Otázka číslo 24
Jazyk Java: proměnné, typy a operátory

Proměnné

Než začneme řešit datové typy, pojďme se shodnout na tom, co je to proměnná. Určitě znáte z matematiky proměnnou (např. x), do které jsme si mohli uložit nějakou hodnotu, nejčastěji číslo. Proměnná je v informatice naprosto to samé, je to místo v paměti počítače, kam si můžeme uložit nějaká data (jméno uživatele, aktuální čas nebo databázi článků). Toto místo má podle typu proměnné také vyhrazenou určitou velikost, kterou proměnná nesmí přesáhnout (např. číslo nesmí být větší než 2 147 483 647).

Proměnná má vždy nějaký **datový typ**, může to být číslo, znak, text a podobně, záleží na tom, k čemu ji chceme používat. Většinou musíme před prací s proměnnou tuto proměnnou nejdříve tzv. deklarovat, čili říci jazyku jak se bude jmenovat a jakého datového typu bude (jaký v ní bude obsah). Jazyk ji v paměti založí a teprve potom s ní můžeme pracovat. Podle datového typu proměnné si ji jazyk dokáže z paměti načíst, modifikovat, případně ji v paměti založit. O každém datovém typu jazyk ví, kolik v paměti zabírá místa a jak s tímto kusem paměti pracovat.

Typový systém

Existují dva základní **typové systémy**: **statický** a **dynamický**.

Dynamický typový systém nás plně odstiňuje od toho, že proměnná má vůbec nějaký datový typ. Ona ho samozřejmě vnitřně má, ale jazyk to nedává nejevo. Dynamické typování jde mnohdy tak daleko, že proměnné nemusíme ani deklarovat, jakmile do nějaké proměnné něco uložíme a jazyk zjistí, že nebyla nikdy deklarována, sám ji založí. Do té samé proměnné můžeme ukládat text, potom objekt uživatele a potom desetinné číslo. Jazyk se s tím sám popere a vnitřně automaticky mění datový typ. V těchto jazycích jde vývoj rychleji díky menšímu množství kódu, zástupci jsou např. PHP nebo Ruby.

Statický typový systém naopak striktně vyžaduje definovat typ proměnné a tento typ je dále neměnný. Jakmile proměnnou jednou deklarujeme, není možné její datový typ změnit. Jakmile se do textového řetězce pokusíme uložit objekt uživatel, dostaneme vynadáno.

Java je staticky typovaný jazyk, všechny **proměnné musíme nejprve deklarovat** s jejich datovým typem. Nevýhodou je, že díky deklaracím je zdrojový kód poněkud objemnější a vývoj pomalejší. Obrovskou výhodou však je, že nám kompiler před spuštěním zkontroluje, zda všechny datové typy sedí. Dynamické typování sice vypadá jako výhodné, ale zdrojový kód není možné automaticky kontrolovat a když někde očekáváme objekt uživatel a přijde nám tam místo toho desetinné číslo, odhalí se chyba až za běhu a interpret program shodí. Naopak Java nám nedovolí program ani zkompilovat a na chybu nás upozorní (to je další výhoda kompilace).

Pojďme si nyní něco naprogramovat, ať si nabyté znalosti trochu osvojíme, s teorií budeme pokračovat až příště. Řekněme si nyní tři základní datové typy:

- Celá čísla: **int**
- Desetinná čísla: **float**
- Textový řetězec: **String**

String píšeme s velkým písmenem na začátku.

Program vypisující proměnnou

Zkusíme si nadeklarovat celočíselnou proměnnou *a*, dosadit do ní číslo 56 a její obsah vypsat do konzole. Založte si nový projekt, pojmenujte ho Vypis (i ke všem dalším příkladům si vždy založte nový projekt). Kód samozřejmě píšeme do těla metody main (jako minule), čili ji zde již nebudu opisovat.

Spustit kód Klikni pro editaci

```
int a;
a = 56;
System.out.println(a);
```

První příkaz nám nadeklaruje novou proměnnou *a* datového typu int, proměnná tedy bude sloužit pro ukládání celých čísel. Druhý příkaz provádí přiřazení do proměnné, slouží k tomu operátor "rovná se". Poslední příkaz je nám známý, vypíše do konzole obsah proměnné *a*. Konzole je chytrá a umí vypsat i číselné proměnné.

