Lab 4

### Uppgift 2: Model-View-Controller

* Vilka avvikelser från MVC-idealet kan ni identifiera i det ursprungliga användargränssnittet? Vad borde ha gjorts smartare, dummare eller tunnare?

CarController ansvarade för mycket mer än bara Controller aspekten då den bland annat hade main funktionen och ett beroende av CarView, borde gjorts tunnare. Det fanns en avsaknad av en Applikation-modell.

* Vilka av dessa brister åtgärdade ni med er nya design från del 2A? Hur da? Vilka brister åtgärdade ni inte?

Vi åtgärdade CarController genom att ta delarna som inte rörde skickandet av styrsignaler till modell / vy och satte det i Application. Vi tog även bort beroendet av CarView i CarController. I nuvarande läget så är model, controller och view separerade på ett rimligt sätt och vi ser inte ytterliggare brister utifrån MVC-pattern-perspektivet.

* Rita ett nytt UML-diagram som beskriver en förbättrad design med avseende pa MVC.

Vårt UML-diagram uppdateras inte utifrån MVC-perspektivet men vi var tvungna att implementera vissa detaljer lite annorlunda än vad som var tänkt i UML-diagrammet från Lab 3. Därmed bifogar vi ett nytt UML-diagram med några mindre justeringar.

### Uppgift 3: Fler designmönster:

* Observer, Factory Method, State, Composite. För vart och ett av dessa fyra designmönster, svara pa följande frågor:
* Finns det något ställe i er design dar ni redan använder detta pattern, avsiktligt eller oavsiktligt? Vilka designproblem löste ni genom att använda det?

Mellan truckbed och lastbilarna använder vi avsiktligt Composite design pattern. Ett state pattern är avsiktligt implementerat även i truckbed då lastbilen inte går att gasa när flaket är uppe. Lastbilens tillstånd uppdateras.

Delegeringen till truckbed gjorde att vi bättre kan följa Single Responsibility Principle och delegera allt vad gäller en truckbed till den klassen.

* Finns det något ställe dar ni kan förbättra er design genom att använda detta design pattern? Vilka designproblem skulle ni lösa genom att använda det? Om inte, varför skulle er design inte förbättras av att använda det?

Istället för en instans av Application i CarView skulle man kunna implementera ett Observer interface istället och på så sätt bättre följa Dependency Inversion Principle.

Vi skulle även kunna göra en ny klass UpdateVehicle som hanterar det mesta som händer i run-time som en ny del i våran modell och plocka ut det från Application för att ytterligare applicera DIP.

I den nya implementationen har varken UpdateVehicle eller Application nått beroende av CarView utan istället på det nya interfacet VehicleObserver.

CarController får ett nytt beroende från Application till UpdateVehicle istället eftersom det är där den ska uppdatera saker i runtime.

* Uppdatera er design med de förbättringar ni identifierat.