# Metody

Pokud by naše třídy obsahovaly pouze proměnné, fungovaly by maximálně jako jakýsi aparát pro ukládání hodnot. My ale potřebujeme, aby náš program ožil. Uměl s proměnnými pracovat. Měnil jejich hodnotu, vypisoval, ověřoval jejich správnost, počítal s nimi, ovlivňoval běh programu a mnoho dalšího. Správné konstruování metod je pro běh programu nezbytné. V podstatě jimi říkáme, které operace s objektem (třídou) či proměnnými lze dělat a které naopak nikoli. Časté jsou i metody vykonávající určité funkce bez navázání na stav vnitřních proměnných.

## Postup konstrukce metody:

* Rozvážíme, které operace chceme, aby naše třída uměla.
* Zkonstruujeme hlavičku:
  + Vymyslíme název, který metodu co možná nejlépe vystihuje. První písmeno názvu metody se v Javě uvádí malým písmenem.
  + Klíčovým slovem public/private určíme, zda bude možno volat metodu i z jiné třídy, např. ze třídy Program.java.
  + Pokusíme se přesně vystihnout, které hodnoty metoda potřebuje pro svou činnost (**určíme vstupy**).   
    Pozor, za vstupní nepovažujeme ty hodnoty, které jsou již definovány v objektu – které má metoda již k dispozici!
  + Uvážíme, zda má metoda vracet nějakou hodnotu či nikoli (výstup). Výstupem může být buď datový typ, objekt nebo nic (void).   
    Pozn. Výjimku tvoří konstruktory. U nich výstup nedefinujeme vůbec.
  + Pokud v rámci dědičnosti naše metoda existuje již v předkovi, pro její přepis doplníme klíčové slovo @Override. Takto učiníme již před samotnou hlavičkou metody.  
    Informace se tedy týká **polymorfismu**, v praxi si jej ukážeme později.
* Tělo metody
  + Píšeme do složených závorek.
  + V případě požadované návratové hodnoty tuto hodnotu označíme v kódu klíčovým slovem return.   
    Pozor - po přečtení příkazu return se vyskočí z metody pryč a další kód metody se již nevykoná. Proto „return“ řádek bývá zpravidla posledním řádkem metody.

Zde pro pořádek uvádím **obecný tvar zápisu hlavičky**:

Private/public výstupní datový typ Název metody (vstupní parametry s datovými typy)

## Dělení metod dle vstupů a výstupů:

* **Má vstupní parametry i výstup**   
  Hodí se např. na matematické operace.
* **Má pouze vstupní parametry**   
  Nejčastěji pro zápis (do proměnné objektu, do seznamu, do souboru, databáze...)
* **Má pouze výstup**  
  Nejčastěji pro čtení (z proměnné, seznamu, souboru, …)
* **Nemá vstupní parametry ani výstup**

## Příklady:

Představme si třídu Zprava, jejíž jediným atributem je textová proměnná uvitaci\_zprava.  
Proměnná je zapsána jako private, lze s ní tedy manipulovat pouze pomocí veřejných metod.

class Zprava

{

private string uvitaci\_zprava = "KRÁSNÝ DOBRÝ DEN !";

Příklad metody, která má vstupní parametry i výstup

public string vrat\_uvitaci\_zpravu\_s\_oslovenim(string jmeno)

{

return String.Format("{0}, {1}", jmeno, uvitaci\_zprava);

}

Příklad metody, která má pouze vstupní parametry

public void nastav\_uvitaci\_zpravu(string text)

{

uvitaci\_zprava = text;

}

Příklad metody, která má pouze výstup

public string vrat\_uvitaci\_zpravu()

{

return uvitaci\_zprava;

}

Příklad metody která nemá vstupní parametry ani výstup

public void Smaz\_uvitaci\_zpravu()

{

uvitaci\_zprava = "";

}

}

Pro pořádek uvádím výčet aktuálně existujících metod našeho projektu:

* Metoda main jako součást třídy Program.cs
* Konstruktory ve třídě Zak.cs

Snad jste si během čtení kapitoly udělali jasný obrázek o důležitosti metod. O různých typech metod a o správném syntaktickém zápisu. V příští kapitole opět rozšíříme náš program Evidence školy.

