KAPITOLA 2

Proměnné a datové typy

V této kapitole:
- Primitivní datové typy
- Proměnné
- Opakování

Mezi základní dovednosti každého programátora bezesporu patří dobrá znalost datových typů. Ta vám umožní efektivní využívání proměnných a v neposlední řadě vyvarování se zdlouhavého hledání chyb. Nesprávné použití datových typů proměnných totiž může vést i k na první pohled překvapivým výsledkům. Ale co vlastně znamenají ty datové typy, o kterých se pořád mluví? Jak a k čemu se používají proměnné? To se dozvíte v této kapitole.

Primitivní datové typy

Datové typy především usnadňují práci programátorům. Určují, kolik místa v paměti bude daná proměnná zabírat, abyste se o to nemuseli starat sami. Java je navíc jazyk se **silnou typovou kontrolou**, při kompilaci programu se tedy projdou proměnné a zkontroluje se, zda jsou v nich uložená data toho správného typu. To může ušetřit cenný čas při hledání chyb. Moderní vývojová prostředí vás navíc na případné chyby umí upozornit už při psaní díky průběžnému kompilování kódu.

Primitivní neboli základní datové typy lze rozdělit na číselné, znakové a logické. Samy názvy dobře napovídají, jaká data tyto typy zastupují.

! **Důležité:** Tyto datové typy se nazývají primitivní proto, že je Java nechápe jako objekty. O objektech se více dozvíte v dalších kapitolách. Pro teď stačí, když si zapamatujete, že každý z primitivních typů má svou obalující třídu a v případě shodnosti názvu je navzájem odlišíte pomocí počátečního písmena. Primitivní typy vždy musí začínat malým písmenem a obalující třídy písmenem velkým.

Číselné datové typy

Pro správné porozumění rozdílům mezi jednotlivými číselnými typy je potřeba si zopakovat některé základní znalosti ze střední školy. Čísla lze rozdělit do několika množin, které se navzájem překrývají, a tudíž čísla z nižší množiny vždy zároveň patří i do všech

vyšších množin. Nás budou zajímat dvě základní množiny, a to **celá čísla** a **reálná čísla**. Důležitým faktorem je také velikost každého datového typu.

Pro zjednodušení si můžete paměť programu představit jako přidělenou zásuvku s mnoha přihrádkami. Každá přihrádka má velikost jeden bit (bit je základní a nejmenší počítačová jednotka) a je potřeba vědět, kolik přihrádek daný typ zabere.

Byte

Nejmenší typ, do kterého je možné ukládat pouze **celá čísla**. V paměti zabírá velikost **8 bitů** neboli právě jeden bajt (anglicky byte). Pokud již máte nějaké povědomí o základní reprezentaci čísel v počítači, jistě tušíte, že do tohoto typu lze uložit pouze ta čísla, která je možné zapsat pomocí 8 bitů. V praxi to znamená čísla v rozsahu -128 až +127. Označuje se celým názvem `byte`.

Short

Druhý nejmenší číselný typ, do kterého lze opět ukládat pouze **celá čísla**. Zabírá v paměti **16 bitů** a moc často se s ním zřejmě nesetkáte. Lze do něj uložit čísla v rozsahu -32 768 až +32 767. Stejně jako `byte` se označuje celým svým názvem `short`.

Integer

Číselný typ, se kterým se setkáte zřejmě nejčastěji, je **integer**. Stále slouží pouze k ukládání **celých čísel**, ale díky velikosti **32 bitů** již jeho rozsah stačí na velké množství potřebných výpočtů. Můžete do něj ukládat čísla v rozsahu –2 147 438 648 až +2 147 438 647. Na rozdíl od výše uvedených typů se neoznačuje celým názvem, ale zkratkou `int`.

Long

Jedná se o poslední a největší typ pro ukládání **celých čísel**. Jeho velikost je **64 bitů**, což v naprosté většině případů postačí i pro vaše nejsložitější výpočty. Má rozsah –9 223 372 036 854 775 808 až +9 223 372 036 854 807 a znovu se označuje celým svým názvem `long`. Všechna ta čísla si samozřejmě není nutné přesně pamatovat, u všech uvedených rozsahů je důležité jen přibližně řádově tušit, jak velké číslo se do nich ještě vejde. Pokud budete potřebovat, najdete přesné hodnoty také v níže uvedené tabulce.

