v tzv. vývojových prostředích,*

[4] V případě nástroje `jshell` nám stačí do něj zadat jen řádek číslo 3.

[5] Java podporuje i víceřádkové řetězce (tzv. *textové bloky*), ty se zapisují do tří dvojitých uvozovek následovně: `"""` (zalomení řádku) ⟨ jednotlivé řádky ⟩ `"""`.

[6] K tomu, co vlastně znamená `System.out.println`, je potřeba mít hlubší znalosti Javy. Postupně se k nim dobereme, pro tento okamžik použití tohoto obratu budeme brát jako dané.

která v sobě zahrnují nejen textový editor a překladač, ale i další nástroje usnadňující programování, např. debugger pro krokování výpočtu, nápovědu, správu zdrojových kódů, atd. Pro jazyk Java jsou k dispozici zcela zdarma dvě plnohodnotná vývojová prostředí NetBeans a Eclipse. Pro potřeby tohoto semináře je jedno, pro které z nich se rozhodnete. V závěru tohoto semináře najdete několik poznámek, které se věnují základní práci s těmito vývojovými prostředími. Vedle toho je mezi programátory oblíbené i vývojové prostředí IntelliJ IDEA, které má však výrazně komplikovanější licenční podmínky.

Úkoly k procvičení 1 *Upravte výše zmíněný program, aby pro proměnné* `h`, `g` *a* `t` *vypočítal hodnotu proměnné* `y`, *která je daná vzorcem*

$$y = h - \frac{1}{2}gt^2.$$

Program vypíše hodnoty všech proměnných. Předpokládejte, že `g` *je konstanta rovna* $9,80665$.

1.4 Řízení výpočtu

Ukázali jsme si, jak vytvořit jednoduchý program, který se skládá z deklarace proměnných, přiřazení hodnot do těchto proměnných a z následného výpisu hodnot na textový výstup programu. V reálných programech ale často narazíte na situace, kdy je potřeba provést rozhodnutí typu *pokud něco, tak udělej něco* nebo *pokud je splněna podmínka, opakuj posloupnost příkazů*.

1.4.1 Podmínky

Pro rozhodnutí typu *pokud něco, tak udělej něco* máme v Javě konstrukci `if`, která má následující tvar.

```
if (podminka) {
  // kód, který se provede,
  // pokud hodnota výrazu "podminka" je true
}
```

Případně můžeme použít klíčové slovo `else` a uvést, co se má stát, pokud podmínka splněna není.

```
if (podminka) {
  // kód, který se provede,
  // pokud hodnota výrazu "podminka" je true
} else {
  // kód, který se provede,
  // pokud hodnota výrazu "podminka" je false
}
```

Pokud bychom měli například proměnnou `x` a zajímala by nás její absolutní hodnota, mohli bychom to s pomocí `if` vyjádřit jako:

```
if (x < 0) {
  abs = -x;
} else {
  abs = x;
}
```

Další možností, jak větvit program je konstrukce `switch-case`, která má dvě podoby. Jednak je to forma příkazu, která kopíruje syntaxi a semantiku jazyka C:

```
switch (výraz) {
case hodnota1: /* 1. větev výpočtu */ break;
case hodnota2: /* 2. větev výpočtu */ break;
// ...
default:
  /* zbývající případy */
}
```

Nejdříve se vyhodnotí zadaný výraz, a podle výsledné hodnoty se hledá odpovídající větev výpočtu, která se provede. Pokud se taková hodnota nenajde, použije se větev označena `default`. Pozor, pokud v dané větvi výpočtu není uvedený příkaz `break` nebo `return`, provádí se i kód dalších větví výpočtu, dokud program nenarazí na jeden z těchto dvou příkazů.

Příklad použití:

```
int number = 1;
switch (number) {
case 1: System.out.println("jedna"); break;
case 2: System.out.println("dvě"); break;
// ...
default:
  System.out.println("hodně"); break;
}
```

Všimněte si, že v jednotlivých větvích výpočtu provádíme nějakou akci, konkrétně vypisujeme řetězec na výstup.