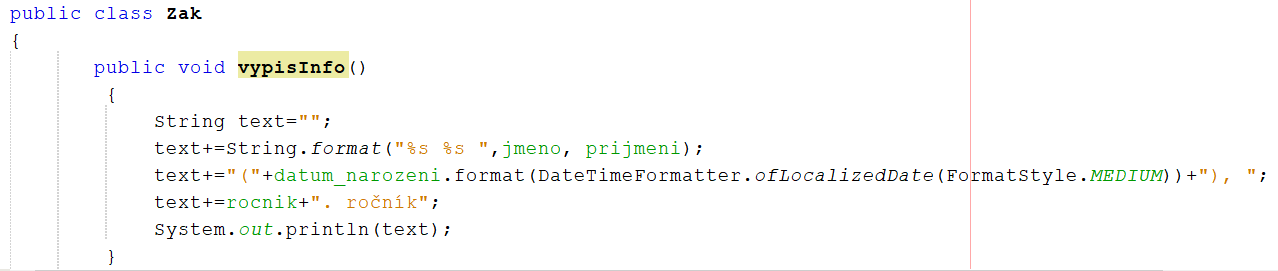
# Metody pro výpis informací

V této kapitole si naprogramujeme metody vypisující či vracející informace o daném objektu.

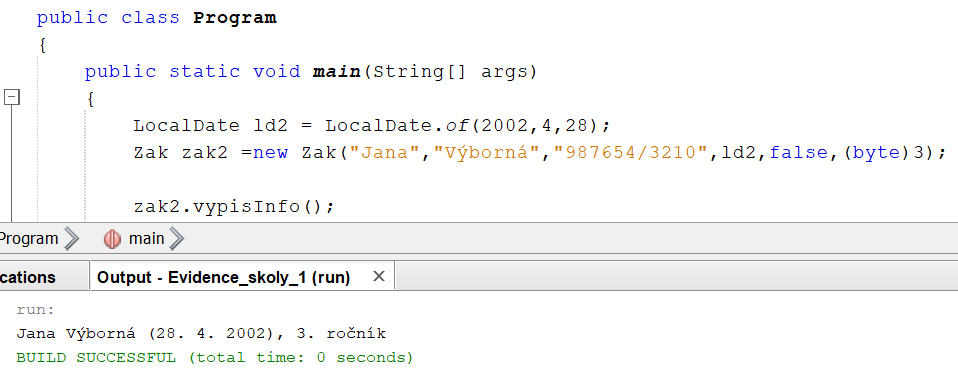
## Metoda vypisInfo

* Metoda bude vypisovat přímo na konzoli informace o žákovi, které nám přijdou důležité. Není nezbytné vypisovat veškeré informace, přizpůsobujeme se zadání.
* Řekněme si, že pro klienta je zajímavý výpis ve formátu:   
  Jana Výborná (28. 4. 2002), 3. ročník
* Metoda nebude mít žádný vstup (neboť objekt má všechny údaje uloženy v sobě, a tudíž k dispozici).
* Metoda nebude mít žádný výstup, neboť pouze vykonává danou činnost – posílá informace k výpisu rovnou na konzoli

Kód metody:



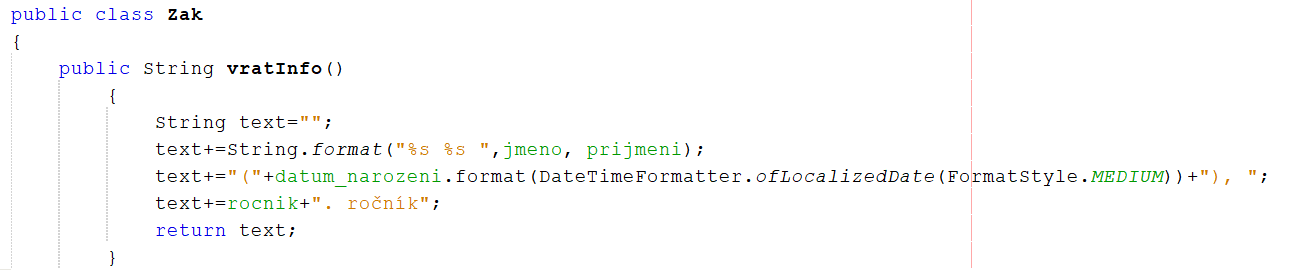
Volání metody a zobrazený výpis:



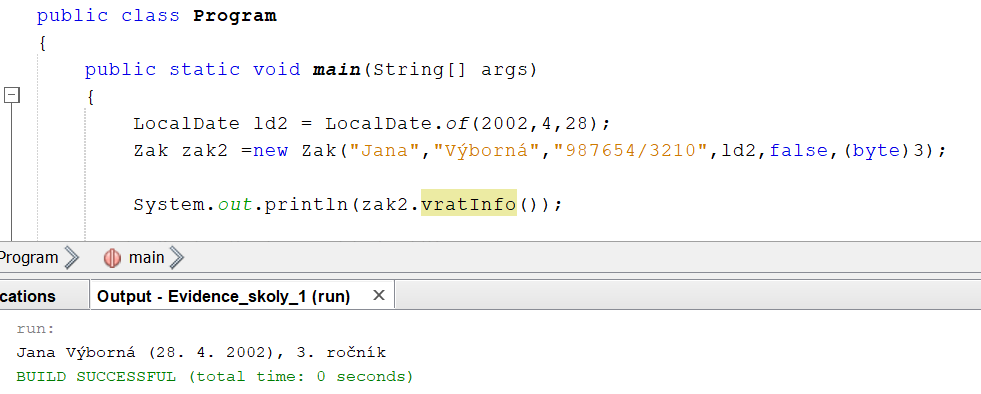
Metoda tedy funguje dle našich požadavků, má ale jeden zásadní nedostatek. Kdykoli bychom informace o objektech typu Zak chtěli vypisovat do jiného prostředí než konzolového, narazili bychom na limit námi konstruované třídy (metody).

Mnohem lepší variantou je tedy zařídit, aby metoda pouze vrátila požadovaný text a až posléze ve třídě Program.java by se rozhodlo, jak bude s výpisem naloženo, kde přesně bude poslán či zobrazen.

Metoda sama o sobě nic vypisovat nebude, bude pouze vracet text, proto změníme název. Místo vypisInfo ji pojmenujeme vratInfo. Do metody opět nemusíme nic posílat, návratová hodnota však zde již zastoupena bude, a to ve formě textu. Připomínám, že v těle metody je návratový řetězec určen klíčovým slovem return:



Až při volání metody z Program.java určujeme, že vrácený text bude zobrazen v konzoli:



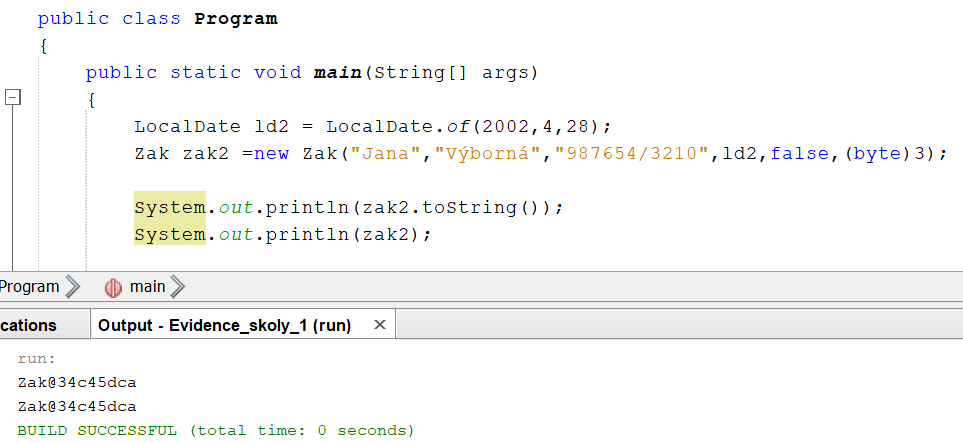
Je zřejmé, že metody vypisInfo i vratInfo dělají totéž. Pro širší použitelnost třídy však striktně doporučuji vyhnout se konstruování metod typu vypisInfo. Proto tuto metodu z kódu smažeme. Je výhodnější samotný výpis textu nechat na starosti hlavní třídě Program.java.

## Metoda toString

Metoda toString je v OOP hojně používaná. Umožňuje nám vracet text, stejně jako výše uvedená metoda vratInfo(). Je zde však jeden zásadní rozdíl. Metoda vratInfo je naší původní metodou. Oproti tomu metodu toString obsahuje již od počátku každý objekt. I náš objekt typu Zak ji obsahuje, aniž bychom ji ve třídě sami definovali. Jak je to možné? Díky dědičnosti.

V Javě existuje jedna hlavní základní třída s názvem Object. Její součástí, jak asi správně hádáte, je i metoda toString. Pokud u Vašeho objektu neoznačíte jiný dědický vztah, je dáno, že jeho rodičem se stává právě třída Object. Tedy metoda toString pochází právě odtud.

