# Další metody pro class Třída

V této kapitole si naprogramujeme pár metod pro naši novou classu Trida. Vždy je nutné podrobit pečlivé rozvaze, jaké metody chceme programovat a co přesně budou vykonávat. Je možné, že zde popsané „podprogramy“ byste řešili jiným způsobem. Záleží na zadání, které jste dostali od klienta. Jaké požadavky přesně na třídu budou kladeny, podle čeho se má metoda rozhodovat atd... následující kapitola popisuje jen pár možností, jak se třídou pracovat.

V minulé kapitole jsem slíbil začít metodou pridejNovehoZaka, která má za úkol přidat nového žáka do seznamu. Pojďme si o ní zapřemýšlet.

Nejjednodušší postup:

* Vytvoříme nového žáka pomocí konstruktoru ještě před voláním metody pridejNovehoZaka.
* Instance žáka bude vstupem to metody: **pridejNovehoZaka(Zak z)**.
* V těle metody pouze vložíme žáka do seznamu pomocí metody add: zaci.add(z);.

Kód by fungoval, co když ale žák ve třídě již existuje? Pak by byl ve třídě 2x což jistě není žádoucí. V tom případě bychom žáka do seznamu naopak přidávat nechtěli. Přibyl nám tedy další problém – jak reálně ověřit, že žák v seznamu již je?

Naskýtají se nám 3 možnosti:

* V metodě nejprve vytvoříme žáka z poskytnutých údajů a následně se dotazujeme seznamu, zda tento objekt již v seznamu je.
* Z objektu typu Zak vygenerujeme identifikátor, tzv. hash. Při prohledávání seznamu žáků tak porovnáváme pouze identifikátory.
* Prohledáváme seznam žáků a srovnáváme pouze určité atributy žáka, které si zvolíme. Například můžeme spoléhat na to, že kombinace jména, příjmení a roku narození má minimální šanci se u více žáků zopakovat. Výhodou tohoto řešení je, že nevytváříme nového žáka předčasně, ale jen tehdy, když to má smysl, tedy až po ověření, že v seznamu ještě není.

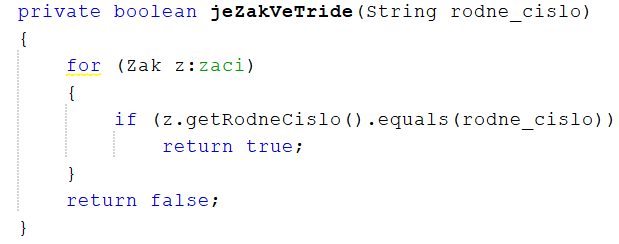
Za mě jsem si zvolil poslední možnost, ale k porovnávání studentů budu používat jediný atribut. Jednoznačný identifikátor – rodné číslo.

Už nyní vím, že přítomnost žáka ve třídě budu řešit v kódu vícekrát. Logicky si tedy vytvořím novou metodu. Nazvu ji jeZakVeTride. Vstupem bude hledané rodné číslo, tedy String. ArrayList do metody vstupovat nebude, neboť už je součástí objektu a přidávání na vstup by nemělo smysl.

Z metody bude vystupovat true/false, tedy žák ve třídě je/není.

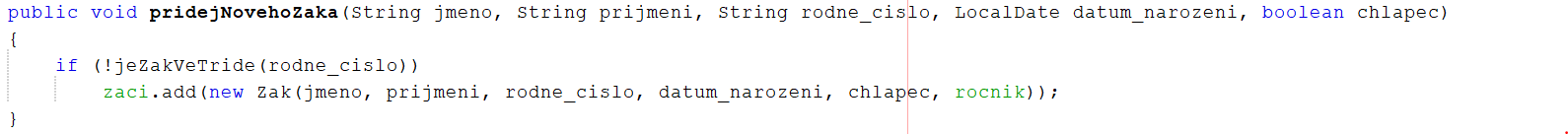
Metodu jsem se rozhodl označit modifikátorem přístupu private. Z pohledu programu to znamená, že metodu nebude možné zavolat z jiné classy než Trida. V podstatě vytváříme podprogram pouze  
pro potřeby této třídy.