Vedle toho existuje varianta `switch-case`, která představuje výraz, který se vyhodnotí na konkrétní hodnotu. Syntaxe je podobná, s tím rozdílem že místo dvojtečky je použita kombinace `->` a výrazy nejsou zakončeny `break`. Předchozí příklad bychom s pomocí této varianty `switch-case` mohli přepsat následovně:

```
int number = 1;
String inWords = switch (number) {
case 1 -> "jedna";
```

```
case 2 -> "dvě";
// ...
default -> "hodně";
};
System.out.println(inWords);
```

Poznámka 7 *Pro rozhodování, která větev výpočtu se použije, můžeme použít řetězce nebo celočíselné hodnoty, např.* `int`*. K dispozici jsou i další možnosti, se kterými se seznámíme v dalších seminářích.*

Poznámka 8 *V případě, že použijeme výraz* `switch-case` *je nutné uvést všechny možnosti, které mohou nastat. To v případě příkazu* `switch-case` *není nutné, ale pouze velmi vhodné.*

1.4.2 Cykly

Abychom mohli v programu vyjádřit opakující se výpočet, nabízí nám jazyk Java hned tři typy tzv. *cyklů*[7] – `for`, `while` a `do-while`. Každý cyklus obsahuje podmínku (pravdivostní výraz) a tělo – seznam příkazů, které se mají provádět, pokud je podmínka splněna.

Asi nejjednodušší je cyklus `while`, který se zapisuje:

```
while (podminka) {
  // kód, který se má opakovaně provést
}
```

Příkazy v těle cyklu se provádí, dokud je podmínka splněna. Následující příklad ukazuje, jak s pomocí `while` vypsat čísla od 1 do 10. Na začátku si do proměnné `i` uložíme hodnotu 1, v podmínce cyklu (za `while`) se podíváme, zda je hodnota menší nebo rovna 10, a pokud ano, vypíšeme hodnotu `i`. Následně zvýšíme hodnotu proměnné o 1 a zopakujeme test. Pokud hodnota proměnné `i` bude větší než deset, tělo cyklu se neprovede a program pokračuje dalším příkazem.

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

Úkoly k procvičení 2 *Upravte ukázkový program tak, že vypíše všechna sudá čísla v intervalu od 1 do 20.*

Tento příklad ukazuje jedno z typických použití cyklů. Na začátku nastavíme proměnné nějakou hodnotu, a dokud platí daná podmínka, provede se tělo cyklu a hodnotu proměnné změníme. Pokud stále platí podmínka, cyklus se opakuje. Pro jednodušší zápis této konstrukce má Java cyklus typu `for`, který má tvar:

[7] někdy se můžete setkat s označením *smyčka*

```
for (inicializace; podminka; krok) {
  // tělo cyklu
}
```

Náš příklad s výpisem čísel bychom mohli tedy zestručnit:

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

Poznamenejme, že je-li proměnná deklarována v inicializační části `for`, je dostupná pouze v rámci tohoto cyklu, tj. v podmínce, v části krok a v těle cyklu.

Posledním typem cyklu, který jazyk Java nabízí, je `do-while`, který se od dříve zmíněného cyklu `while` liší tím, že tělo cyklu je vždy provedeno alespoň jednou, a až poté je ověřena podmínka, která udává, jestli se má výpočet v těle cyklu opakovat. V programu se tento typ cyklu zapisuje:

```
do {
  // tělo cyklu
} while (podminka);
```

Úkoly k procvičení 3 *Předpokládejme, že máme dvě proměnné* `a` *a* `b`*, které obsahují celá kladná čísla představující délky stran obdélníku. Napište program, který vykreslí obdélník pomocí znaků* `'-'` *a* `'|'`*. Například pro* `a = 4` *a* `b = 2` *by výstup měl vypadat následovně:*

```
------
|    |
|    |
------
```

Nápověda: Pokud chcete vypsat znak bez zalomení řádku, můžete použít `System.out.print('x');`*, případně pokud chcete nechat řádek zalomit, můžete použít speciální znak* `\n`*.*

Pokud kód, který se má provést v těle cyklu `while`, `for`, nebo jedné z větví `if`, obsahuje právě jeden příkaz, můžeme složené závorky `{ }` vypustit. V praxi se ale vypouštění závorek příliš nedoporučuje, jelikož dělají kód srozumitelnější a čitelnější.

Doporučená literatura

1. The Java™ Tutorials: Getting Started
2. The Java™ Tutorials: Learning the Java Language

A Základní práce s vývojovým prostředím

Vývojová prostředí (někdy též IDE, z anglického Integrated Development Environment) jsou komplexní nástroje, které mají usnadnit programátorovi rutinní úkoly spojené s tvorbou programu. Na první pohled ale mohou působit složitě a nepřístupně, proto v následujících odstavcích najdete postupy, které vám umožní vytvořit program a spustit jej, abyste si mohli vyzkoušet získané znalosti. Náš popis omezíme na volně dostupná vývojová prostředí NetBeans a Eclipse. Řada programátorů používá ještě vývojové prostředí IntelliJ Idea, které se ovládá velmi podobně.