Tabulka 2.1 Celočíselné datové typy

Označení typu	Velikost	Nejmenší možné číslo	Největší možné číslo
`byte`	8 bitů	-128	+127
`short`	16 bitů	-32 768	+32 767
`int`	32 bitů	-2 147 438 648	+2 147 438 648
`long`	64 bitů	-9 223 372 036 854 775 808	+ 9 223 372 036 854 775 807

Důležité: Možná vás už napadla otázka, co se stane, pokud se číslo svým rozsahem do zvoleného typu prostě nevejde. V tom případě dojde k takzvanému přetečení a výsledek operace nebude na první pohled vůbec dávat smysl. Například při sečtení dvou příliš velkých kladných čísel můžete dostat jako výsledek číslo záporné. Je potřeba si na podobné věci dávat pozor a volit datové typy pečlivě a opatrně.

Float

Nyní jsme se dostali k prvnímu typu pro reprezentaci **reálných čísel** neboli čísel s desetinou částí. Java ukládá reálná čísla podle mezinárodního standardu IEEE 754, to znamená, že se zobrazují stejně jako v jiných jazycích používajících tento standard. Typ **float** ukládá reálná čísla o velikosti **32 bitů** a používá se, pokud potřebujete ušetřit místo. Není nutné přesně vědět jeho rozsah, maximální uložená hodnota může být přibližně $3{,}4x10^{38}$. Označuje se celým svým názvem `float`. Reálné typy se v programech zapisují s desetinnou tečkou.

Double

Obvykle základní používaný typ pro ukládání **reálných čísel** je typ **double**. Má takzvanou dvojitou přesnost a zabírá v paměti prostor **64 bitů**. Rozhodně by se vám nemělo podařit přesáhnout rozsah tohoto typu. Opět se označuje celým svým názvem `double`. Stejně jako `float` by tento typ nikdy neměl být používán pro ukládání hodnot, u kterých je nutná absolutní přesnost, jako jsou například peníze. U běžných výpočtů (téměř všech) ale ztrátu přesnosti vůbec nepoznáte.

Poznámka: Ačkoli vás možná po upozornění na možnost přetečení napadlo, že stačí všechna čísla reprezentovat největšími typy, není to vhodné řešení. Pokud se celé číslo, které potřebujete použít, nevejde ani do rozsahu typu `long`, je sice na místě nahradit ho typem `double`, to je ale krajní řešení. Je dobrým zvykem používat nejmenší možný typ kvůli úspoře místa a v případě reálných typů i kvůli jinému způsobu jejich ukládání. Celá čísla jsou z principu ukládána přesně, reálná čísla nejdou vždy precizně vyjádřit a může dojít ke ztrátě jejich přesnosti.

Logický datový typ

Tento typ představuje jeden bit informace, který může nabývat pouze dvou hodnot. Jeho velikost ale není přesně specifikována. Zastupuje logické konstanty používané ve výrazech: `true` – logická jednička neboli pravda a `false` – logická nula neboli nepravda. Používá se zejména v podmínkách a jiných testovacích výrazech. Označuje se anglickým výrazem `boolean` a lze mu přiřadit výše zmíněné dvě hodnoty, `true` a `false`.

Znakový datový typ

Znak je jediný typ zastupující text mezi primitivními datovými typy. Označuje se anglickou zkratkou `char` (ze slova character) a má velikost **16 bitů**. Představuje právě jeden

znak v kódování UNICODE, které Java vnitřně používá, jelikož u něj nedochází k problémům při používání národních znaků. Hodnoty tohoto typu musí být vždy uzavřeny do apostrofů a je více možností jejich zápisu:

- Právě jedním znakem zapsatelným na klávesnici, například: `'A'`, `'ž'`, `'8'`, `'%'`.
- Specifickým číselným kódem ve formátu `\uXXXX`. Tabulku těchto čísel naleznete snadno na internetu například po zadání „*unicode table*“ do vašeho vyhledávače. Tento způsob se většinou používá pro znaky, které nelze snadno zadat na klávesnici, například: `'\u0024'` je znak `$` a `'\u0126'` je znak `Ħ`.
- Použitím sekvence se zvláštním významem pro netisknutelné znaky nebo pro znaky, které mají v Javě zvláštní význam. Využijete je později v části o textových řetězcích. Nejčastěji používané sekvence najdete v následující tabulce.

Tabulka 2.2 Znakové sekvence se zvláštním významem

Sekvence	Význam
`'\n'`	Nový řádek – ve výpisu textu dojde na tomto místě k jeho zalomení, nezobrazuje se.
`'\t'`	Tabulátor – do textu se vloží znak tabulátoru, nezobrazuje se.
`'\b'`	Backspace – smaže ve výpisu předchozí znak, nezobrazuje se.
`'\r'`	Návrat na začátek řádky, nezobrazuje se.
`'\\'`	Zpětné lomítko, jako speciální znak ruší funkci ostatních speciálních znaků za nimi, které je potom možné vypsat jako text.
`'\''`	Apostrof, bez zpětného lomítka by ukončil sekvenci znaku a ten by se jevil jako prázdný.
`'\"'`	Uvozovky, bez zpětného lomítka by začínaly nebo ukončovaly textový řetězec.