Konzolová aplikace 56

Pro desetinnou proměnnou by kód vypadal takto:

Spustit kód Klikni pro editaci

```
float a;
a = 56.6F;
System.out.println(a);
```

Je to téměř stejné jako s celočíselným. Jako desetinný oddělovač používáme tečku a na konci desetinného čísla je nutné zadat tzv. suffix F, tím říkáme, že se jedná o float.

Program Papoušek

Napíšeme program papoušek, který bude dvakrát opakovat to, co uživatel napsal. Ještě jsme nezkoušeli z konzole nic načítat. Slouží k tomu metoda nextLine(), která uživateli umožní zadat do

konzole řádku textu a nám vrátí zadaný textový řetězec. Abychom mohli z konzole načítat, založíme si nový projekt s názvem Papousek a jeho kód upravíme tak, aby vypadal takto:

```
package papousek;

import java.util.Scanner;

public class Papousek {

   public static void main(String[] args)
   {
      Scanner sc = new Scanner(System.in, "Windows-1250");

   }
}
```

Pro názornost je vymazána šedá dokumentace, ale klidně si ji tam nechte. Změna spočívá v importování java.util.Scanner, což nám umožňuje přístup k metodám pro vstup z konzole. Konečně ten dlouhý řádek na začátku metody nedělá nic jiného, než že nám vytvoří proměnnou sc, na které můžeme volat onu metodu nextLine(), která načte z konzole další řetězec. Vysvětlit si ho by bylo nyní příliš komplikované, berte ho jako že tam je, časem ho pochopíme.

Pokud budete potřebovat v kterémkoli ze svých programů načíst text z konzole, je nutné program takto upravit a přidat proměnnou sc!

Nyní se přesuňme k samotnému kódu programu a upravme obsah metody main() do následující podoby:

Spustit kód Klikni pro editaci

```
        Scanner sc = new Scanner(System.in, "Windows-1250");
        System.out.println("Ahoj, jsem virtuální papoušek Lóra, rád opakuji!");
        System.out.println("Napiš něco: ");
        String vstup;
        vstup = sc.nextLine();
        String vystup;
        vystup = vstup + ", " + vstup + "!";
        System.out.println(vystup);


```

To už je trochu zábavnější. První řádek jsme ji již vysvětlili výše, další dva řádky jsou jasné (vypisují text). Dále deklarujeme textový řetězec *vstup*. Do *vstupu* se přiřadí hodnota z metody nextLine() na konzoli, tedy to, co uživatel zadal. Pro výstup si pro názornost zakládáme další proměnnou typu textový řetězec. Zajímavé je, jak do *vystup* přiřadíme, tam využíváme tzv. konkatenace (spojování) řetězců. Pomocí operátoru "+" totiž můžeme spojit několik textových řetězců do jednoho a je jedno, jestli je řetězec v proměnné nebo je explicitně zadán v uvozovkách ve zdrojovém kódu. Do proměnné tedy přiřadíme vstup, dále čárku, znovu vstup a poté vykřičník. Proměnnou vypíšeme a skončíme.

Konzolová aplikace Ahoj, jsem virtuální papoušek Lóra, rád opakuji!
Napiš něco:

Nazdar ptáku
Nazdar ptáku, Nazdar ptáku!

Do proměnné můžeme přiřazovat již v její deklaraci, můžeme tedy nahradit:

```
String vstup;
vstup = sc.nextLine();
```

za

```
String vstup = sc.nextLine();
```

Program by šel zkrátit ještě více v mnoha ohledech, ale obecně je lepší používat více proměnných a dodržovat přehlednost, než psát co nejkratší kód a po měsíci zapomenout, jak vůbec funguje.

Program zdvojnásobovač

Zdvojnásobovač si vyžádá na vstupu číslo a to poté zdvojnásobí a vypíše. Asi bychom s dosavadními znalostmi napsali něco takového:

```
Scanner sc = new Scanner(System.in, "Windows-1250");
System.out.println("Zadejte číslo k zdvojnásobení:");
int a = sc.nextLine();
a = a * 2;
System.out.println(a);
```

Všimněte si zdvojnásobení čísla *a*, které jsme provedli pomocí přiřazení. Java nám nyní vyhubuje a podtrhne řádek, ve kterém se snažíme hodnotu z konzole dostat do proměnné typu int. Narážíme na typovou kontrolu, konkrétně nám nextLine() vrací String a my se ho snažíme uložit do intu. Budeme ho potřebovat tzv. **naparsovat**.

