Uppgift 2

CarView har strong composition med avsikt på:

en CarController

en DrawPanel

en WindowSettings

CarController har strong composition med avsikt på:

en CarView

en WindowSettings

en TimerListner

DrawPanel är dependent på CarController (onödigt; bara två getters)

Det är composition från CarController till CarView och vise versa (double dependency; bör ändras; CarController bör inte ha composition med avsikt på CarView, utan borde ha en annan typ av dependency)

Hög coupling mellan CarController och CarView; det här kan vara dåligt ifall man vill använda den ena men inte den andra (bryter mot high cohesion, low coupling)

CarController och CarView innehåller objekt av varandra

Man bör försöka få bort CarView-referenserna från CarController

I innit components och CarView generellt skulle man kunna skippa att spara CarController som en variable och bara passa som ett argument till innit components (Law of Demeter)

Vi gör Model, View och Controller bra (separation of concern)

Uppgift 3

Vilka ansvarsområden har klasserna:

CarView sköter allt som har med fönstret att göra och har även en listener som kollar om knapparna trycks på. Man hade kunnat göra två olika klasser (**dekomposition**); en som sköter knapptryckningar och som sköter själva fönstret. Skulle bli lättare att läsa och det skulle bli lättare att hitta saker i klasserna, samt blir det bättre med avsikt på SoC.

CarController sköter själva logiken som får bilarna att röra på sig och håller dem innan fönstret.

Cars representerar själva informationen om bilen själv, samt hur den rör sig. Från ett perspektiv av SoC är Cars i en lite sämre position än vad den annars skulle vilja vara. Vi hade t.ex kunnat fixa movable som sin egen klass för att se till att Cars enbart jobbar med statistiken av bilen och inte hur den rör sig, så den håller sig till funktioner unikt till bilar/fordon.

Saab95 representerar informationen om Saab95 modellen och är en subklass av Cars. Detta blir enligt SRP en mer effektiv lösning på hur programmet kan struktureras då vi egentligen bara har en anledning att modifiera Saab95 om det är någon nödvändig information som behövs ändras

Volvo240 har samma fördelar som Saab95 då de egentligen inte har mycket skillnad från varandra annat än att de inte delar funktioner mellan varandra.Det är en fin lösning på SoC då en klass inte borde behöva representera flera olika modeller samtidigt när de inte har något gemensamt med varandra bortsett från att de är en bil.

Scania och **TransportTruck** är subclasser av Cars, och innehåller information om Scania-, respektive TransportTruck-modellen. Har samma fördelar som de andra modellerna.

Workshop lagrar bilar. Inte så mycket mer denna klassen gör och har därför inte någon anledning att egentligen ändras annat än om vi skulle vilja ändra ansvarsområdet på hela klassen.

DrawPanel ritar rutan och allt där i, den ansvarar även för att flytta på bilarna (att flytta på bilarna borde ändras till att vara CarController’s ansvar)

WindowSettings gör bara en grej (innehåller information om fönstret), och följer därför automatiskt SoC och SRP.

Att göra:
carHeight och carWidth bör flyttas till WindowSettings. windowHeight och windowWidth bör omvandlas till static variabler, och deras konstruktor bör tas bort. Man bör ta bort CarController's och CarView's composition med avseende på WindowSettings, och istället skapa getters för att hämta dess variabler. Man bör även flytta distanceConstantY till windowSettings och skapa en getter för den.

Just nu har DrawPanel en metod för att flytta på bilarna, detta borde flyttas över till CarController alt. göras automatiskt utan att behöva kallas som en metod i en annan klass

Hela klassen Cars bör göras abstrakt. Cars bör bara ha ansvar för bilens data och inte för bilens movement. Så en movable-klass skapas så att move metoderna ligger helt och hållet utanför Cars så allting unikt för bilar hamnar i Cars metoden.

Dekomposition ska göras på CarView så att en klass sköter alla knapptryckningar och en annan som sköter själva fönstret.