Poznámka: Je možné, že se vám bude funkce sekvencí `'\n'` a `'\r'` jevit jako stejná. Rozdíl je daný použitým operačním systémem, každý operační systém používá jako znak konce řádku jinou sekvenci. Původně rozdíl vznikl podle zvyku z psacích strojů, kdy nový řádek pouze posunul papír nahoru, ale nepřesunul se na začátek, zatímco návrat na začátek řádku zase posunul psací hlavu doleva, ale neumožňoval přechod na nový řádek.

Proměnné

Teď už víte, jaká data je možné ukládat. Stále ale ještě nevíte kam. K ukládání dat slouží právě **proměnné**. Představují adresu místa, kde jsou v paměti uložená určitá data. Podle datového typu, který dané proměnné určíte, Java vyhradí potřebné místo v paměti a zapamatuje si jeho adresu. Protože není v lidských silách zapamatovat si přesnou adresu tak, jak ji používá počítač, mohou být proměnné téměř libovolně pojmenované. Jménu, které proměnné přidělíte, se říká **identifikátor** a v programu místo přesné adresy pracujete s ním.

Deklarace proměnné

Aby bylo možné proměnnou v programu použít, je nutné ji nejprve deklarovat neboli upozornit Javu na její přítomnost. Díky deklaraci proměnné Java vyhradí a pojmenuje místo v paměti, které pak můžete dále používat. Při deklaraci proměnné je tedy nutné mít vybrán její datový typ a název. Nejdříve zapíšete označení datového typu, za mezeru zapíšete identifikátor proměnné neboli vámi zvolené jméno a celý příkaz již bez mezer ukončíte středníkem.

Například zápis celočíselné proměnné pojmenované `cislo` by tedy vypadal takto:

```
int cislo;
```

Z jakých datových typů máte na výběr, bylo řečeno v předchozí části kapitoly. Jména proměnných si můžete zvolit téměř libovolně, i zde ovšem existují jistá omezení:

- jméno **nesmí začínat číslem**, ale jinde v názvu se čísla vyskytovat smí,
- jméno **nesmí obsahovat mezery**,
- jméno **nesmí tvořit hodnoty** `true`, `false` a `null`,
- jméno **musí být unikátní ve svém rozsahu platnosti** (bude vysvětleno dále v této kapitole),
- jménem **nesmí být některé z rezervovaných klíčových slov jazyka Java**, ačkoli obsaženo být může. Seznam rezervovaných klíčových slov najdete v tabulce níže.

Tabulka 2.3 Rezervovaná klíčová slova jazyka Java

`abstract`	`continue`	`for`	`new`	`switch`
`assert`	`default`	`goto`	`package`	`synchronized`
`boolean`	`do`	`if`	`private`	`this`
`break`	`double`	`implements`	`protected`	`throw`
`byte`	`else`	`import`	`public`	`throws`
`case`	`enum`	`instanceof`	`return`	`transient`
`catch`	`extends`	`int`	`short`	`try`
`char`	`final`	`interface`	`static`	`void`
`class`	`finally`	`long`	`strictfp`	`volatile`
`const`	`float`	`native`	`super`	`while`

Poznámka: Programátoři v Javě se poměrně důsledně drží konvencí v pojmenovávání proměnných, které byste měli dodržovat i vy. Alespoň pokud chcete, aby byly vaše programy pro ostatní srozumitelné. Názvy proměnných podle konvence vždy začínají malým písmenem, co nejlépe vystihují účel proměnné, a pokud obsahují více slov, každé další slovo začíná velkým písmenem. Příkladem takového názvu je třeba `vypoctenyObsahSteny`.

Při deklarování více proměnných můžete každou z nich zapsat na samostatný řádek. Jestliže jsou proměnné různých datových typů, nemáte ani jinou možnost. Pokud jsou ale proměnné stejného datového typu, stačí ho uvést jen na začátek řádku a za ním vyjmenovat všechny názvy oddělené čárkami. Teprve za posledním názvem ukončíte příkaz středníkem.