Parsování

Parsováním se myslí převod z textové podoby na nějaký specifický typ, např. číslo. Mnoho datových typů má v Javě již připraveny metody k parsování, pokud budeme chtít naparsovat např. int ze Stringu, budeme postupovat takto:

```
String s = "56";
int a = Integer.parseInt(s);
```

Metoda parseInt() se volá na třídě Integer, bere jako parametr textový řetězec a vrátí číslo. Využijeme této znalosti v našem programu:

Spustit kód Klikni pro editaci

```
Scanner sc = new Scanner(System.in, "Windows-1250");
System.out.println("Zadejte číslo k zdvojnásobení:");
String s = sc.nextLine();
```

```
    int a = Integer.parseInt(s);
    a = a * 2;
    System.out.println(a);
```

Nejprve si text z konzole uložíme do textového řetězce *s*. Do celočíselné proměnné *a* poté uložíme číselnou hodnotu řetězce *s*. Dále hodnotu v *a* zdvojnásobíme a vypíšeme do konzole.

Konzolová aplikace Zadejte číslo k zdvojnásobení:
1024
2048

Parsování se samozřejmě nemusí povést, když bude v textu místo čísla např. slovo, ale tento případ zatím nebudeme ošetřovat.

Na skeneru existuje i metoda `nextInt()`, u které by se mohlo na první pohled zdát, že nám vrátí již naparsované číslo a tedy i ušetří práci. Ve skutečnosti tato metoda bohužel ponechá ve vstupu znak Enteru, kterým uživatel číslo potvrdil. Tento znak tam zůstane tak dlouho, dokud jej nenačtete z konzole spolu s dalším textem, což může způsobit neočekávané problémy ve vašich programech. Proto používejte ke čtení z konzole vždy metodu `nextLine()`.

Jednoduchá kalkulačka

Ještě jsme nepracovali s desetinnými čísly, zkusme si napsat slibovanou kalkulačku. Bude velmi jednoduchá, na vstup přijdou dvě čísla, program poté vypíše výsledky pro sčítání, odčítání, násobení a dělení.

Spustit kód Klikni pro editaci

```
    Scanner sc = new Scanner(System.in, "Windows-1250");
    System.out.println("Vítejte v kalkulačce");
    System.out.println("Zadejte první číslo:");
    float a = Float.parseFloat(sc.nextLine());
    System.out.println("Zadejte druhé číslo:");
    float b = Float.parseFloat(sc.nextLine());
    float soucet = a + b;
    float rozdil = a - b;
    float soucin = a * b;
    float podil = a / b;
    System.out.println("Součet: " + soucet);
    System.out.println("Rozdíl: " + rozdil);
    System.out.println("Součin: " + soucin);
    System.out.println("Podíl: " + podil);
    System.out.println("Děkuji za použití kalkulačky, aplikaci ukončíte libovolnou klávesou.");
```

Konzolová aplikace Vítejte v kalkulačce

```
Zadejte první číslo:
3.14
Zadejte druhé číslo:
2.72
Součet: 5.86
Rozdíl: 0.42
Součin: 8.5408
Podíl: 1.15441176470588
Děkuji za použití kalkulačky, aplikaci ukončíte libovolnou klávesou.
```

Všimněte si dvou věcí. Zaprvé jsme zjednodušili parsování z konzole tak, abychom nepotřebovali Stringovou proměnnou, protože bychom ji stejně již poté nepoužili. Zadruhé na konci programu spojujeme řetězec s číslem pomocí znaménka plus. Java překvapivě nezahlásí chybu, ale provede tzv. implicitní konverzi a zavolá na čísle metodu Integer.toString() nebo přímo na Stringu String.valueOf(). Kdyby tomu tak nebylo nebo jsme se dostali do situace, kdy potřebujeme převést cokoli na String, zavoláme metodu String.valueOf() a jako parametr ji dáme naši proměnnou. Java to v tomto případě udělala za nás, v podstatě vykoná:

```
System.out.println("Součet: " + String.valueOf(soucet));
```

Právě jsme se tedy naučili opak k parsování - převést cokoli do textové podoby. V příští lekci, Typový systém podruhé: Datové typy, si řekneme více o typovém systému v Javě a představíme si další datové typy.

