# Dědičnost

Je čas přidat do našeho projektu třídu Pedagog. Mezi její atributy by mohly patřit jméno, příjmení, titul, rodné číslo, datum narození a seznam aprobací.

Z pohledu našeho programu vidíme styčné body mezi objekty typu Zak a Pedagog. Jedná se o atributy jméno, příjmení, rodné číslo a datum narození. Abychom předešli duplikování kódu v programu, můžeme použít dědičnost. Zde uvádím jedno z možných řešení:

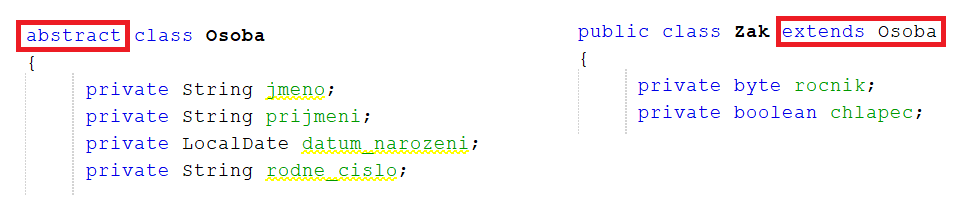
V tomto řešení je třída Osoba brána jako předek. Třídy Zak a Pedagog mají roli potomků. V rámci dědičnosti tedy platí, že potomek vždy dědí veškeré atributy předka, zároveň je však o něco rozšířen.

Můžete namítnout, že objekt třídy Osoba nebude v našem příkladu nikdy vytvořen. Má smysl tvořit pouze žáky či pedagogy. Je to pravda. Třída Osoba je pro nás vlastně jakousi pomocnou konstrukcí. Proto, pokud neplánujeme z naší třídy vytvářet instance, lze ji označit klíčovým slovem abstract. Nikde není napsáno, že hierarchie dědičnosti musí obsahovat abstraktní třídu. Nemusí. V našem případě je to ale vhodné, proto abstraktní třídu použijeme.

## Dědický vztah Osoba – Žák

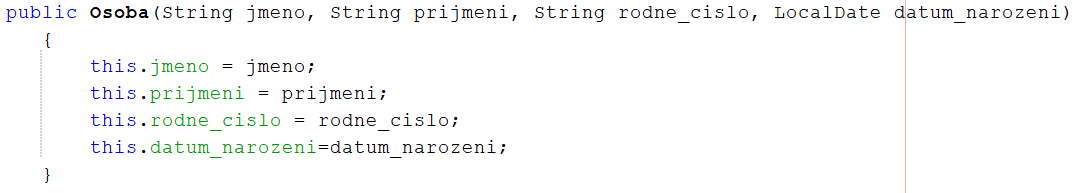
Dědičnost jsme vzhledem k posloupnosti budování učiva doposud v našem programu nepoužili. Čeká nás nyní tedy nepříjemný úkon – rozbít již existující definici třídy Zak na dvě třídy. V praxi bychom si výhody použití dědičnosti jistě uvědomili již během návrhu programu a tomuto procesu se tak vyhnuli:

* Vytvoříme novou třídu Osoba a v ní privátní atributy dle návrhu výše.
* Třídu Osoba označíme klíčovým slovem abstract.
* V definici třídy Zak pomocí klíčového slova extends uvedeme předka – třídu Osoba.
* Ze třídy Zak odstraníme atributy, které dle návrhu výše do ní nepatří, tyto přesuneme do třídy Osoba.
* Metody get a set vztažené k atributům ve třídě Osoba rovněž přesuneme ze třídy Zak do třídy Osoba



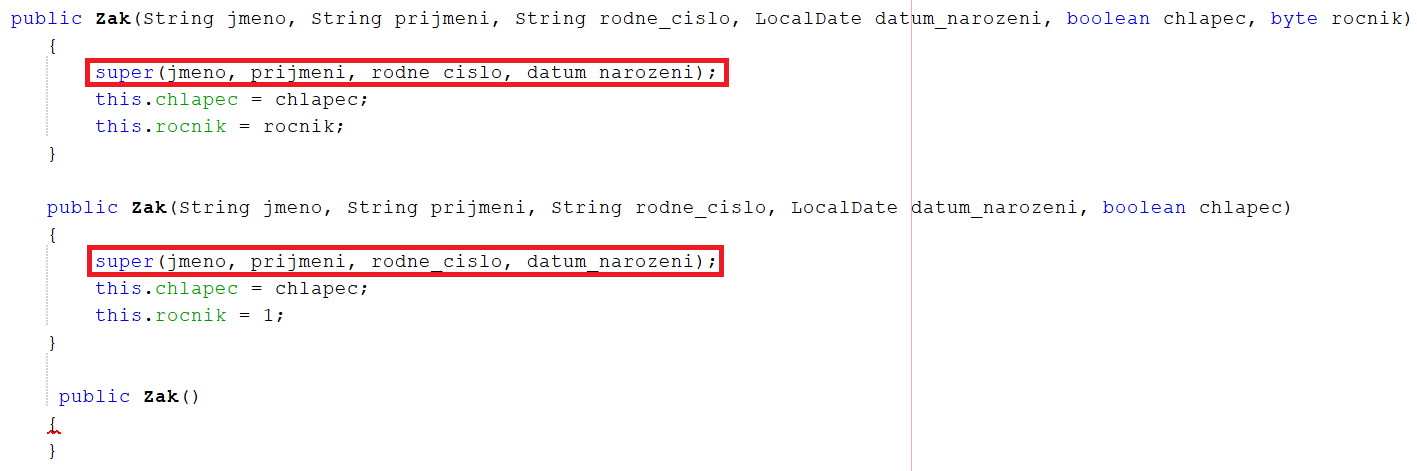
## Konstruktory v rámci dědičnosti

Naprogramovat konstruktor třídy Osoba není problém. Osoba má 4 proměnné, u konstruktoru tedy použijeme 4 parametry:



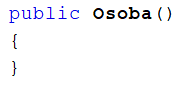
Zajímavější na celé věci je, že tento konstruktor nelze zavolat – není možné vytvořit instanci třídy **Osoba**, a to přesto, že konstruktor je public. Vysvětlení je jednoduché. Výše jsme třídu označili jako abstraktní. Právě proto, aby nebylo možno z ní tvořit instance. Konstruktor je zde pouze pro potřeby potomků dané třídy, kteří budou moci tuto metodu použít díky zákonitostem dědičnosti.

Pojďme se podívat, jak se prakticky změní konstruktor(y) u poděděné třídy Zak. Změna ovlivní všechny přetížené konstruktory:



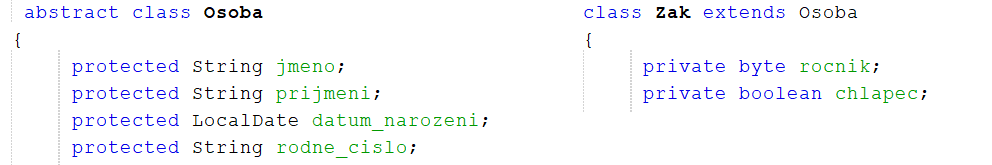
Jak je z obrázku patrné, v hlavičce konstruktorů uvádíme veškeré pro nás důležité proměnné. Změna je až v těle konstruktoru – na začátku vidíme řádek označený klíčovým slovem super. Pomocí něj v podstatě voláme konstruktor předka, tedy konstruktor třídy Osoba. Konečně jej tedy využíváme. Při tvorbě objektu typu Zak si odskočíme do třídy Osoba, která nám s konstrukcí částečně pomůže. Atributy nacházející se striktně v objektu Zak si třída musí vlastním konstruktorem obstarat sama.

Z obrázku výše je dále patrná chyba v prázdném konstruktoru třídy Zak. Problém je, že předek (třída Osoba) prázdný konstruktor neobsahuje. Nepřipouští tedy možnost vytvořit objekt bez plnění proměnných. Řešení je jednoduché. Do třídy Osoba doprogramujeme přetížený prázdný konstruktor.



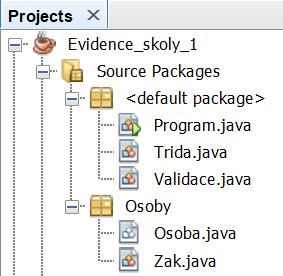
Konstruktory máme vyřešené, nicméně v programu se objevilo více chyb, které zde dříve nebyly. Konkrétně třída Zak přímo nevidí atributy ze třídy Osoba. Nemůže s nimi pracovat ve svých metodách. Není se čemu divit. Proměnné ve třídě Osoba mají modifikátory přístupu nastaven jako private. Jsou zapouzdřeny v objektu a nelze s nimi aktuálně pracovat jinak, než pomocí metod get a set. Situaci bychom mohli vyřešit přepisem modifikátoru z private na public. Jak jsme však dříve zmínili, tato změna může přinést další komplikace.

Existuje ještě jedna možnost – použití modifikátoru přístupu protected. Ten zpřístupní atribut třídám v rámci dědičnosti.



To ale samo o sobě nestačí. Abychom zamezili situaci, že na proměnné přímo uvidí například třída Program.java, je nezbytné navíc umístit dědické třídy do vlastního balíčku.

V rámci našeho projektu vytvoříme balíček a nazveme jej Osoby. Následně do něj přemístíme jak třídu Osoba, tak třídu Zak. Tímto je situace vyřešena. Děděné třídy vidí na své atributy, pro třídy mimo tento balíček jsou však proměnné neviditelné. Jsou přístupné pouze přes naše metody get a set. Práce s nimi je tedy plně v našich rukou. Co povolíme a co naopak zakážeme je na nás.



V poslední kapitole dokončíme třídu Pedagog a program otestujeme.