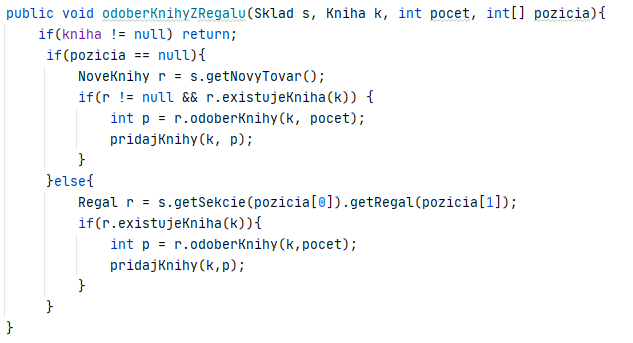
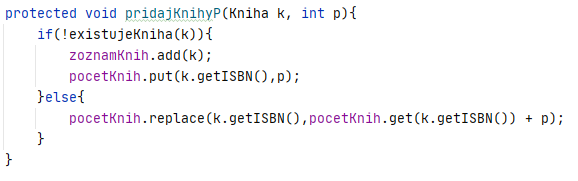
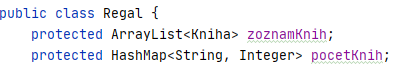
# Princípy OOP

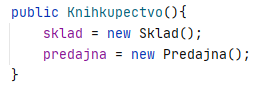
**Asociácia –** používam hlavne v triedach používateľov, menovite v **skladnik.java**, kde veľa funkcií vyžaduje objekt **sklad** a **kniha** na správne fungovanie. Príklad funkcie z triedy **skladnik.java**



**Agregácia –** tento princíp sme použili pri spôsobe ukladania kníh v **regáli** (**Regal.java**)a **palete (NoveKnihy.java). Knihy** nie sú súčasťou jedného regálu ale môžu sa premiestňovať po všetkých regáloch a navyše **sklad.java** pracuje s katalógom **kníh**, kde sú uložene referencie na knihy.



**Kompozícia –** je súčasťou kníhkupectva ako celku, pretože **kníhkupectvo (knihkupectvo.java)** by nemalo existovať bez toho aby v sebe nemalo **sklad (sklad.java)** a **predajňu (predajna.java)**. A zase **sklad** ani **predajna** by nemala existovať bez knihkupectva. Tento vztah je aj medzi **skladom** a **sekciami**; a ešte medzi **sekciami** a **regalmi.**



**Dedenie –** umožňuje, aby funkcionalita z jednej triedy bola súčasťou druhej. Tento vzťah sme uplatnili medzi **Miestnost.java** a**Sklad.java/Predajna.java**. A medzi **Regal.java** a **NoveKnihy.java,** ktoré obidve slúžia na ukladanie kníh určitou formou. **Viacnásobne dedenie** sme využili pri triedach používateľov, kde napr. **skladník** dedí atribúty a funkcie **zamestnanca** a ten zase dedí **používateľa**, ktorý je ešte dedení zákazníkom.





Pretože niektoré triedy nepotrebujú aby sa dali vytvoriť ich inštancie, napr. **Pouzivatel.java, Zamestnanec.java** a **Miestnost.java**, tak na tieto triedy sme použili **abstrakciu**.

Odvíja sa nám z toho **Polymorfizmus** typu **Upcasting,** kde mám premennú typu **Pouzivatel** používam v **Knihkupectvo.java** vo funkcií **main.** Táto premenná ukazuje na prihláseného používateľa a spúšťa funkciu na **spracovanie vstupu**.



...



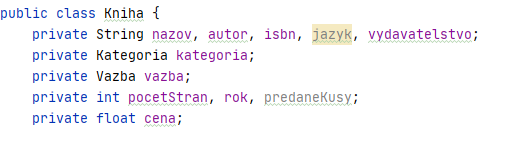


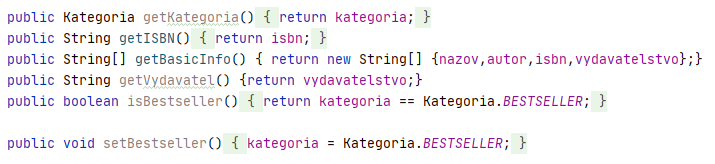
**Abstraktná metóda –** využitá v abstraktnej triede **Miestnost.java,** kde každá trieda ktorá dedí by si mala, ak chce metódu používať **init(int)**, by si ju mala sama implementovať.



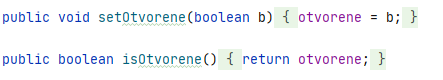
Trieda i Miestnost.java

**Enkapsulácia –** všetky nestatické atribúty sú **private**, prípadne ak sa v triede používa atribút svojej rodičovskej triedy, tak v rodičovskej triede je tento atribút **protected.** Skoro všetky tieto atribúty majú nastavení **getter** ale iba niektoré majú nastavene **setter**.



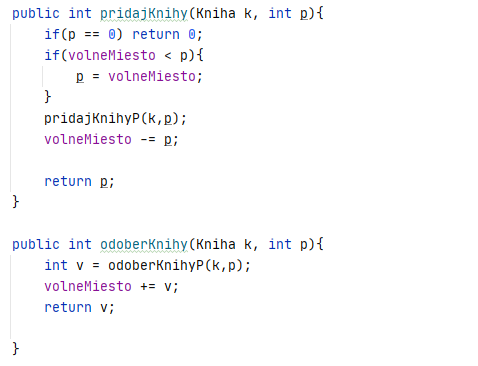


Trieda ii Kniha.java

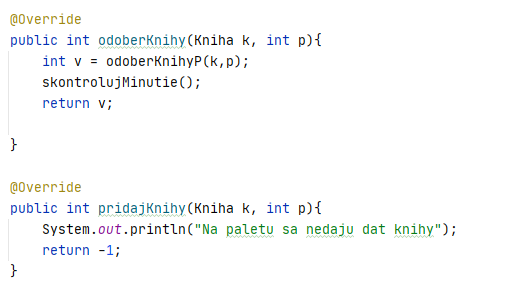


Trieda iii Predajna.java

**Overriding –** tento princíp nám dovoľuje z dedených tried zobrať funkcie a upraviť ich na nami žiadanú funkcionalitu. Využil som ho hlavne v dedeniach triedy **používateľ**, kde sa nachádza funkcia **spracuj**, ktorá spracováva vstupné príkazy. Ďalej som ho použil v triede **NoveKnihy.java**, ktorá dedí triedu **Regal.java** , tak že funkcia **odoberKnihy** namiesto uvoľňovania miesta kontroluje či ešte zostali knihy v **NoveKnihy.**

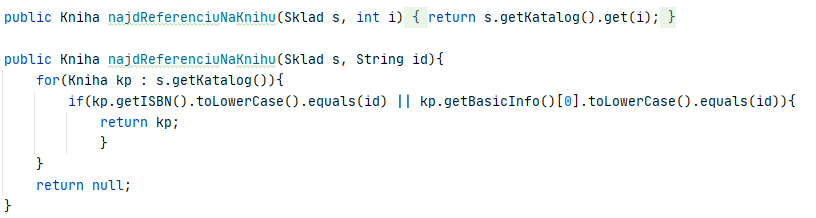


Trieda iv Regal.java



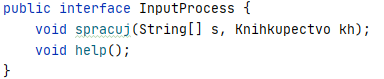
Trieda v NoveKnihy.java

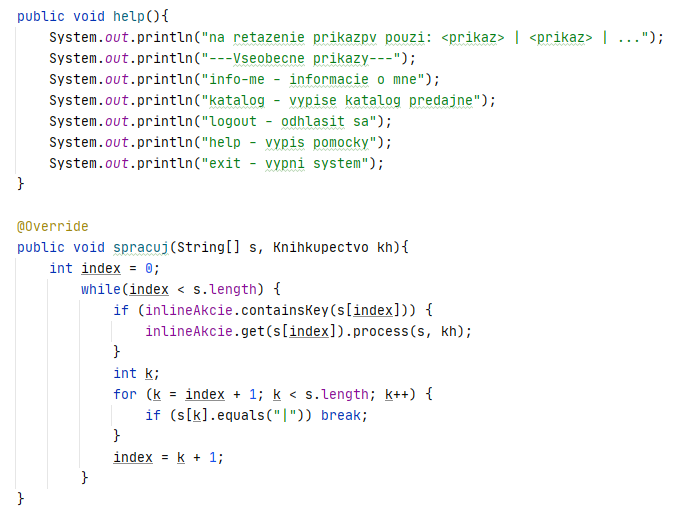
**Overloading –** príklad overloading-u mám v triede **skladnik.java**, kde skladník môže nájsť referencie na **knihu** pomocou reťazca znakov alebo podľa katalógového čísla. Využil som aj preťažovanie konštruktorov; hlavne v **Sekcia.java**





**Interface –** slúži ako predpis čo by mali triedy ktorého implementujú vyzerať. Tento princíp máme v používateľoch, kde najvyššia trieda **Pouzivatel.java** implementuje rozhranie **InputProcess.java** diktujúce, že triedy musia implementovať funkcie **spracuj(…)** a **help()**.





Trieda vi Pouzivatel.java

**Statické atribúty –** tento druh atribút som použil v práci s predefinovanou veľkosťou **predajne, skladu, sekcii** a aj **regálov**, ale aj na zistenie či sa užívateľ od posledného vykonania zmenil a nachádza sa v triede **Knihkupectvo.java.**

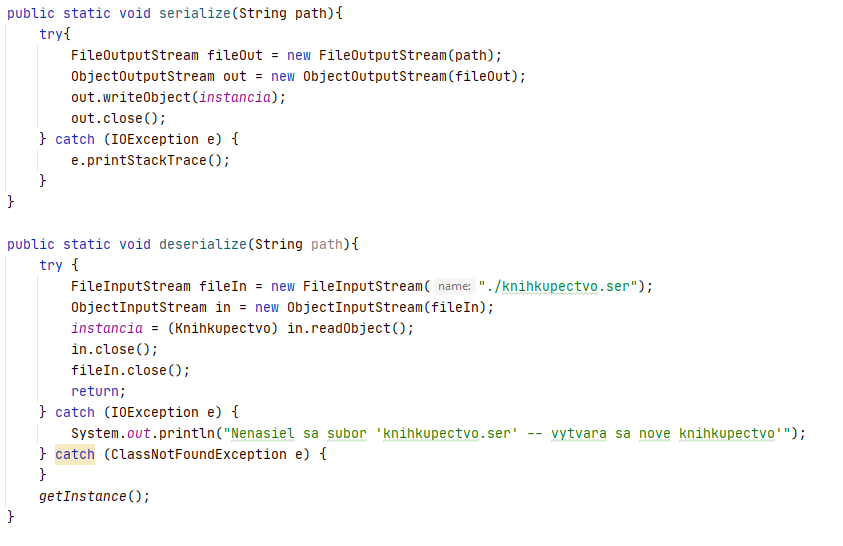


Trieda vii priklad statickej atributy z Miestnost.java



Trieda viii staticky atribut na zmenu pouzivatela z Knihkupectvo.java

**Statické metódy –** statické metódy používame v **Knihkupectvo.java** na serializáciu a deserializáciu dát zo súboru *knihkupectvo.ser*



**Finálne atribúty –** sú v programe použité na vzťah kompozície v **Knihkupectvo.java**, kde inštancia tejto triedy nikdy nemení **sklad** ani **predajňu,** preto tieto dve atribúty môžu byť označené ako **final**. Ale ako bolo ukázané vyššie tak používame aj statické premenné s označením **final**



Trieda ix Knihkupectvo.java



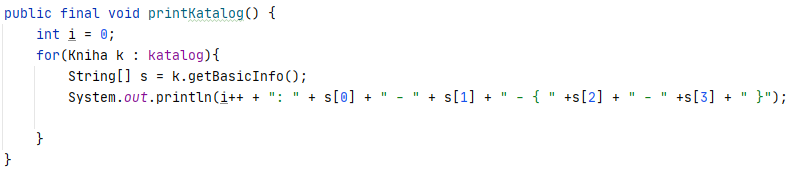
Trieda x Regal.java

**Finálna trieda –** implementovaná ako trieda **Kniha.java**, pretože každá kniha je predsa len kniha, či už je hrubá alebo tenká, takže každá kniha spadá pod túto triedu.

****

**Finálne metódy –** príkladom finálnych metód sú funkcie na ukladanie kníh v regáli v **Regal.java**, tieto funkcie sú podstate backend-om pre prekonateľné funkcie **pridajKnihy(…)** a **odoberKnihy(...)**. Jedná sa o **pridajKnihyP(...)** a **odoberKnihyP(...)**, ktoré vykonávajú logiku ukladania kníh a tá je iba jedna, a teda by žiadnym dedičom nemali byť prekonané. Ďalším príkladom je **printKatalog()** v **Miestnost.java**, kde je táto funkcia označená za final a to kvôli tomu že by nemal existovať žiadny iný spôsob výpisu atribútu **katalog**.****

Trieda xi Regal.java



Trieda xii Miestnost.java

**Singleton a privátny konštruktor –** tento návrhový model sme použili v triede **Kníhkupectvo.java**, teda môže naraz existovať iba jediná inštancia tejto triedy, teda v našom programe existuje nanajvýš jedno knihkupectvo

