Avansert Javaprogrammering 2 - Innlevering 1

Eivind Vegsundvåg(vegeiv13)

Innledning

Løsningen er implementert grafisk ved hjelp av JavaFX og benytter seg av 3 kontrollerklasser, hvorav én er en tjeneste/kontroller-hybrid, 3 oppgaveklasser, og tre stilsatte visninger(disse finnes i resources). Hovedvisningen i løsningen består av fire felter: ett for hver status i tjenesten som representerer en klient grafisk, og ett felt for å fremvise utleiebiler samt status for disse. Kundebokser og bilbokser er å ansé som en egen undervisning.

Jeg har valgt å bruke JavaFX-rammeverket DataFX for enkel serieinstansiering av visninger og kontrollere. De feltene som er annotert med ActionTrigger, ActionMethod og FXMLViewFlowContext er å ansé som DataFX-komponenter.

Løsningen er event-driven ved hjelp av JavaFX8. Den bruker Properties for å oppdatere verdier uten egen kode for formålet.

Testdekning for løsningen er omtrent 99% blant de klassene som ikke er DataFX-kontrollere, og 35% innenfor kontrollere. Dessverre er ikke hovedkontrolleren testet, ettersom denne isåfall burde vært integrasjonstestet, og DataFX ikke har støtte for testing i JavaFX for Java8.

Forutsetninger

Løsningen leveres som et Maven-prosjekt, og er avhengig av at prosjektet importeres under denne modellen i utviklingsmiljøet. Brukte versjon av DataFX støtter ikke pakking til eksekverbar .jar fil, og løsningen kan utelukkende kjøres i utviklingsmiljø.

Trådhåndtering

Trådene i løsningen er delvis abstrahert bort. Klassen ClientService utvider klassen javafx.concurrent.Service, som bruker trådeksekveringen til JavaFX uten eksplisitte kall til dette.

Metoden CarRental.rentCar(Client) setter tråder på vent frem til en bil er ledig, og metodene CarRental.returnCarByClient(Client) og CarRental.addNewCar() signaliserer når en bil er tilgjengelig.

Notat til student som retter oppgaven

Løsningen løser alle krav om tekstuell output ved hjelp av grafiske elementer. Det er derfor kun feilmeldinger som forekommer på System.out. Løsningen kan derfor stå frem som mer kompleks enn den er. Jeg anbefaler derfor å se nøye over pakkestrukturen og klassedokumentasjonen nedenfor før man setter seg inn i løsningen.

Ellers kan det være greit å se på konstruktøren og post-konstruktøren til MainController som inngangspunkt i programflyten, og være klar over følgende:

- MainController setter opp én ClientService per Client. Disse er også grafisk fremstilt som én kundeboks med navn og progress bar
- ClientService er i stor grad event driven. Hvert objekt representer også én tråd. Den starter med å fyre av én ClientTask, og restarter seg selv etter hvert som disse er ferdig.
- CarController forsøker kun å oppdatere bilene etter hvert som de leies.

Klassedokumentasjon

Alle klasser er nøye dokumentert med JavaDocs. Klassen CarRentalApplication setter opp JavaFX-miljøet og åpner et vindu. Formålet med klasser oppsummeres her:

Package cars

Denne klassen inneholder klasser direkte relatert til leiebilene i løsningen.

CarFactory

Denne klassen oppretter biler på kommando, med nummerskilt i et mønster angitt. Det er mulig at den pseudotilfeldige generatoren oppretter identiske biler.

CarFactoryException

Denne feilmeldingen kastes hvis man sender inn ugyldig informasjon ved opprettelse av en instans av CarFactory

CarRental

Denne klassen oppretter en bilfabrikk som genererer biler etter et gitt mønster. Dette er litt dumt: det er sannsynligvis å foretrekke at man helder sender med ferdig bilfabrikk. Dette løses før mappelevering. Instanser av CarRental sørger for en trådsikker måte å leie og levere inn biler.

RentalCar

Denne klassen er kun en verdiholder, og inneholder et nummerskilt og en kunde. Hvis kunden er null, er bilen å anse som leibar.

Package clients

Denne klassen inneholder klasser direkte relatert til kunder.

Client

Denne klassen er kun en verdiholder. Logikk rundt bruk av Clients finnes i ClientService og i pakken clients.concurrent. Klassen inneholder et navn og en status.

ClientState

Den enumeratoren brukes til å representere leiestatus for et kundeobjekt. Statusen READY antyder at kunden skal vente med å leie en bil, WAITING antyder at kunden venter på å få leie en bil, og RENTING antyder at kunden leier en bil.

Package clients.concurrent

Denne pakken inneholder klasser tilknyttet logikk rundt leie av biler av kunder.

ClientTask

Denne abstrakte klassen definerer objektene som skal være tilgjengelig for alle ClientTasks, samt en enkel overstyring av metoden Runnable.run() som implementerer avventing på Phaser-objekt.

ReadyTask

Denne klassen er en utvidelse av ClientTask og har i oppgave å vente en viss periode mens den oppdaterer en progress bar ved hjelp av sin egen fremdrift.

WaitingTask

Denne klassen er en utvidelse av ClientTask og har i oppgave å sikre en leiebil.

RentingTask

Denne klassen er en utvidelse av ClientTask og har i oppgave å vente en viss periode mens kunden leier en bil, og oppdaterer en progress bar ved hjelp av sin egen fremdrift.

Package controllers

CarController

Denne klassen støtter opp en visning for en bil. Den sørger for å knytte sammen strengegenskaper slik at oppdateringer for en bil gjenspeiles i brukergrensesnittet.

ClientService

Denne klassen støtter opp en visning for en kunde, samtidig som den oppretter nye ClientTasks for å drive videre leie.

MainController

Denne klassen viser alle andre visninger i en hovedvisning, og plasserer kunder på riktig sted etter hvert som de går videre i sekvenser.

Oppsummering

Alt i alt er jeg svært stolt av denne løsningen. Den er event driven, viser styrken til Java8, og møter oppgavekravene på en kreativ måte. Bruk av DataFX har gjort det svært enkelt å sørge for grei arkitektur, slik at all koden er kort og konsis. Manglende testdekning i MainController er litt dumt, men forsøk på å instansiere denne utenfor Flow-APIet til DataFX resulterer i feilmeldinger. Ettersom visningsnøstingen ikke er utelukkende Flow, virker det derfor problematisk å innstille seg på kun Flow-miljø i testingen.