Lab 4

Uppgift 2:

- 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?

    View har panels/spinners vilket inte visar information från modellen och därför bör inte vara i view. View ska vara DUM. Flyttar till controller istället.

    Controller har en main i sig som använder set metoder. Den har också en listener (observer) i sig vilket betyder att den inte är THIN då den gör mer än att bara hantera “externa inputs” som till exempel att adda cars i lists. Detta löste vi genom att skapa en Application som hanterar vår main funktion och innehåller vår listener (observer) samt initierar vår MVC.

    Dock missade vi…

    CarController hade fortfarande för brett användningsområde vilket följde SRP dåligt. Den hade konkreta och hård kodade metoder som fick interagerade med vår modell. Detta löser vi med en CarModel klass som har alla metoder vi vill applicera på våra bilar. Vi förflyttar även vår ArrayList med vehicles till CarModel istället för CarController vilket leder till att CarController får ett tydligare ansvarsområde och blir mer tunn!

    Vi bytte även så att vår VehicleFactory kallades ifrån CarModel istället för Application då det följer SRP bättre.

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

    Visades på redovisning

Uppgift 3:

- 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 pattern, avsiktligt eller oavsiktligt? Vilka designproblem löste ni genom att använda det?

Observer pattern har vi inte

Factory method har vi implementerat avsiktligt i form av VehicleFactory. Detta gör att vi bättre följer DIP och Open/Closed principle genom att vi inte behöver skapa nya Vehicle objekt i vår CarController utan att det räcker med en VehicleFactory som vi kan använda för att skapa våra bilar. Detta ger oss också lower coupling då CarController inte behöver dependa på varenda subclass till vehicle som den vill använda sig av.

State pattern har vi via vår Enum Direction genom att vår move metod ökar/sänker bilens x/y koordinat beroende på vilket håll bilen kollar åt, dvs bilens “state”

Composite pattern har vi inte.

    - Finns det något ställe där 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?

Observer kan vi lägga till så att repaint händer så fort en bil position ändras. Detta tillåter också så att fler observers (view objekt) kan uppdateras vid olika händelser, t.ex om en bil nuddar väggen så sprängs saker.

Factory pattern har vi redan implementerat och kan ej hitta fler lämpliga appliceringsområden.

State pattern används redan men vi kan också användas för CarTransport genom att ha en PlatformState som antingen är up eller down vilket då låser move funktionen när Platform inte är uppe och

kanske t.om låser store funktionen när Platform inte är nere. Däremot känns detta inte nödvändigt.

Composite pattern tror jag inte hade förbättrat någonting konkret med den nuvarande designen vi har. Eftersom vi inte har så många “nested” objekt så finns det inget riktigt behov av det (Vi har ingen tydlig trädstruktur). Det enda jag kan tänka mig där det eventuellt skulle vara användbart är om man har en workshop med en CarTransport i sig som i sin tur har massor av bilar på sig, då kan ett Composite pattern vara användbart ifall man t.ex vill ta reda på hur många bildörrar som finns i workshoppen eller liknande.