Primitivní datové typy

Proměnné primitivního datového typu si dokážeme jednoduše představit. Může se jednat např. o číslo nebo znak. V paměti je jednoduše uložena přímo hodnota a my k této hodnotě můžeme z programu přímo přistupovat. Slovo přímo jsem tolikrát nepoužil jen náhodou. V této sekci tutoriálů se budeme věnovat výhradně těmto proměnným.

Celočíselné datové typy

Podívejme se nyní na tabulku všech vestavěných celočíselných datových typů v Javě, všimněte si typu int, který již známe.

Datový typ	Rozsah	Velikost
byte	-128 až 127	8 bitů
short	-32 768 až 32 767	16 bitů
int	-2 147 483 648 až 2 147 483 647	32 bitů
long	-9 223 372 036 854 775 808 až 9 223 372 036 854 775 807	64 bitů

Asi vás napadá otázka, proč máme tolik možných typů pro uložení čísla. Odpověď je prostá, záleží na jeho velikosti. Čím větší číslo, tím více spotřebuje paměti. Pro věk uživatele tedy zvolíme byte, protože se jistě nedožije více, než 127 let. Představte si databázi milionu uživatelů nějakého systému, když zvolíme místo byte int, bude zabírat 4x více místa. Naopak když budeme mít funkci k výpočtu faktoriálu, stěží nám bude stačit rozsah intu a použijeme long.

Nad výběrem datového typu nemusíte moc přemýšlet a většinou se používá jednoduše int. Typ řešte pouze v případě, když jsou proměnné v nějakém poli (obecně kolekci) a je jich tedy více, potom se vyplatí zabývat se paměťovými nároky. Tabulky sem dávám spíše pro úplnost. Mezi typy samozřejmě funguje již zmíněná implicitní konverze, tedy můžeme přímo přiřadit int do proměnné typu long a podobně, aniž bychom něco konvertovali.

Desetinná čísla

U desetinných čísel je situace poněkud jednodušší, máme na výběr pouze dva datové typy. Samozřejmě se liší opět v rozsahu hodnoty, dále však ještě v přesnosti (vlastně počtu des. míst). Double má již dle názvu dvojnásobnou přesnost oproti float.

Datový typ	Rozsah	Přesnost
float	$+-1.5 * 10^{-45}$ až $+-3.4 * 10^{38}$	7 čísel
double	$+-5.0 * 10^{-324}$ až $+-1.7 * 10^{308}$	15-16 čísel

Pozor, vzhledem k tomu, že desetinná čísla jsou v počítači uložena ve dvojkové soustavě, dochází k určité ztrátě přesnosti. Odchylka je sice téměř zanedbatelná, nicméně když budete programovat např. finanční systém, nepoužívejte tyto dat. typy pro uchování peněz, mohlo by dojít k malým odchylkám.

Když do floatu chceme dosadit přímo ve zdrojovém kódu, musíme použít sufix F, u double sufix D (u double ho můžeme vypustit, jelikož je výchozí desetinný typ):

```
float f = 3.14F;
double d = 2.72;
```

Jako desetinný separátor používáme ve zdrojovém kódu vždy tečku, nehledě na to, jaké máme v operačním systému regionální nastavení.

Další vestavěné datové typy

Podívejme se na další datové typy, které nám Java nabízí:

Datový typ	Rozsah	Velikost/Přesnost
char	U+0000 až U+ffff	16 bitů

Datový typ	Rozsah	Velikost/Přesnost
boolean	true nebo false	8 bitů

Char

Char nám reprezentuje jeden znak, narozdíl od Stringu, který reprezentoval celý řetězec charů. Znaky v Javě píšeme do apostrofů:

```
char c = 'A';
```

Char patří v podstatě do celočíselných proměnných (obsahuje číselný kód znaku), ale přišlo mi logičtější uvést ho zde.

