**Uppgift 2**

*Vilka beroenden är nödvändiga?*

Nödvändiga beroenden: Scania -> AngularLift, Transporttruck -> OnOffLift, Carcontroller -> Positionable, Carcontroller -> Vehicle, Carcontroller -> HasTurbo, CarController -> Scania\*, CarController -> DrawPanel, CarController -> CarView.

*Vilka klasser är beroende av varandra som inte borde vara det?*

Både Volvo240 och Saab95 är subklasser till Car. I nuläget tillför Car ingen funktionalitet och skulle kunna ersättas med ett gränssnitt. Vi har dock valt att använda en klass eftersom det i framtiden kan finnas gemensamma egenskaper för alla bilar.

*Finns det starkare beroenden än nödvändigt?*

CarController har för många beroenden, som vi nämner nedan behövs den delas upp och “jämna ut” de beroenden den redan har.

*Kan ni identifiera några brott mot övriga designprinciper vi pratat om i kursen?*

Gällande sammanhållning (cohesion), finns det ett antal tillfällen då klasserna gör mer än vad de borde. Ett exempel på detta är Vehicle som hanterar allt som är genemsamt för alla fordon. Vi ser möjlighet att istället låta Vechicle vara beroende av mindre komponenter. På detta vis hade koden blivit mer överbliclig.

Ett annat exempel är att carController i nuläget hanterar körningen av application samt all logic bilarna använder och all visuell kommunikation mellan carView och Drawpanel. Vi kan dela upp dessa till seperata klasser för att följa SRP.

I dagsläget implementerar både Vehicle och Workshop gränssnittet Positionable. För Vehicle sker detta indirekt genom att först implementera Movable, som i sin tur implementerar Positionable. Eftersom implementationen av Positionable är indentisk mellan Vehicle och Workshop hade det varit bättre att omvandla Positionable till en klass. Detta hade tagit bort den upprepade koden.

**Uppgift 3**

*Vilka ansvarsområden har era klasser?*

**Workshop.java:** Ansvarsområde är att sammanställa de funktioner alla Workshops har, samt alla variabler de skall ha. Exempelvis en load funktion.

**Saab95.java:** Innehåller Saab95 specifika funktioner och egenskaper.

**Car.java:** En klass var syfte är att sammanställa alla “Cars” funktioner. I nuläget finns det inget gemensamt som bilarna delar men i framtiden så skulle den kunna vara nödvändig.

**OnOffLift.java:** Ansvarar för funktionaliteten av en OnOffLift. Transporter är beroende av denna klass.

**AngularLift.java:** Ansvarar för funktionaliteten av en AngularLift. Endas Scania är beroende av denna i nuläget.

**CarController.java:** Ansvarar för sammanfogningen av Carview knapparna och deras respektive funktionalitet från övriga klasser. Ansvarar också för “Tid” i programmet samt vilka objekt som skapas i programmet.

**DAFFXH.java:** Ansvarar för attributen för en DAFFXH lastbil. För tillfället har den ingen unik funktionalitet.

**CarView.java:** Ansvarar för de knappar och handlingar som kan genomföras när programmet körs.

**VolvoWorkshop.java:** Har endast ansvar för de värden som är satta via Workshop-klassen.

**Volvo240.java:** Har ansvar över Volvobilens unika funktionalitet, samt dess värden.

**Truck.java:** En samlingsclass för alla lastbilar. För tillfället har den endast ansvar att sätta värden på nya lastbilar som genereras likt Cars.

**Scania.java:** Har ansvar över Scaniabilen funktionalitet samt beroendet av AngularLift.java vilket Scania bilen använder sig av.

**DrawPanel.java:** Ansvarar för de visuella aspekterna av programmet, genererar bilder och färg.

**TransportTruck.java:** Ansvar över transportlastbilars funktionalitet. Beroende av OnOffLift.java

**Vehicle.java:** Ansvar över alla fordons funktionalitet i form av rörelse, position, motorfunktionalitet, hastighet, riktning, följesfuktionalitet, värden för bildetaljer.

*Vilka anledningar har de att förändras?*

*På vilka klasser skulle ni behöva tillämpa dekomposition för att bättre följa SoC och SRP?*

Framför allt ser vi att vi har stora möjligheter att förbättra CarController och Vehicle.

**Uppgift 4**

Rita ett UML-diagram över en ny design som åtgärdar de brister ni identifierat med avseende både på beroenden och ansvarsfördelning.

Motivera, i termer av de principer vi gått igenom, varför era förbättringar verkligen är förbättringar.

Skriv en refaktoriseringsplan. Planen bör bestå av en sekvens refaktoriseringssteg som tar er från det nuvarande programmet till ett som implementerar er nya design. Planen behöver inte vara enormt detaljerad. Se Övning 3: UML, static vs dynamic för ett exempel på en refaktoriseringsplan.

Finns det några delar av planen som går att utföra parallellt, av olika utvecklare som arbetar oberoende av varandra? Om inte, finns det något sätt att omformulera planen så att en sådan arbetsdelning är möjlig?

Refraktorsplan

Steg:

Vehicle:

1. Börja med att skapa en Position class.
2. Flytta getX och getY funktionerna till denna klass
3. Flytta X- och YPos variablerna till denna klass
4. Ersätt direction variabel med en direction enum
5. Uppdattera alla subclasser (Car, truck, transportTruck) med nya funktionaliteten

CarController:

1. Skapa ”Simulationcontroller” classen
2. Flytta all ”Fysik” logic till denna class
3. Skapa en ”viewcontroller” class
4. Flytta all användare logic från CarController till denna class. (användare handlingar som gas/brake)
5. Koppla dessa funktioner till funktionerna i ”Simulationcontroller” och till knapparna i carview

CarView:

1. Skapa SimulationListener interface. CarView ska implementer denna interface
2. Simulation Controller ska notifiera CarView via SimulationListner när något förändras

Workshop:

1. Skapa ”canLoad()” metoden i Workshop (Den ska kolla om bilen som vill laddas får laddas)
2. Flytta positionsvariabler och funktioner till Position klassen (vehicle steg 2 & 3).

TransportTruck:

1. I transporttruck, ändra funktioner för att anpassas till nya Position classen

Main:

1. Flytta ut main() funktionen i CarController till en dedikerad application class (Om denna var den enda funktionen i CarController så ändra endast class namnet till Application)
2. Definiera och initiera CarView, SimulationController och ViewController i denna class