Metoda prochází seznam a hledá shodu v rodném čísle. Pozor, předpokladem fungování je správné nadefinování metody getRodneCislo ve třídě Zak.

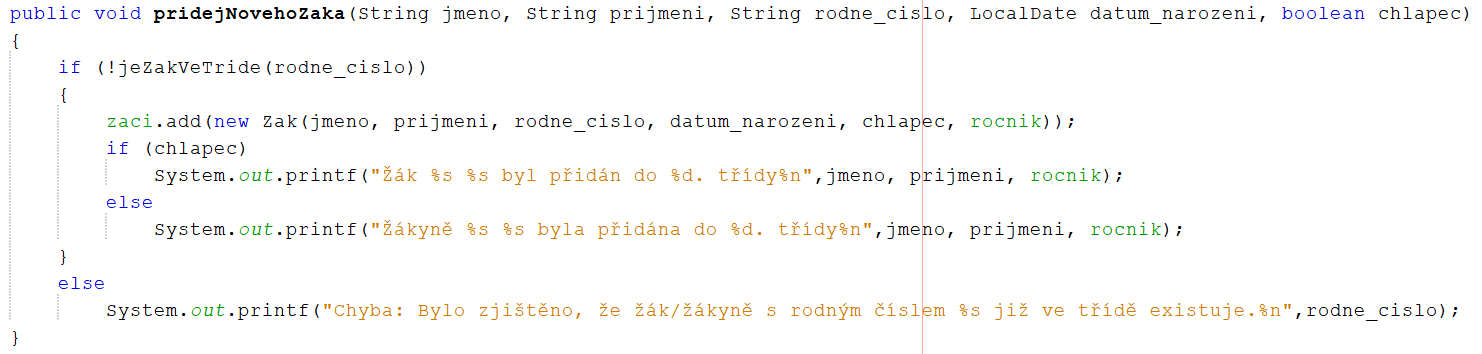


Jak je z kódu patrné, v podmínce pro porovnání nepoužíváme operátor ==, nýbrž metodu equals. Uvědomme si, že v podstatě porovnáváme objekty – textové řetězce, tedy použití equals je na místě.