A.1 Organizace zdrojových kódů

Obě vývojová prostředí pracují s pojmy pracovní plocha (*workspace*) a projekt (*project*). Workspace lze chápat jako jeden adresář, ve kterém existuje několik projektů, a projekty představují jednotlivé ucelené programy. Pro potřeby tohoto semináře bude nejvhodnější, když si vytvoříte samostatnou pracovní plochu, a pro každý díl semináře si vytvoříte nový projekt.

V rámci projektu obvykle existují další podadresáře. Typicky je to podadresář `src` se zdrojovými kódy tříd (soubory s příponou `*.java`) a adresář `bin`, který obsahuje přeložený kód tříd do binární podoby určené ke spuštění (soubory s příponou `*.class`).

A.2 Eclipse: Vytvoření projektu

1. Z menu vyberte *File* → *New* → *Java Project*;
2. zadejte název projektu;
3. stiskem tlačítka *Finish* projekt vytvoříte.

A.3 Eclipse: Vytvoření třídy

1. Z menu vyberte *File* → *New* → *Class*;
2. zadejte jméno třídy (např. `MyFirstJavaProgram`);
3. stiskem tlačítka *Finish* třídu vytvoříte.

A.4 Eclipse: Spuštění programu

Program spustíte tak, že se přesunete do třídy, která obsahuje metodu `public static void main(String[] args)`, aby vývojové prostředí vědělo, odkud začít provádět kód, a z menu vyberte volbou *Run* → *Run*. Při opětovném spuštění už by nemelo být nutné nacházet se ve třídě s metodou `main`.

A.5 Netbeans: Vytvoření projektu

1. Z menu vyberte *File* → *New Project*;
2. zvolte *Java Application* a pokračujte *Next*;
3. zde si zvolte název projektu a případně adresáře, kam projekt umístit;
4. políčko *Create Main Class* nastavte tak, aby nebylo zaškrtnuté;
5. projekt vytvoříte stiskem tlačítka *Finish*.

A.6 Netbeans: Vytvoření třídy

1. Z menu vyberte *File* → *New File*;
2. zvolte *Java Class* a pokračujte *Next*;
3. zadejte název třídy (např. `MyFirstJavaProgram`), případně projekt, kam má třída patřit;
4. stiskem tlačítka *Finish* třídu vytvoříte.

A.7 Netbeans: Spuštění programu

Program spustíte z menu volbou *Run* → *Run project*, případně pomocí příslušné ikony v horní liště. Je možné, že budete vyzváni, ať určíte třídu, která obsahuje metodu `public static void main(String[] args)`, aby vývojové prostředí vědělo, odkud začít provádět kód.

A.8 Vybrané tipy pro práci s IDE

Doplňování kódu. Psaní programu vám může výrazně usnadnit automatické doplňování textu, které najdete pod kombinací kláves *Ctrl+mezera*. Stačí zadat prvních pár znaků z názvu metody či proměnné a *Ctrl+mezera* a vývojové prostředí vám nabídne vhodné doplnění.

Formátování kódu. Aby byl program dobře čitelný, je dobré věnovat péči formátování kódu, zejména odsazování do bloků. Abyste se mohli věnovat jen programování a nestarat se o formátování, máte k dispozici volbu *Source* → *Format*, která se postará o automatické naformátování kódu a vhodné odsazení.

Procházení kódem. U větších projektů nastane po čase problém s orientací ve zdrojových kódech. Když kliknete myší na proměnnou nebo název metody a držíte současně klávesu *Ctrl*, budete přesunuti na místo, kde byla proměnná nebo metoda deklarována.

Přejmenování. Často se stane, že název proměnné, metody či třídy nezvolíte napoprvé úplně šťastně. Přejmenovávat všechny výskyty nevhodně pojmenované proměnné nebo metody může být nepohodlné a časově náročné.

Práci vám může ušetřit volba *pravé tlačítko myši nad proměnnou/metodou/třídou* → *Refactor* → *Rename*, která se postará o systematické přejmenování proměnné, metody nebo třídy v celém projektu.

Komentáře. Pokud máte kód, se kterým nechcete pracovat, např. je zkušební, můžete jej zakomentovat pomocí *Ctrl+/*, případně vrátit zpět stisknutím stejných kláves.