

Car finder

Jani Sillanpää

Dokumentaatio

Huhtikuu 2021

Tieto- ja viestintätekniikka

Ohjelmistotekniikka

SISÄLLYS

[1 JOHDANTO 3](#_Toc69385731)

[2 README, OHJELMAN AJAMINEN 4](#_Toc69385732)

[3 OHJELMAN KUVAUS 5](#_Toc69385733)

[3.1 CarModel 6](#_Toc69385734)

[3.2 CustomProxyModel 6](#_Toc69385735)

[3.3 CostsCalculator 7](#_Toc69385736)

[3.4 FiltterInputDialog 8](#_Toc69385737)

[3.5 Car 9](#_Toc69385738)

[4 User experience 10](#_Toc69385739)

[4.1 Matti Meikäläinen 36 v. 10](#_Toc69385740)

[4.2 Timo Teininen 17 v. 10](#_Toc69385741)

[4.3 Justiina Äitinen 25 v. 10](#_Toc69385742)

[5 KEHITYSKOHTEET 11](#_Toc69385743)

# JOHDANTO

Työ on tehty 4 op mukaisesti, eli lokaalisointeja, käännöstekstejä yms. ei ole tehty. User experience osuudessa on tehty use caseja 3 kappaletta.

Alun perin työn aiheeksi valikoitui aihegeneraattorista ohjelma, jossa käyttäjä voi selata ohjelmointikieliä valitakseen yhden käyttökustannuksien perusteella. Tätä varten ohjelmassa pitäisi olla suodatusominaisuus auttaakseen käyttäjää valinnassa. Ohjelman aihetta muutettiin vaihtamalla ohjelmointikielet autoiksi, milloin ohjelman kontekstista tuli hieman arkipäiväisempi ja mielekkäämpi ohjelmoida.

Ohjelman datana käytetään netistä löytynyttä JSON-tiedostoa, joka sisälsi 371 auto-objektia ominaisuuksina vuosi, id, teho (hevosvoimat), valmistaja, malli, hinta ja url-osoite kuvaan.

Ajatuksena oli tehdä näkymä, jossa listatut autot näkyvät. Painikkeesta aukeisi dialogi, jossa voisi asettaa suodatusasetukset. Auton käyttökustannuksia pääsisi laskemaan valitsemalla näkymästä haluttu auto (rivi) ja painaa painiketta, milloin aukeisi oma dialogi kyseisen auton käyttökustannuksien laskemiselle.

# README, OHJELMAN AJAMINEN

Pura pakattu tiedosto, avaa projektitiedosto Qt Creatorilla, käännä.

Halutessa GitHub: <https://github.com/Jantero93/Qt_Harkka>

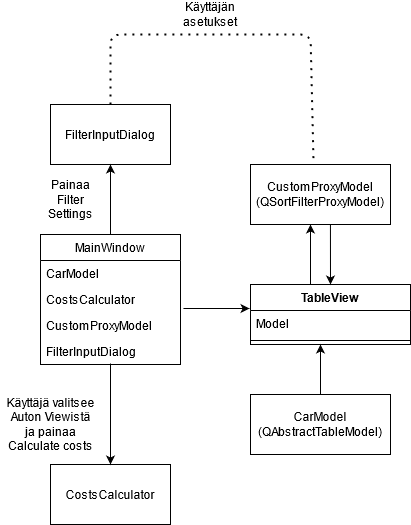
Ohjelma kääntyy kurssin ohjeiden mukaan asennetulla Qt Creatorilla ongelmitta. Projektikansion juuressa oleva CARS\_data.json täytyy siirtää build-kansioon, jotta ohjelma pystyy lataaman testidatan.

# OHJELMAN KUVAUS

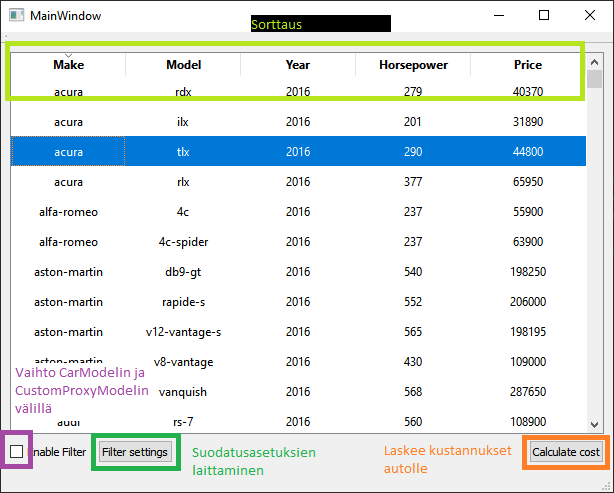
Kuvassa 1 ohjelman hierarkiasta karkea kuvaus kaaviona. MainWindowissa hallitaan ohjelman eri komponenttien muodostamaa kokonaisuutta. MainWindowissa sijaitsee TableViewi, jossa modelina käytetään CarModelia (koko data) tai suodatettua osaa, minkä modelin tarjoaa CustonProxyModel.

Filter Settings -painiketta aukeaa dialogi, jossa käyttäjä asettaa haluamansa rajausvaihtoehdot. MainWindowissa data haetaan getterillä ja asetetaan setterillä CustomProxyModelille.

Calculate costs -painike on valittavaksi, kun TableViewistä on valittu jokin rivi (auto). Painiketta painaessa aukeaa dialogi, jossa voi laskea auton kustannuksia vuositasolla. MainWindow huolehtii autodatan hakemisen valitulta modelilta ja tarjoaa sen CalculateCostille setterillä.



KUVIO 1. Ohjelman hierarkian kuvaus



KUVA 1. MainWindow kuvakaappaus

## CarModel

CarModel huolehtii koko datan näyttämisestä TableViewissä. Luoka on tehty pitkältä samalla tavalla kuin tunnilla näytetyssä esimerkissä. Luoka on periytetty QAbstractTableModelista. Luokasta on korvattu (override) metodit headerData, data, rowCount, columnCount ja sort.

headerData luo Viewille sarakkeille otsikot, auton ominaisuudet. Data huolehtii viewille vietävästä dataasta. Sortti mahdollistaa Viewin järjestämisen auton ominaisuuksien mukaan.

## CustomProxyModel

CustomProxyModel voi asettaa Viewille muokatun modelin toisesta modelista, tässä työssä CarModelista. Luokka periytyy QSortFilterProxyModelista ja luokasta on korvattu metodit filterAcceptsRow ja headerData.

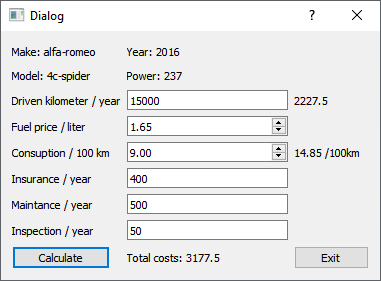
HeaderData huolehtii Viewille sarakkeiden otsikoista samaan tapaan kuin CarModelissa. filterAcceptsRow-funktolla voidaan määritellä, mitkä source modelin rivit näytetään Viewissä.

Funktiosta palautetaan true, mikäli rivi halutaan näyttää. Funktiossa vertaillaan liuta ominaisuuksia ja palautetaan vertailujen perusteella true/false.

## CostsCalculator

CostsCalculator asettaa auton tiedot yläosaan, jotka MainWindow on setterillä asettanut. Tämän jälkeen käyttäjä voi laskea auton kustannukset erilaisilla muuttujilla.

Käyttäjälle tulostuu annettujen ajokilometrien viereen summa, joka aiheutuu pelkästään polttoainekustannuksista. Myös polttoaineen kulutuksen viereen tulostuu summa, joka kuvaa kustannuksia per ajettu sata kilometriä. Alas tulostuu kokokustannukset mukaan lukien polttoaine-, vakuutus-, ylläpito- ja katsastuskustannusket.



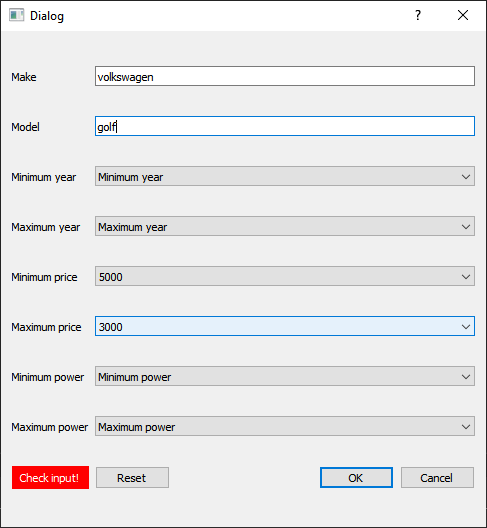
KUVA 2. Esimerkki laskelma CostsCalculatorilla

## FiltterInputDialog

FillterInputDialogissa käyttäjä voi asettaa haluamansa suodatuskriteerit. Suurin osa kriteereistä on kiinteitä arvoja Comboboxeissa, jotka tuntui parhaimmalta ratkaisulta, kun haluttiin käytettävyyttä, mutta kumminkin pitää virhesyötteet mahdollisimman vähäisinä.

Ristiriitaisissa asetuksissa, esimerkiksi minimihinta on suurempi kuin maksimihinta, käyttäjälle tulostetaan label ”Check input!”. Käyttäjää ei estetä etenemästä eteenpäin valitsemillaan asetuksilla ja ohjelma ei kaadu siihen. CustomProxyFiltteri ei vain anna yhtäkään riviä, koska sen filterAcceptsRow-funktio ei koskaan palauta true.

Käytettävyyden lisäämiseksi käyttäjälle tarjotaan Reset-nappi, jolla voi nopeasti tyhjentää kriteerit. Jos Comboboxissa on ominaisuuden nimi, esimerkiksi ”Minimum year”, CustomProxyModel ei huomioi sitä suodattaessa rivejä-



KUVA 3. FilterInputDialog kuvakaappaus

## Car

Auto-datalle on tehty oma luokka, joka on melko triviaali. Toinen vaihtoehto luokan sijaan olisi ollut structi, mutta luokkaan päädyttiin, koska haluttiin kuormittaa (overload) operaattori = . Tämä varmaankin olisi ollut myös mahdollista structille, mutta koodin kirjoitus hetkellä ei tullut tätä mieleen tarkistaa.

Luokalle on tehty oletus- ja parametrirakentaja. Mutkia on hieman oiottu, joten luokan attribuutit ovat julkisia, näihin ei uudelleensijoiteta ohjelman aikana.

# User experience

Ohjelmaa varten on kehitetty muutama erilainen käyttötapaus (use case). Käyttötapausta varten identifioidaan käyttäjä, mitä käyttäjä haluaa tehdä sovelluksella, mitä käyttäjälle tapahtuu ohjelman ajon aikana, miten käyttäjä saavuttaa tavoitteensa.

## Matti Meikäläinen 36 v.

Matti haluaa ostaa uuden auton säästöillään, mutta hän ei tiedä minkälaisia autoja on hänen haluamallaan hintahaitarilla. Matilla lähenee keski-iän kriisi, joten mahdollisimman tehokaskin auto tarvitsisi saada. Matti asettaa suodatinasetuksista haluamansa hinta-alueen, jolloin hänen on nopeampi selata sopivia autoja.

## Timo Teininen 17 v.

Timo on juuri saamassa ajokorttia, mutta hänellä ei ole autoa vielä. Hänellä ei ole paljon säästöjä, joten hän haluaa halvan auton. Timo on erinomaisen fiksu nuori, koska hän tietää, että halvat autot ovat tehottomia ja vanhoja, mutta kääntöpuolena vakuutukset saattavat olla kalliimpia.

Timo suodattaa haluamansa autot ja laskee eri autoilla paljonko käyttökustannuksiin kuluisi rahaa.

## Justiina Äitinen 25 v.

Justiina on juuri saanut esikoisen. Hän tarvitsee mahdollisimman uuden, turvallisen auton. Hän asettaa suodatinasetuksista mahdollisimman uudet autot ja ylärajan hinnalle. Hän laskee vielä käyttökustannuksia vuositasolla varmistukseen, että on varaa pitää autoa.

# PROSESSI, JOHTOPÄÄTÖKSET, POHDINTA

Työtä oli melko suoraviivaista tehdä ja juurikaan suurempia ongelmia ei ollut.

Eniten aikaa vei dokumentaation QSortFilterProxyModelin dokumentaation lukemisesta ja tästä oman luokan toteuttamisesta. Myös halutun datan hakeminen CarModelista CustomProxyFilterin rivin indeksillä vei aikaa kiitettävästi.

Ohjelmaa koodatessa on pyritty käyttämään mahdolisimman paljon Qt:n omia luokkia ja toteutuksia ”ja ajattelemaan Qt:isesti”. Hirveesti en muuttaisi mitään, jos nyt lähtisin koodaamaan ohjelmaa uudelleen. Ehkä TableViewillä en vaihtelisi modelia CarModelin ja CustomProxyModelin välillä, vaan tekisin CustomProxyModeliin booleanin näytetäänkö filtteröity tieto vai koko model. Tämän jälkeen ei tarvitsi TableViewillä vaihdella modelia.

Graafiseen ulkoasuun ei hirveästi ole panostettu, labeleissa on oletusfontit, koot jne. Numeroissa olisi hyvä olla tuhateroitin tuomaan selkeyttä numeroiden kokoluokkaan. Paino on ollut enemmän toimintalogiikan rakentamisessa ja GUI hiomiseen ei jäänyt niin paljoa aikaa.