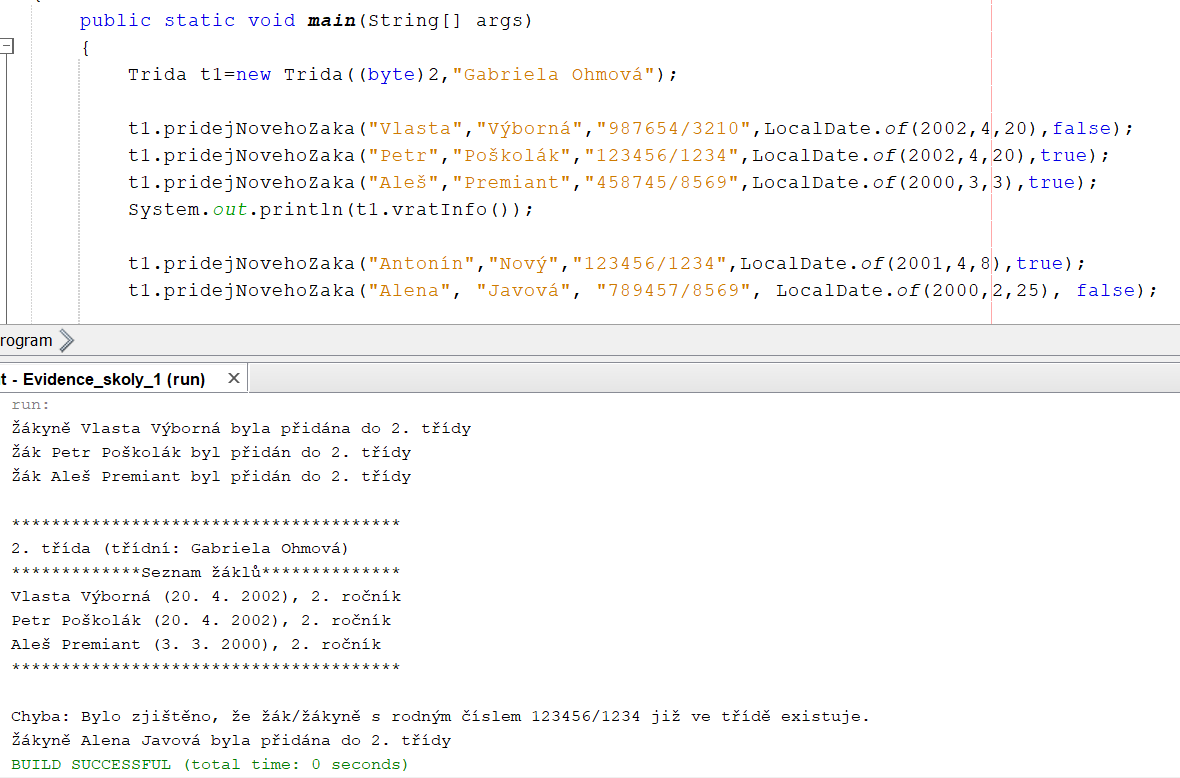
Do metody pridejNovehoZaka budou vstupovat veškeré atributy potřebné pro tvorbu žáka (pro volání konstruktoru třídy Zak). Tedy až na jeden, a to ročník. Ten se doplní automaticky dle třídy, do které žáka přidáváme. Pokud budu žáka přidávat do třídy, která má v atributu rocnik uloženu hodnotu 3, automaticky se v těle metody tato hodnota 3 přepíše do atributu ročník u žáka.



Abychom pro uživatele poskytli základní zpětnou vazbu, doplníme metodu o vypisování informací o stavu metody na konzoli. Uživatel bude mít přehled, co se v programu právě děje. Pro jednoduchost proti dosavadním zvyklostem nepoužijeme návratovou hodnotu metody. Vypisování bude řešit kód přímo v těle metody:



Pojďme si nyní funkčnost metody pridejNovehoZaka vyzkoušet. Nejprve ve třídě main vytvoříme novou třídu t1. Následně do ní přidáme 3 různé žáky. Úspěšné přidání můžeme kontrolovat na konzoli jak během samotného volání metody, tak během volání metody vratInfo na objektu t1.



Pokud se pokusíme přidat jiného žáka, ovšem s již existujícím rodným číslem, z hlediska funkcionality metody můžeme na konzoli číst chybu při zápisu do seznamu. V našem případě Antonín Nový má stejné rodné číslo, jako Petr Poškolák. V jednom z objektů je chyba, neboť stejné rodné číslo u více žáků není možné použít. Záznam(y) je nutno opravit.