BigDecimal

Typ BigDecimal řeší problém ukládání desetinných čísel v binární podobě, ukládá totiž číslo vnitřně jako pole. Používá se tedy pro uchování peněžních hodnot. Nebudeme si zde ukazovat použití, protože se používá jako objekt. Pokud budete dělat někdy finanční výpočty, tak si na něj vzpomeňte.

Pozn.: V Javě jsou čísla tzv. odděděna od třídy Number. To je spíše informace do budoucna. Jelikož nyní nevíme, co dědičnost znamená, důležitá pro nás není. Number obsahuje ještě čtyři podtřídy, kterými se nebudeme podrobněji zabývat. BigDecimal a BigInteger slouží k výpočtům s vysokou přesností. Třídy AtomicInteger a AtomicLong se používají v aplikacích s více podprocesy. Opět je důležité, abyste věděli, že něco takového existuje a případně se sem později vrátili.

Boolean

Boolean nabývá dvou hodnot: true (pravda) a false (nepravda). Budeme ho používat zejména tehdy, až se dostaneme k podmínkám. Do proměnné typu boolean lze uložit jak přímo hodnotu true/false, tak i logický výraz. Zkusme si jednoduchý příklad:

Spustit kód Klikni pro editaci

```
boolean b = false;
boolean vyraz = (15 > 5);
System.out.println(b);
System.out.println(vyraz);
```

Výstup programu:

```
Konzolová aplikace false
true
```

Výrazy píšeme do závorek. Vidíme, že výraz nabývá hodnoty true (pravda), protože 15 je opravdu větší než 5. Od výrazů je to jen krok k podmínkám, na ně se podíváme příště.

Referenční datové typy

K referenčním typům se dostaneme až u objektově orientovaného programování, kde si také vysvětlíme zásadní rozdíly. Zatím budeme pracovat jen s tak jednoduchými typy, že rozdíl nepoznáme. Spokojíme se s tím, že referenční typy jsou složitější, než ty primitivní. Jeden takový typ již známe, je jím String. Možná vás napadá, že String nemá nijak omezenou délku, je to tím, že s referenčními typy se v paměti pracuje jinak. Hodnotové typy začínají narozdíl od typů primitivních velkým písmenem.

String má na sobě řadu opravdu užitečných metod. Některé si teď probereme a vyzkoušíme:

String

StartsWith(), endsWith() a contains()

Můžeme se jednoduše zeptat, zda řetězec začíná, končí nebo zda obsahuje určitý podřetězec (substring). Podřetězcem myslíme část původního řetězce. Všechny tyto metody budou jako parametr brát samozřejmě podřetězec a vracet hodnoty typu boolean (true/false). Zatím na výstup neumíme reagovat, ale pojďme si ho alespoň vypsat:

Spustit kód Klikni pro editaci

```
String s = "Krokonosohroch";
System.out.println(s.startsWith("krok"));
System.out.println(s.endsWith("hroch"));
System.out.println(s.contains("nos"));
System.out.println(s.contains("roh"));
```

Výstup programu:

```
Konzolová aplikace false
true
true
false
```

Vidíme, že vše funguje podle očekávání. První výraz samozřejmě neprošel díky tomu, že řetězec ve skutečnosti začíná velkým písmenem.

toUpperCase() a toLowerCase()

Rozlišování velkých a malých písmen může být někdy na obtíž. Mnohdy se budeme potřebovat zeptat na přítomnost podřetězce tak, aby nezáleželo na velikosti písmen. Situaci můžeme vyřešit pomocí metod toUpperCase() a toLowerCase(), které vrací řetězec ve velkých a v malých písmenech. Uveďme si reálnější příklad než je Krokonosohroch. Budeme mít v proměnné řádek konfiguračního souboru, který psal uživatel. Jelikož se na vstupy od uživatelů nelze spolehnout, musíme se snažit eliminovat možné chyby, zde např. s velkými písmeny.

