Programuju: Java!

Struktura zdrojového kódu v Javě

Nejprve se pojďme podívat na to, jak správně vypadá struktura javového zdrojového kódu. Aplikace napsané v Javě se skládají z jednotlivých tříd – `class`, což jsou samostatné soubory, jež jsou vzájemně propojené. U jednoduchých aplikací se může vyskytovat i jen jedna třída.

Samotné třídy se pak skládají z jednotlivých metod, které obsahují různé proměnné, příkazy, podmínky, …

```
Trida {
      deklaraceNoveMetody() {
            promenna.volaniMetody();            // prikaz
            promenna.volaniJineMetody();        // prikaz
      }
      deklaraceDalsiNoveMetody() {
            promenna.volaniJesteJineMetody();   // prikaz
            if (promenna.nejakaPodminka()) {    // podminka
                promenna.volaniMetody();        // prikaz v podmince
            }
      }
}
```

Deklarace metody je přesně ten prostor, ve kterém se můžete dostatečně realizovat. Sem zapisujete jednotlivé příkazy, podmínky, … , které dohromady tvoří vaši jedinečnou aplikaci.

Proměnné

Co je to proměnná?

Na začátku je potřeba se shodnout na tom, co to proměnná je. Každý z nás zná proměnné z matematiky, takže představu máme. Ve světě počítačů to funguje obdobně - **je to místo v paměti počítače, do kterého můžete uložit nějakou hodnotu** (text, číslo, datum, skupinu nějakých údajů, …). Každá proměnná musí být nějakého určitého typu – tomu se říká **datový typ** proměnné. Každá proměnná má v paměti počítače vyhrazené pevné místo podle svého datového typu.

Java je staticky typovaný jazyk – to znamená, že všechny proměnné musíme nejprve před použitím deklarovat s jejich datovým typem. Obrovskou výhodou statické typovanosti je, že nám Java před spuštěním aplikace zkontroluje, zda všechny datové typy sedí a pokud ne, upozorní nás na to.

Pro zajímavost – existují i netypované jazyky, například PHP nebo Perl. V takových jazycích můžete do jedné proměnné vkládat v průběhu programu různé typy hodnot.

Datové typy

Jak už jsme si řekli, každá proměnná musí být v Javě nějakého určitého typu. Jaké tedy máme možnosti?

typ	popis	min. hodnota	max. hodnota
Byte	celé číslo	-128	+127
Short	celé číslo	-32768	+32767

Integer	celé číslo	-2147483648	+2147483647
Long	celé číslo	-9223372036854775808	+9223372036854775807
Float	číslo s desetinným rozvojem	-3.40282e+38	+3.40282e+38
Double	číslo s desetinným rozvojem	-179769.999999 (až 308 cifer)	+179769.999999 (až 308 cifer)
Character	znak UNICODE	/u0000	/uFFFF
Boolean	logická hodnota true/false		
String	řetězec znaků		

Pro naše účely tento výčet stačí, chcete-li dozvědět více, můžete najít další informace například tady:
HTTP://WWW.ITNETWORK.CZ/JAVA/ZAKLADY/JAVA-TUTORIAL-TYPOVY-SYSTEM-PODRUHE-DATOVE-TYPY-STRING

Deklarace a inicializace

Proměnnou musíme nejdříve tzv. deklarovat, tedy říci jazyku jak se bude jmenovat a jakého datového typu bude (= co do ní chceme vložit za obsah). Java jí v paměti vyhradí místo a teprve potom s ní můžeme pracovat. Deklarovat proměnnou musíme právě jednou.

```
Integer cislo1;                 // deklarace promenne cislo1 typu Integer
Integer cislo2;                 // deklarace promenne cislo2 typu Integer
String  krestniJmeno;           // deklarace promenne krestniJmeno typu String
String  prijmeni;               // deklarace promenne prijmeni typu String
String  celeJmeno;              // deklarace promenne celeJmeno typu String
```

Po deklaraci můžeme proměnnou používat. Přiřazovat do ní hodnotu (i opakovaně) nebo její hodnotu využít a přiřadit do jiné proměnné.

```
cislo1 = 1500;                  // prirazeni hodnoty 1500 do promenne cislo1
krestniJmeno = "Sandra";        // prirazeni textu do promenne krestniJmeno
cislo2 = 300;                   // prirazeni hodnoty 300 do promenne cislo2
cislo1 = cislo1 + 600;          // vyhodnoceni hodnoty 2100 a prirazeni do cislo1
prijmeni = "Vesela";            // prirazeni textu do promenne prijmeni
celeJmeno = krestniJmeno + " " + prijmeni;   // vyhodnoceni a prirazeni celeJmeno
cislo2 = cislo1 + cislo2;       // vyhodnoceni 2400 a prirazeni do promenne cislo2
```

Java umožňuje zkrácený zápis deklarace proměnné s okamžitým přiřazením hodnoty.

```
Integer cislo3;                 // puvodne: dlouhy zapis: pouze deklarace
cislo3 = 35;                    // pouze prirazeni hodnoty

Integer cislo4 = 45;            // zkraceny zapis: deklarace s okamzitym
                                // prirazenim hodnoty
```

Oba způsoby zápisu jsou rovnocenné. Používejte ten, který se vám víc líbí.