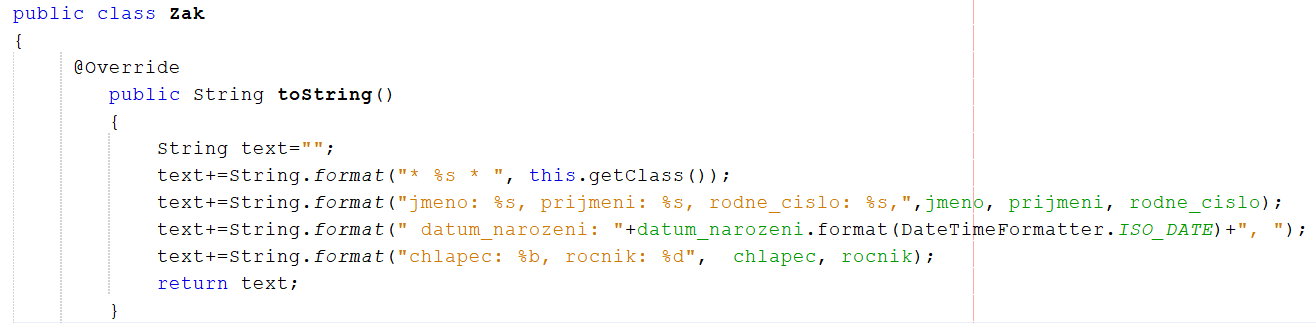
Metoda toString má v defaultním nastavení za úkol vracet textovou definici daného objektu – název třídy objektu a hash code objektu. Na screenu níže vidíte důkaz – metodu toString jsem ve třídě Zak nedefinovali, přesto nám ale program při jejím volání nevypisuje chybu, nýbrž textovou definici objektu zak2:



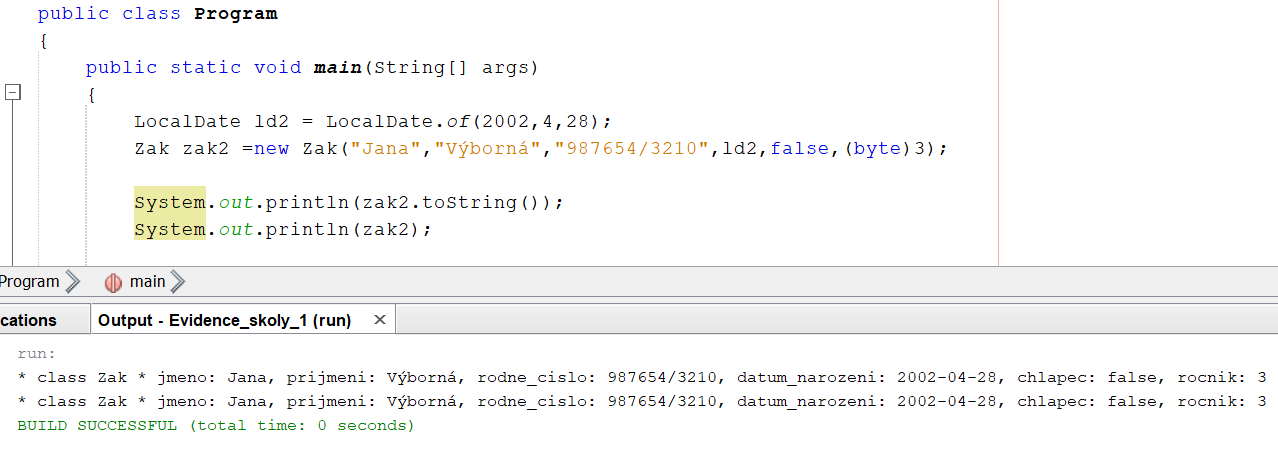
Dále je z obrázku vidět, že zde existuje **implicitní konverze** - pokud budeme chtít vypsat informace o objektu (zde pomocí System.out.println), kompilátor Javy implicitně metodu toString na objektu zavolá. Vidíte, že oba výpisy na konzoli jsou totožné.

Pokud byste chtěli metodu toString používat, ale její defaultní výpis je Vám k ničemu, lze chování této metody přepsat. Stačí ji ve vlastní třídě nadefinovat a před její definici použít zápis @Override.

Osobně jsem si zvyknul používat metodu toString tak, že mi vrací typ objektu a hodnoty všech svých vnitřních proměnných:



Volání metody a výpis pak vypadá následovně:



V následující kapitole si zkusíme vytvořit pro potřeby našeho programu několik dalších metod.