Spustit kód Klikni pro editaci

```
String konfig = "Fullscreen shaDows autosave";
konfig = konfig.toLowerCase();
System.out.println("Poběží hra ve fullscreenu?");
System.out.println(konfig.contains("fullscreen"));
System.out.println("Budou zapnuté stíny?");
System.out.println(konfig.contains("shadows"));
System.out.println("Přeje si hráč vypnout zvuk?");
System.out.println(konfig.contains("nosound"));
System.out.println("Přeje si hráč hru automaticky ukládat?");
System.out.println(konfig.contains("autosave"));
```

Výstup programu:

```
Konzolová aplikace Poběží hra ve fullscreenu?
true
Budou zapnuté stíny?
true
Přeje si hráč vypnout zvuk?
false
Přeje si hráč hru automaticky ukládat?
true
```

Vidíme, že jsme schopni zjistit přítomnost jednotlivých slov v řetězci tak, že si nejprve řetězec převedeme celý na malá písmena (nebo na velká) a potom kontrolujeme přítomnost slova jen malými (nebo velkými) písmeny. Takhle by mimochodem mohlo opravdu vypadat jednoduché zpracování nějakého konfiguračního skriptu.

Trim()

Další nástrahou mohou být mezery a obecně všechny tzv. bílé znaky, které nejsou vidět, ale mohou nám uškodit. Obecně může být dobré trimovat všechny vstupy od uživatele. Zkuste si v následující aplikaci před číslo a za číslo zadat několik mezer, trim() je odstraní. Odstraňují se vždy bílé znaky kolem řetězce, nikoli uvnitř:

Spustit kód Klikni pro editaci

```
System.out.println("Zadejte číslo:");
String s = sc.nextLine();
System.out.println("Zadal jste text: " + s);
System.out.println("Text po funkci trim: " + s.trim());
int a = Integer.parseInt(s.trim());
System.out.println("Převedl jsem zadaný text na číslo parsováním, zadal jste: " + a);
```

Replace()

Asi nejdůležitější metodou na Stringu je nahrazení určité jeho části jiným textem. Jako parametry zadáme dva podřetězce, jeden co chceme nahrazovat a druhý ten, kterým to chceme nahradit.

Metoda vrátí nový String, ve kterém proběhlo nahrazení. Když daný podřetězec metoda nenajde, vrátí původní řetězec. Zkusme si to:

Spustit kód Klikni pro editaci

```
String s = "C# je nejlepší!";
s = s.replace("C#", "Java");
System.out.println(s);
```

Výstup programu:

Konzolová aplikace Java je nejlepší!

Format()

Format() je velmi užitečná metoda, která nám umožňuje vkládat do samotného textového řetězce zástupné značky. Ty jsou reprezentovány jako procento a zkratka datového typu. Metoda se volá na typu String, prvním parametrem je textový řetězec se značkami, další dále následují proměnné v tom pořadí, v kterém se mají do textu místo značek vložit. Všimněte si, že se metoda nevolá na konkrétní proměnné (přesněji instanci, viz další díly), ale přímo na typu String.

Spustit kód Klikni pro editaci

```
int a = 10;
int b = 20;
int c = a + b;
String s = String.format("Když sečteme %d a %d, dostaneme %d", a, b, c);
System.out.println(s);
```

Výstup programu:

Konzolová aplikace Když sečteme 10 a 20, dostaneme 30

Značky jsou:

- %d pro celá čísla
- %s pro Stringy
- %f pro float. U float můžeme definovat délku desetinné části, např: %.2f pro dvě desetinná místa.

Konzole sama umí přijímat text v takovémto formátu, jen musíme místo println() volat printf(). Můžeme tedy napsat:

Spustit kód Klikni pro editaci

```
int a = 10;
int b = 20;
int c = a + b;
System.out.printf("Když sečteme %d a %d, dostaneme %d", a, b, c);
```

Toto je velmi užitečná a přehledná cesta, jak sestavovat řetězce, i přesto se však v Javě řetězce spojují spíše pomocí operátoru "+".

Length()

Poslední, ale nejdůležitější je length(), tedy délka. Vrací celé číslo, které představuje počet znaků v řetězci.

Spustit kód Klikni pro editaci

```
        System.out.println("Zadejte vaše jméno:");
        String jmeno = sc.nextLine();
        System.out.printf("Délka vašeho jména je: %d", jmeno.length());
```

Je toho ještě spoustu k vysvětlování a jsou další datové typy, které jsme neprobrali. Aby jsme však stále neprobírali jen teorii, ukážeme si již v příští lekci, Podmínky (větvení), podmínky a cykly. Potom bude naše programátorská výbava dostatečně velká k tomu, abychom tvořili zajímavé programy