Například zápis několika celočíselných proměnných s názvy `cislo`, `obsah` a `obvod` by tedy vypadal takto:

```
int cislo, obsah, obvod;
```

Inicializace proměnné

Po deklaraci tedy už Java o proměnné ví a přidělila jí místo v paměti. Je čas do proměnné uložit nějaká data, tedy ji inicializovat. Opět se jedná o velice jednoduchý příkaz, který můžete zapsat buď samostatně a uložit data do již deklarované proměnné, nebo ho můžete spojit s deklarací do jednoho příkazu. K uložení dat do již deklarované proměnné tedy zapíšete její název a za znak rovnosti zadáte zvolenou hodnotu následovanou středníkem.

Například přidělení hodnoty `5` již deklarované proměnné `cislo` by vypadalo takto:

```
cislo = 5;
```

Jednoduché, že? Stejně snadné je i spojení deklarace a inicializace proměnné do jednoho příkazu. Před název proměnné z předchozího příkladu jen přidáte označení datového typu. Výsledek bude následující:

```
int cislo = 5;
```

Poznámka: O konvenci při pojmenovávání proměnných již řeč byla. Teď je vhodná chvíle dozvědět se něco o pojmenovávání konstant. Jestliže v programu deklarujete a inicializujete proměnnou, která se v jeho průběhu nebude nijak měnit (tedy konstantu), je podle konvence vhodné ji pojmenovat velkými písmeny. V případě víceslovného názvu jednotlivá slova odděluje znak podtržení. Příkladem správně pojmenované konstanty je třeba `CISLO_PI`.

Stejným způsobem můžete postupovat i v případě deklarace více proměnných stejného typu za sebou, za název každé proměnné jen přidáte znak rovnosti a přidělenou hodnotu. Například přidělení hodnot celočíselným proměnným `cislo`, `obsah` a `obvod` spojené s jejich deklarací by vypadalo takto:

```
int cislo = 5, obsah = 25, obvod = 20;
```

Přidělit hodnotu více proměnným v jednom příkazu je ale možné pouze při jejich deklaraci. Jestliže již jsou proměnné v programu deklarovány, musí jim být přiřazována hodnota vždy v samostatném příkazu zakončeném středníkem. Například deklarace a inicializace proměnné `obsah` a následná změna hodnoty by se zapsala takto:

```
int obsah = 25;
obsah = 30;
```

Důležité: Každou proměnnou je v programu nutné deklarovat (tedy uvést datový typ) pouze jednou. Přidělenou hodnotu můžete měnit libovolně, ovšem již bez opětovného uvádění datového typu. Pokud byste při přepisování hodnoty opakovaně uvedli i její datový typ (znovu ji deklarovali), Java si bude myslet, že definujete novou proměnnou stejného názvu, a ohlásí chybu.

Rozsah platnosti proměnné

Program v Javě je rozdělen do takzvaných bloků kódu. Každý blok je uzavřen do složených závorek `{}`. Tyto bloky mimo jiné určují, kde bude proměnná viditelná a kdy bude uvolněno místo v paměti, které proměnná zabírá. Všechny proměnné je možné použít pouze v bloku, ve kterém jsou deklarované, a v jeho vnořených blocích. Ve vnějších blocích nelze k těmto proměnným přistupovat.

Java má, k úlevě mnoha programátorů, automatickou správu paměti. Znamená to, že se sama stará o uvolnění vámi rezervované paměti. Ve chvíli, kdy se při vykonávání programu dostane na konec bloku, ve kterém byla proměnná deklarovaná, a ta by již nebyla z jiných míst programu přístupná, uvolní zabírané místo jiným datům.

Podívejme se na následující příklad průběhu programu, který není tvořen přímo kódem v jazyce Java, ale pouze příklady umístění bloků a proměnných:

```
{int cislo = 5;
/* Java vyhradila místo pro celočíselnou proměnnou
 * a uložila do něj hodnotu 5, proměnná cislo
 * je v tomto bloku viditelná */
    {cislo = 3;
    /* proměnná cislo je ve vnořeném bloku
     * také viditelná, hodnota 5 byla přepsána hodnotou 3 */
    char znak = 'x';
    /* Java vyhradila místo pro znak a uložila do něj
     * písmeno x, v tomto bloku lze použít
     * proměnné cislo a znak */
    } //Java uvolnila místo zabírané proměnnou znak
/* zde lze stále použít proměnnou cislo,
 * ale již nelze používat proměnnou znak */
char druhyZnak = 'a';
//Java vyhradila místo pro znak a uložila do něj písmeno a
    {int neco = 0;
    /* Java vyhradila místo pro celočíselnou proměnnou
     * a uložila do něj hodnotu 0 */
        {double realne = 2.3;
        /*Java vyhradila místo pro reálnou proměnnou
         * a uložila do něj hodnotu 2.3.
         * V tomto místě lze použít proměnné cislo,
```