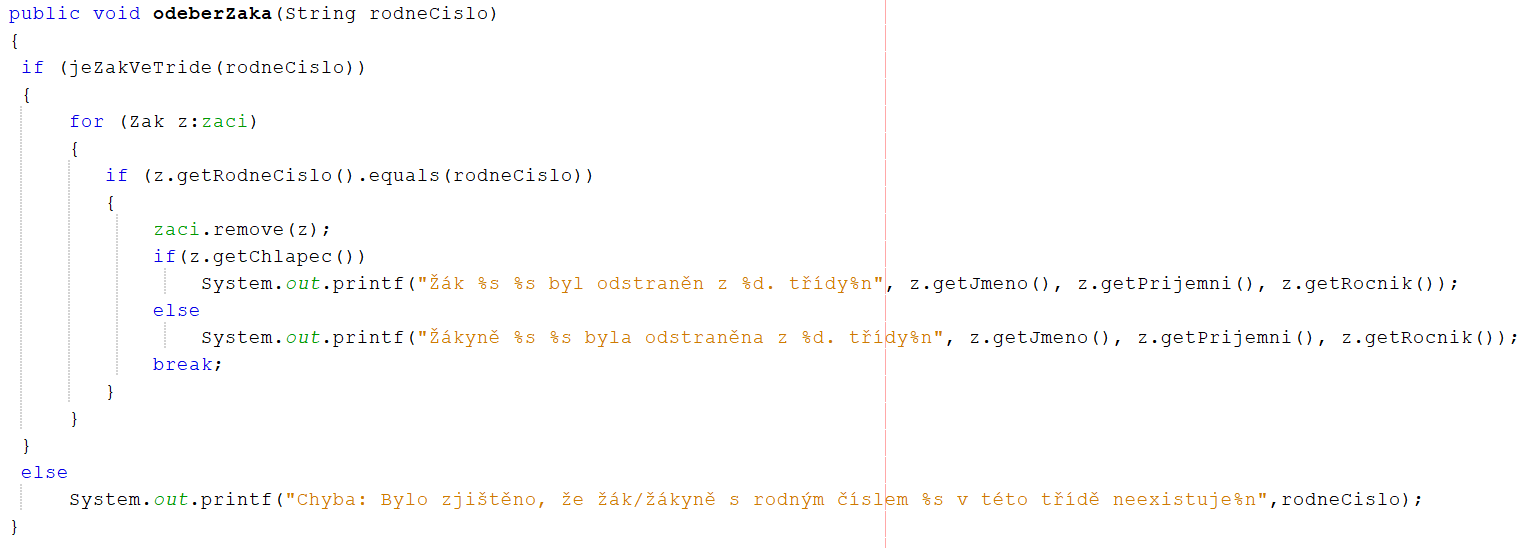
Alena Javová je posledním přidávaným žákem, její rodné číslo nekoliduje s žádným jiným, byla tedy úspěšně přidána do seznamu.

Již jsme schopni nového žáka do třídy přidat. Pojďme tedy zkusit stávajícího žáka ze třídy odstranit.

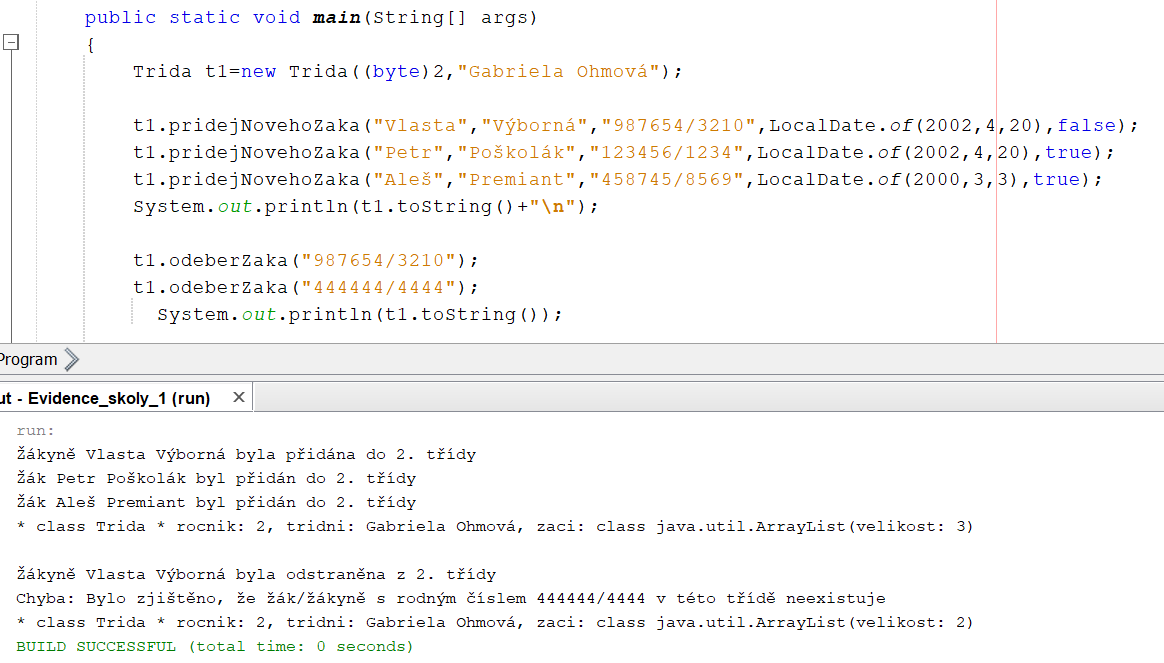
Metodu odeberZaka budeme řešit obdobně jako metodu předchozí. Nejprve podle rodného čísla zjistíme, zda se žák ve třídě opravdu nachází. Cyklem projdeme celý seznam žáků. V případě nalezení příslušného rodného čísla následně žáka ze seznamu třídy odstraníme. Činíme tak pomocí metody remove. Výpis o aktuálním dění bude mít na starosti opět samotný kód metody.