Pozor! Zkrácený zápis s deklarací proměnné a okamžitým přiřazením ale pořád obsahuje deklaraci, takže lze ve třídě použít jen jednou.

Není tedy možné napsat:

```
Integer pocetPenez;             // pouze deklarace v urcite casti kodu
Integer pocetPenez = 1500;      // pokus o znovu deklaraci s prirazenim hodnoty o
                                // kus dál
```

Tímto byste v podstatě Javu žádali o deklaraci proměnné a tutéž proměnou byste o řádek níž deklarovali ještě jednou. Jak je napsané výše, každá proměnná může být deklarovaná pouze jednou.

Jména proměnných si můžete zvolit téměř libovolně, i zde ovšem existují jistá omezení:

- jméno **nesmí začínat číslicí**, ale jinde v názvu se číslice vyskytovat smí
- jméno **nesmí obsahovat mezery**
- jméno **musí být unikátní** ve svém rozsahu platnosti
- jménem **nesmí být** některé z **rezervovaných klíčových slov** jazyka Java, obsaženo být může
- konvence pro jména proměnných říká, že by měly být psány **camelCasem** (velbloudí hrby) – první písmeno názvu vždy malým písmenem, pokud název obsahuje více slov, tak první písmeno každého dalšího slova vždy velké (cisarovyNoveSaty)
- 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 (CISLO_PI).

Upozornění na závěr: Java je case-sensitive (rozlišuje malá a velká písmena). Proměnné malypes a malyPes jsou různé.

Platnost proměnné

Program v Javě je rozdělen do bloků kódu. Každý blok je uzavřen do složených závorek { }. Tyto bloky mimo jiné určují, kde bude proměnná viditelná. 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.

Podívejme se na následující zápis programu:

```
CelaTrida {
	Integer cislo = 5;
	/* Java vyhradila místo pro celočíselnou proměnnou a uložila do něj hodnotu
	5, proměnná cislo je v tomto bloku viditelná */
	prvniBlok {
		cislo = 3;
		/* proměnná cislo je ve vnořeném bloku také viditelná, hodnota 5 byla
		přepsána hodnotou 3 */
		String 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 */
	String druhyZnak;
	//Java vyhradila místo pro druhyZnak.
	druhyBlok {
		Integer neco = 0;
		/* Java vyhradila místo pro celočíselnou proměnnou neco a uložila do
		ní hodnotu 0 */
		vnorenyBlok {
			Double realne = 2.3;
			druhyZnak = "a";
			/*Java uložila do proměnné realne hodnotu 2.3 a do proměnné
			druhyZnak písmeno a. V tomto místě lze použít proměnné realne,
			cislo, neco, druhyZnak. Nelze použít proměnnou znak. */
		}
```

```
        /* V tomto místě lze použít proměnné: cislo, neco, druhyZnak. Nelze
         použít proměnné znak a realne. */
    }
    /* V tomto místě lze použít proměnné cislo, druhyZnak. Nelze použít
    proměnné znak, neco a realne. */
}
```

Jde v podstatě o princip dědictví. Potomci dědí od rodičů, ovšem ne naopak. Sourozenci mezi sebou se navzájem nedědí. Když bych tedy přepsala výše uvedený kód do vztahu rodič-potomek, dostanu následující:

```
Rodic {
	Integer cislo;
	prvniPotomek {
		String znak;
	}
	String druhyZnak;
	druhyPotomek {
		Integer neco;
		potomekDruhehoPotomka {
			Double realne;
		}
	}
}
```

Rodič vidí v sobě deklarované proměnné (`cislo`, `druhyZnak`) a umí s nimi pracovat.
První potomek vidí své proměnné (tedy `znak`) a proměnné rodiče deklarované do svého počátku (tedy `cislo`).
Druhý potomek vidí své proměnné (tedy `neco`) a proměnné rodiče deklarované do svého počátku (tedy `cislo` a `druhyZnak`). Proměnnou `znak` nevidí, jelikož není o úroveň výš a v sousední úrovni.
Potomek druhého potomka vidí své proměnné (tedy `realne`) a proměnné rodičů a prarodičů deklarované do svého počátku (tedy `cislo`, `druhyZnak`, `neco`).

Pokud bychom chtěli používat například proměnou `realne` v celé třídě, musel by původní kód vypadat například takto:

```
CelaTrida {
	Integer cislo = 5;
	Double realne;
	prvniBlok {
		cislo = 3;
		String znak = "x";
	}
	String druhyZnak;
	druhyBlok {
		Integer neco = 0;
		vnorenyBlok {
			realne = 2.3;
			druhyZnak = "a";
		}
	}
}
```