Uppgift 2: Model-View-Controller

Användargränssnittet ni utgick från i del A var en ansats till implementation av Model-View-Controller Pattern, men där gränsdragningen mellan model, view, controller och applikation inte var något vidare genomtänkt (för att inte säga usel).

- **Vilka avvikelser från MVC-idealet kan ni identifiera i det ursprungliga användargränssnittet? Vad borde ha gjorts smartare, dummare eller tunnare?**
- **Vilka av dessa brister åtgärdade ni med er nya design från del 2A? Hur då? Vilka brister åtgärdade ni inte?**
- **Rita ett nytt UML-diagram som beskriver en förbättrad design med avseende på MVC.**

Vår modell är smart. Alla algoritmer och beräkningar som är essentiella finns endast i den, vilket följer MVC-pattern. Controllern skall vara tunn, det vill säga att endast ta in inputs och koppla det till rätt action, inget mer än så, vilket den redan är. Möjligtvis att man skulle kunna ändra på gas- och brake-knapparna så att ingen beräkning sker i de överhuvudtaget, trots att den nu är minimal. Main-metoden, däremot, skall inte ligga i controllern, utan i en separat applikation.

Viewen skall vara “dum”, alltså inte känna till någon av de övriga modulerna i programmet direkt, dock ok att göra det indirekt via observers.

Uppgift 3: Fler designmönster

- Observer, Factory Method, State, Composite. För vart och ett av dessa fyra designmönster, svara på följande frågor:
    - Finns det något ställe i er design där ni redan använder detta , avsiktligt eller oavsiktligt? Vilka designproblem löste ni genom att använda det?
    - Finns det något ställe där ni kan förbättra er design genom att använda denna design pattern? Vilka designproblem skulle ni lösa genom att använda det? Om inte, varför skulle er design inte förbättras av att använda det?
- Uppdatera er design med de förbättringar ni identifierat.

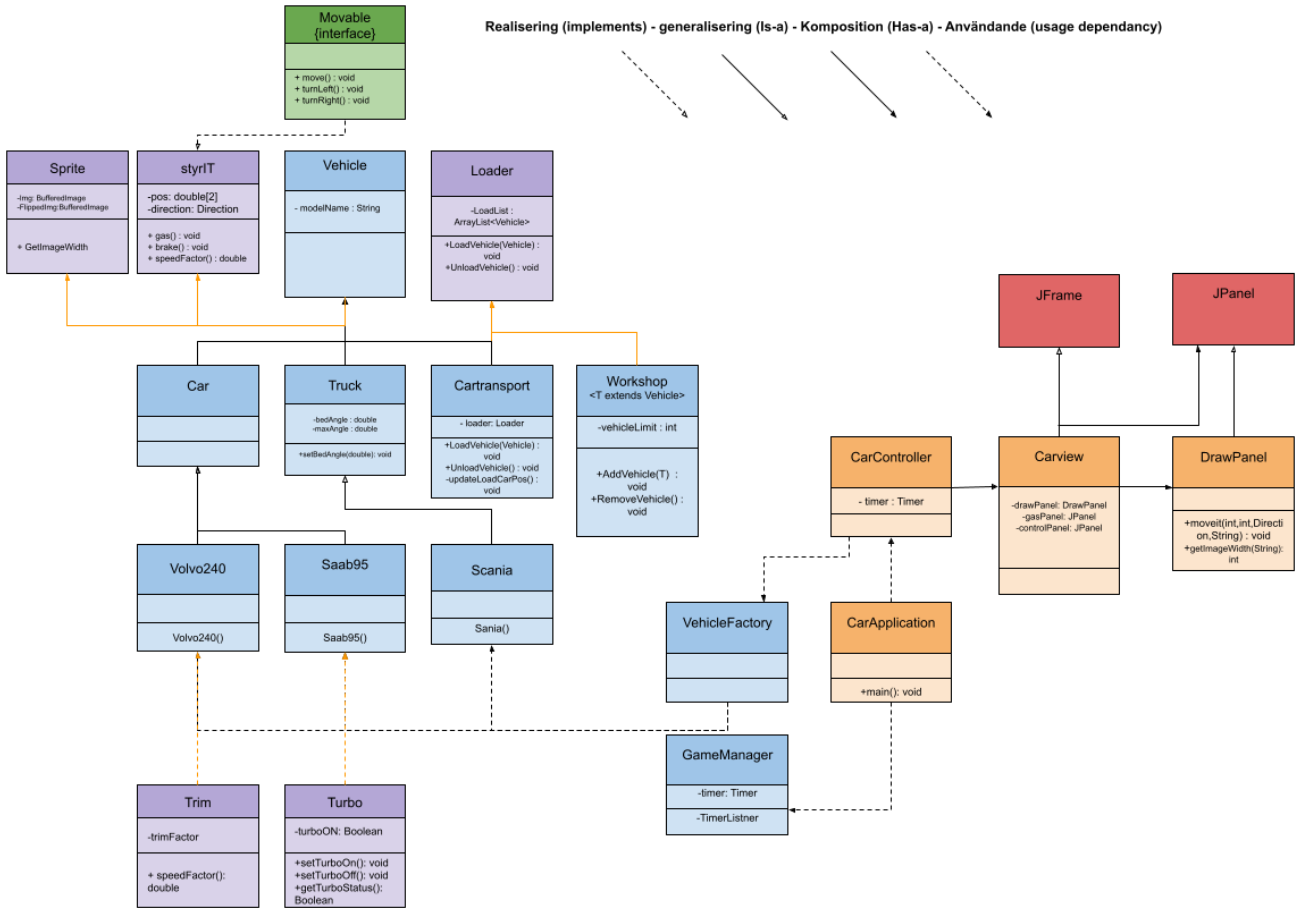
Observers finns, och dessa har vi använt oss av oavsiktligt då de följde med koden som vi fick från git, alltså inget som vi implementerade själva. Fler kanske finns, **återkommer till detta,** dock i så fall inte medvetet

Observer pattern gör att controller kan ha indirekt tillgång till .

Vi införde även ett factory pattern som gör att det finns en fabrik med unika metoder som var för sig kallar på de olika fordonens konstruktorer. Fördelen med detta är att vi ökar cohesion i modellen och minskar coupling mellan modellen och application.

Nytt UML-diagram

Byt håll på pilarna från vissa komponenter, om vi utgår från det som vi redan har!



Uppgift 5: Kan något designmönster vara relevant att använda för denna utökning?

Vi har använt Factory Pattern Method samt observers för att genomföra denna utveckling, däremot anser vi att fler designmönster skulle öka komplexiteten i programmet över vad kravspecifikation kräver. Det är möjligt att fler komponenter samt en något bättre struktur skulle kunna främja expanderingsmöjligheterna i programmet samt froda en bättre maintenance-miljö. Eftersom labben är så pass liten i scope och programmet kommer att arkiveras efter kursen finner vi att dessa förbättringar inte kommer nyttja någon, och därmed är överflödiga.