Neboť atributy žáka jsou označeny jako private, musíme se k jejich vnitřním hodnotám dostat přes gettery. Ujistíme se tedy, že nám příslušné gettry ve třídě žáka nechybí. Pokud ano, jednoduše je doprogramujeme, abychom se vyhnuli chybovému hlášení během kompilace programu.

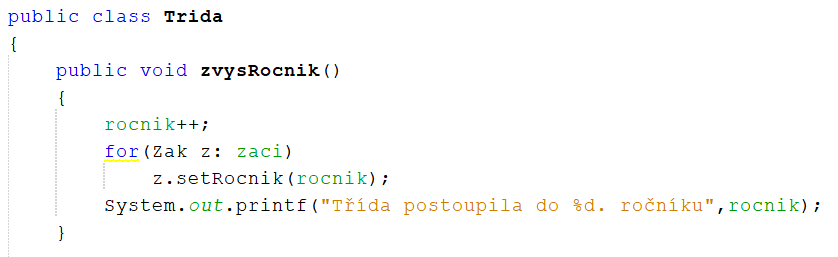
Pokud metoda odeberZaka zjistí, že žák se v seznamu třídy nenachází, pouze uživatele o této situaci informuje.



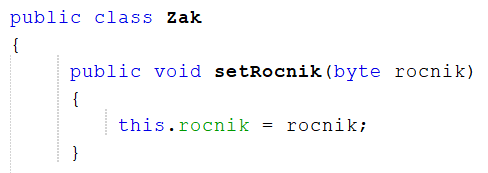
Ověření metody je snadné. Do třídy přidáme 3 různé žáky. Odstraníme žáka s existujícím rodným číslem a poté s neexistujícím. Čteme výpisy na konzoli. Je zřejmé, že ze seznamu žáků byl odebrán pouze jeden prvek.



Poslední metodou naprogramovanou v class Trida bude metoda pridejRocnik. Kód bude mít za úkol nejen upravit hodnotu vnitřního atributu rocnik. Upraví rovněž ročník u každého žáka nacházejícího se v seznamu příslušné třídy. Nedávalo by smysl, kdyby třída měla v názvu 3, ale všichni její žáci by v sobě měli uložen ročník 2. Projdeme tedy celý seznam žáků a u každého z nich zavoláme setter právě na vlastnost ročník.



Pokud ve třídě Zak metoda set schází, jednoduše ji doprogramujeme:



Ověření opět provedeme z metody main třídy Program.java:

