Refaktorointiraportti

Ohjelmistotuotantoprojektissa loimme kokonaisvaltaisen varastonhallintatyökalun, joka on suunnattu tietokonekaupan varastonhallintaan. Sovellus kasvoi valtavasti projektin aikana ja kasvun yhteydessä projektiin kertyi erilaisia “Bad Smellejä” ja sekalaista koodia, kuten englannin- ja suomenkielistä koodia sekaisin ja nämä tarvitsi refaktoroida pois. Jokainen ryhmän jäsen työsti koodia omalla tavallaan, emmekä olleet määritellyt ennen projektia selkeitä sääntöjä, miten koodi pitää luoda. Tästä esitetty esimerkki kuvissa 1 ja 2.



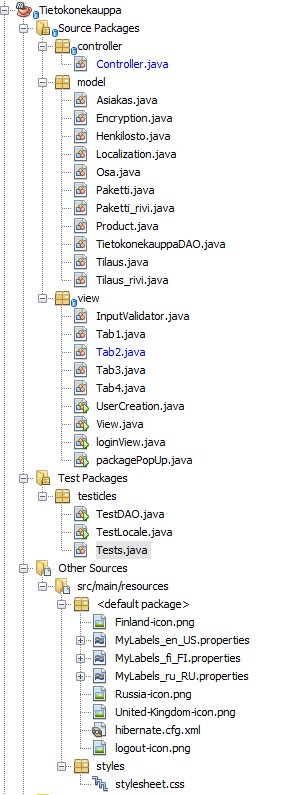
Kuva 1. Englannin- ja suomenkielen muuttujia sekaisin.



Kuva 2. Muuttujat siistitty samalle kielelle.

## Refaktorointi

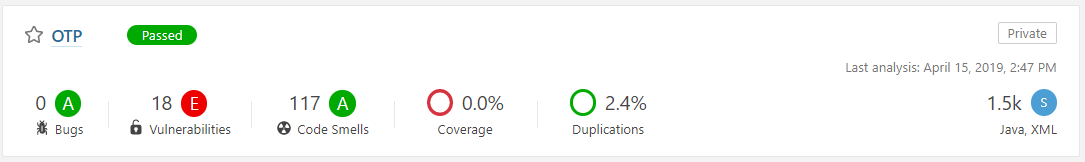
Projektin alussa sovelluksessa oli erittäin vähän tiedostoja, mutta tiedostot olivat valtavia ja erittäin sekavan näköisiä. Käyttöliittymäluokka View.java sisälsi kaikki tabien luonnit, kirjautumisen, käyttäjänluonnin ja kaikki käyttöliittymään liittyvät metodit. Tiedosto sisälsi ennen refaktorointia yli 700 koodiriviä ja sovelluksen kasvaessa se olisi laajentunut yli 3000 riviin. Tällöin sovelluksen hallinta olisi ollut mahdotonta. Käyttöliittymän arkkitehtuuri jaettiin omiin pienempiin luokkiin, joilla jokaisella on määrätty käyttötarkoituksensa. Tämä esitetty kuvassa 3.



Kuva 3. Sovelluksen kansiorakenne.

### Bad Smells ja haavoittuvaisuudet

Sovelluksesta etsittiin Bad Smells koodeja käyttäen SonarQube työkalua, joka näyttää koodin mahdolliset ongelmat. Ensimmäiset SonarQube tulokset esitetty kuvassa 4.



Kuva 4. Ensimmäiset SonarQube tulokset.

Alla listattuna toimenpiteet Bad Smellsien ja haavoittuvuuksien korjaamiseksi.

View:

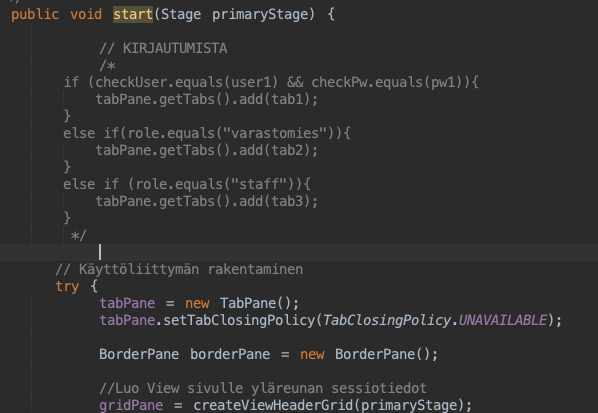
* Tab-luokista luotiin Singletoneja, jolloin luokan ilmentymän voi luoda vain kerran.
* Jokainen sovelluksen Tab-luokka perii Javan abstraktin Tab-luokan ja View.java tuntee vain tämän.
* Karsittiin turhat kommentit, duplikaatit, poistettiin aputulostukset.
* Vaihdettiin printstacktrace loggeriksi SonarQuben suosituksesta.
* Salasanan salaus toteutetaan erillisessä tiedostossa Encryption.java

Model:

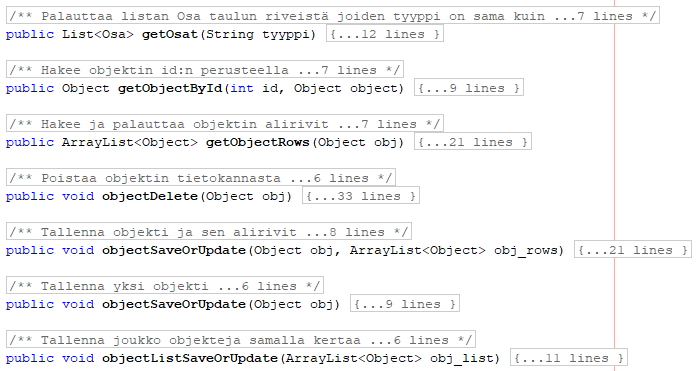
* Karsittiin turhat kommentit, duplikaatit, poistettiin aputulostukset.
* Vaihdettiin printstacktrace loggeriksi SonarQuben suosituksesta.

Controller:

* Controllerista luotiin Singleton, jolloin luokan ilmentymän voi luoda vain kerran.
* Karsittiin turhat kommentit, duplikaatit, poistettiin aputulostukset.
* Muokattiin metodit createOsa(), createPaketti käyttämään yhtä metodia objectSaveOrUpdate.



Kuva 5. Turhia kommentteja.



Kuva 6. Metodit createOsa(), createPaketti käyttämään yhtä metodia objectSaveOrUpdate.

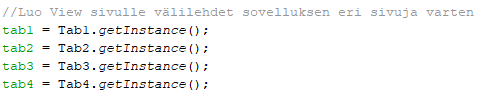
## 

## Suunnittelumallit

### Factory method

Jokainen sovelluksen Tab-luokka perii Javan abstraktin Tab-luokan, mutta annettu aliluokka päättää minkä luokan ilmentymä luodaan. Tämä esitetty kuvassa 7.

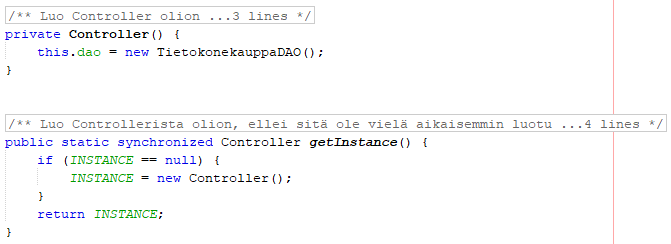




Kuva 7. Factory Method

### Singleton

Käytimme Singleton mallia tabien luonnissa ja controllerissa, jolloin luokkien ilmentymät voi luoda vain kerran. Näitä ei vahingossakaan voi luoda uudelleen new-operaattorilla, koska konstruktori on privaatti.



Kuva 8. Esimerkki Singleton mallista sovelluksessa.

### State

Emme luo omaa TableView taulukkoa osille. TableViewin tilaspesifi käyttäytyminen paikallistetaan radiopainikkeiden tiloihin. Tämä esitetty kuvassa 9.



Kuva 9. Radiopainikkeiden koodit.

# Yhteenveto

Projektin aikana opimme, että pitää sopia ennen projektia miten ohjelma ja koodi tuotetaan. Kun kaikki kirjoittavat samankaltaista koodia, tulee ohjelmasta huomattavasti helpommin hallittava ja muunneltava. Suunnittelumallien sisällyttäminen ohjelmaan tuotti hankaluuksia, sillä sovellusta oli jo rakennettu ja suunniteltu liian pitkälle ennen kuin näitä koskevat vaatimukset annettiin. Suunnittelumallit kurssin aikana opittu Composite malli olisi sopinut tietokonepaketin luontiin, mutta opimme tämän vasta kun tietokonepaketin luontikoodi oli jo valmis. Opimme kurssin aikana paljon koodin refaktoroinnista ja sen tarpeen tunnistamisesta.