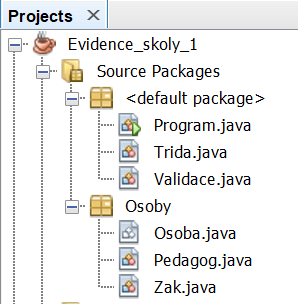
# Dědičnost – třída Pedagog

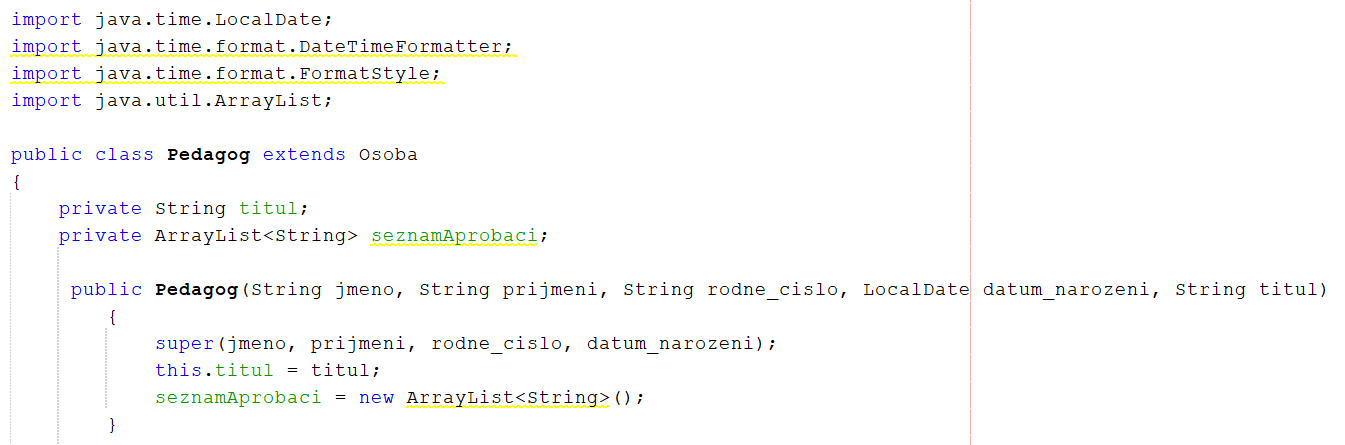
Velkou výhodou dědičnosti je, že můžeme z jednoho předka dědit více potomků. **Pozor,** **nelze z více předků nakombinovat jednoho potomka.**

V minulé kapitole jsme naznačili tvorbu nové třídy Pedagog. Jako základ použijeme naši abstraktní třídu Osoba. Třídu samozřejmě vytvoříme v nedávno vytvořeném balíčku Osoby.



Každý registrovaný pedagog bude mít povinné atributy jméno, příjmení, rodné číslo a datum narození. Mezi nové atributy přidáme titul a seznam pro ukládání aprobací.

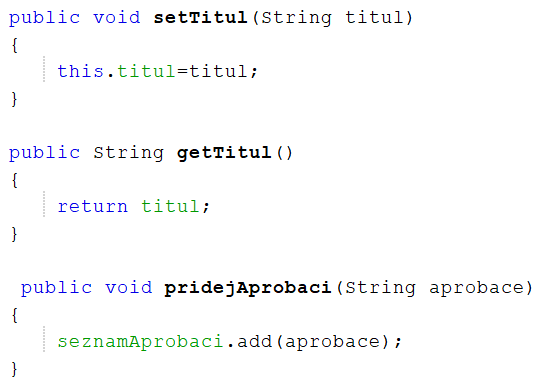
Dědičnost již máme probranou, proto si rovnou naprogramujeme i konstruktor.



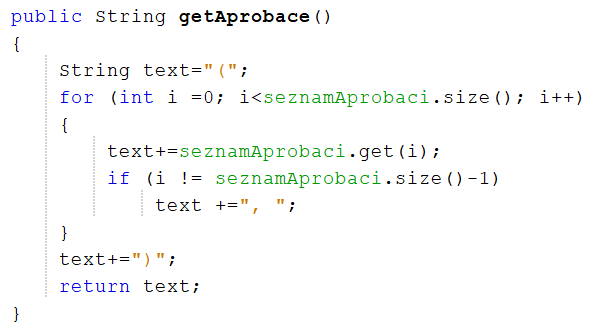
Pro třídu Pedagog po rozvaze doprogramujeme pár vhodných metod:

* getTitul
* setTitul
* pridejAprobaci – přidá do seznamu aprobací pedagoga další předmět
* getAprobace – vrátí textový řetězec všech aprobací vyučujícího v závorkách
* vratJmeno – vrátí celé jméno pedagoga včetně titulu
* vratInfo – vrátí některé informace o pedagogovi včetně jeho aprobací

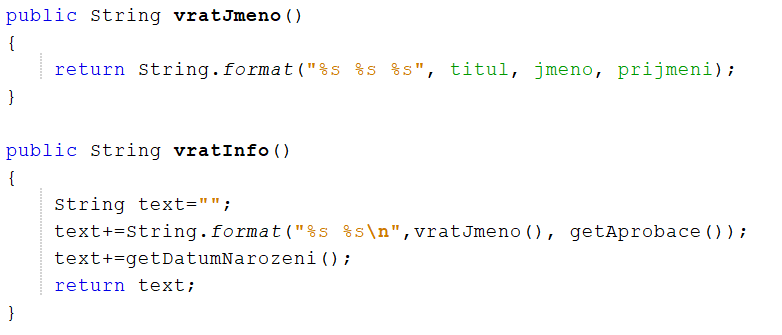
Metody getTitul, setTitul a pridejAprobaci jsou elementární, s podobnými kódy jsme se již dříve setkali:



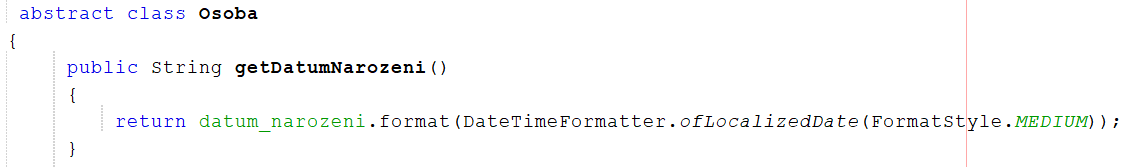
Metoda getAprobace tvoří textový řetězec aprobací uložených v seznamu Pedagoga. Text je v závorkách, jednotlivé předměty jsou odděleny čárkou.



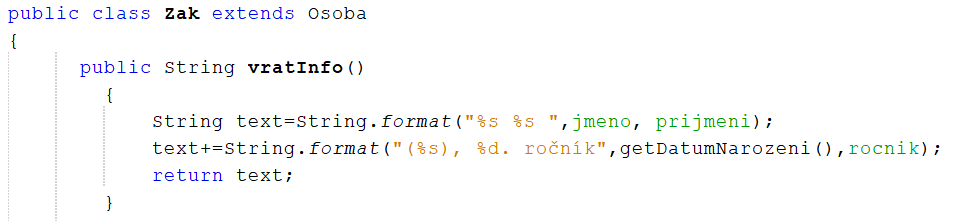
Metody vratJmeno a vratInfo vracejí informace o Pedagogovi v přehledné formě. Zajímavé je, že metoda vratInfo používá ve svém těle naše metody getAprobace a vratJmeno .



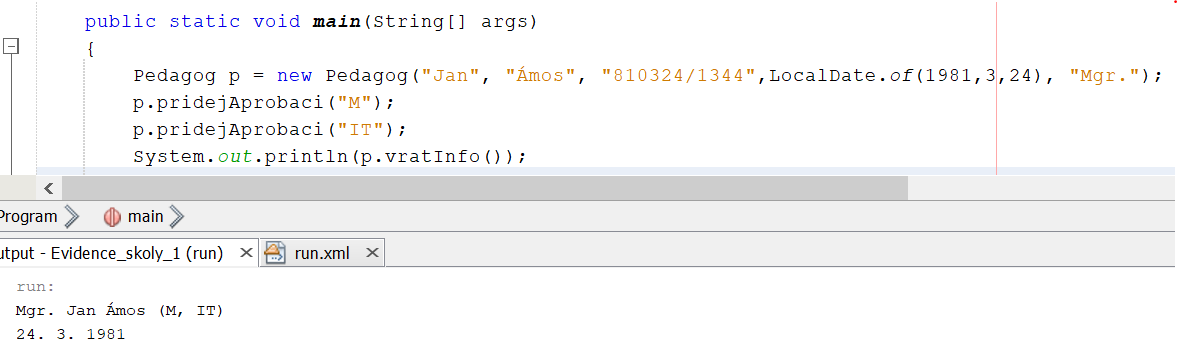
Jak lze výše přečíst z kódu, metoda vratInfo navíc používá metodu getDatumNarozeni. Ta ovšem ještě není naprogramovaná. Že by se mi mohla hodit jsem zjistil až během práce na projektu. Atribut data narození je spjat se třídou Osoba, proto metodu getDatumNarozeni doprogramuji právě zde:



Pokud chcete, můžete novou metodu getDatumNarozeni volat i z metody vratInfo třídy Zak. Doposud tomu tak nebylo. Stačí poupravit původní kód:



Třídu Pedagog a její metody máme naprogramovány. Pojďme nyní ověřit funkčnost stávajícího kódu. Vytvoříme nového pedagoga, přidáme mu 2 aprobace a vypíšeme o něm informace na konzoli.



Bystřejším z Vás jistě neušlo, že jsme do programu nezakomponovali kód ověřující validaci rodného čísla u pedagoga. Aby byla implementace efektivně vložena, musel by se nový pedagog ověřovat na úrovni třídy Škola. Tak, jako se žák ověřuje na úrovni třídy Třída. Neboť třídu Škola naprogramovánu nemáme, validaci u pedagogů vynecháme. Dalším problémem je položka týkající se pohlaví, kterou třída Pedagog neobsahuje. Bez této proměnné není možné celkovou validaci provést. Pokud by přesto klient funkcionalitu vyžadoval, není problém náš příklad poopravit a mírně rozšířit. Případnou implementaci necháme na čtenáři.

Pojďme ještě naposledy zasáhnout do našeho kódu. Pokud zavzpomínáme, novou školní třídu jsme tvořili pomocí čísla ročníku a textové proměnné nazvanou třídní. Nyní můžeme tento text vhodněji nahradit právě objektem typu Pedagog. Pojďme se podívat, jak to ovlivní náš kód:



Máme naprogramovánu třídu, pedagoga i žáky. Můžeme vyzkoušet, jak program funguje. Jak spolu jednotlivé třídy kooperují.

Pozn.: Aby program nevypisoval nadbytečné informace, v příkladu níže jsou použita pouze validní rodná čísla:

