OOPD - Lab 4

Uppgift 2: Model-View-Controller

Vilka avvikelser från MVC-idealet kan ni identifiera i det ursprungliga användargränssnittet?

I det ursprungliga användargränssnittet finns det ett direkt beroende mellan CarView och CarController i CarView. Denna koppling mellan CarView och CarController bör inte finnas. Detta eftersom enligt MVC ska viewn enbart notifiera controllern om användarens input och sedan ska controllern i sin tur manipulera modellen. Eftersom viewn enbart ska notifiera controllern om användarens input passar inte de action listerners som använder sig av CarController in i view utan bör ligga i controller enligt idealet.

I CarController finns också flera metoder som inte passar in i en Controller. Dessa påverkar bland annat Vehicle direkt och passar bättre att ligga i modellen enligt MVC-idealet.

Vad borde gjorts;

Smartare modell:

Detta kan åstadkommas genom att bland annat flytta klassen TimeListener och andra metoder från Controller till Model som nämns nedan. Detta för att det ska bli tydligt vad som är modellen och att det ska bli enkelt att interagera med endast en

Dummare view:

En dummare view kan fås genom att flytta ut de action listeners som har med button att göra till CarController. Övrig kod som berör knapparna kan också läggas i controller eftersom…?

Tunnare controller:

För att få en tunnare controller behöver vi flytta den nästlade klassen TimeListener till en egen klass. Förslagsvis till en klass som kallas för “Model” som är helt oberoende av användargränssnitt. Även de metoder som tillhör TimeListener och som hanterar logik vid kollisioner bör flyttas till klassen Model. Detta för att CarController endast ska hantera användarens input och styra modellen utifrån detta, samt uppdatera den. Även de metoder som direkt påverkar olika Vehicles bör ligga i Model istället för i Controller för att skapa en tunnare controller. Istället bör det bara ligga metoder i Controller som anropar metoderna i Model som påverkar Vehicle.

Vilka av dessa brister åtgärdade ni med er nya design från del 3? Hur då? Vilka brister åtgärdade ni inte?

De brister som vi åtgärdade i labb 3:

Vi uppdaterade CarController så att den inte längre hade main i sig, utan skapade en egen main klass

Vi ändrade så att DrawPanel blev våran View och ändrade CarView till UserInterface eftersom det är DrawPandel som beter sig som en view.

*De brister som vi **inte** åtgärdade i labb 3:n*
Det vi inte åtgärdade var att flytta klassen Timelistener och andra metoder från CarController till en ny klass Model.

Uppgift 3:

Observer: Vi har i nuläget inte använt oss av *observer pattern*. Vi har dock smått börjat på en viss implementation av *observer* genom att ha classen *TimeListener*. Klassen registrerar händelser (ex. en kollision) och gör lämplig handling vid en händelse, den meddelar dock inte modellen om vad som skett. Här skulle vi kunna lägga till en interface som sedan implementeras av CarModel samt View. Syftet är att vi sedan i Model kan, mha. interfacet, uppdatera view(läs repaint) så fort modellen notifierats av event i actionListener. Vi planerara att implementera ett Observer pattern.

Factory Method: Finns för nuvarande inte någon *factory method*, men vi ser det som en möjlighet att implementera det i vårt program. Då vi för närvarande endast hanterar bilar och lastbilar av olika slag i modellen fungerar det att vår logik i vehicle klassen. Detta blir dock problematiskt om vi någon gång i framtiden skulle vilja implementera nya former av fordon, ex. sjöfart, flygplan eller liknande. En lösning hade varit att använda oss av *factory method* och därmed kunna skapa objekt i en superklass, men låta dess subklasser ändra vilken typ av objekt som skapas. Vi vill skapa en klass *CarFactory* som ansvarar för att skapa objekt (bilar, trucks, transporter, etc.). Även detta är ett design mönster som vi tänker implementera.

State: används inte för närvarande, skulle dock gå att implementera, men känns smått onödigt. Möjligt att ett fordon har två states, “movable” eller “notMovable”. Det som skulle kunna göras är att använda sig av states här för att göra det möjligt att i framtiden kunna utöka programmet till att kunna hantera flera objekt med ännu fler states. Vi kände dock att vi inte hade tid att implementera detta så det har inte gjorts, men att göra det hade höjt kvalitén på koden och dess möjlighet att vidareutvecklas.

Composite: Detta såg vi som en möjlighet att implementera, men vi hade inte riktigt tid att genomföra implementeringen till 100%. Vi har dock börjat med en implementation i form av klassen

CarList som ansvarar för att hantera en lista av de bilar som existerar vid en viss tidpunkt i systemet. Klassens funktioner kan då hämta info om specifika objekt, lägga till bilar eller ta bort bilar.