Ansvarsområden för klasser:

- Car: en generell mall för en bil. Alla bilobjekt som skapas ska ha dessa metoder (e.g. gas, brake, move, incrementspeed & decrementspeed).
- Scania, Volvo240, Saab95, CarTransport: alla dessa är subklasser av Car (extends). Utöver de generella attributerna från Car har de även metoder specifikt för den biltypen (e.g. TurboOn och TurboOff i Saab95). Scania och CarTransport har även ett flak (PlatformTwoStates & PlatformWithAngle är egna klasser). CarTransport har även metoder för att lasta på och av bilar.
- CarTransport: har en Deque<Car> för att kunna lasta på bilar som ska "transporteras". Deque används då vi vill följa first in-last out.
- CarShop: använder en ArrayList för att lagra bilar i verkstaden. Har också metoder för att lägga in och ta ut bilar ur en CarShop.
- DrawPanel: ritar fönstret (grafik) där bilarna finns, samt håller koll på bilarnas position under exekvering. Används i CarController.
- CarView: har ansvar för panelen (JPanel) och knapparna (JButton) för panelen som man använder för att påverka bilarna under exekvering. Dessa används i CarController. CarView använder också en CarController (Has-A).
- CarController: använder en TimerListener för att flytta alla bilar som existerar i en ArrayList. CarController kör också programmet. Klassen har även metoder som behövs för att en bil ska köra.

Möjliga förändringar:

- CarController, CarView respektive DrawPanel har flera ansvar och är dessutom beroende av varandra (e.g. CarController och CarView är beroende av varandra (aggregation)). Därför hade det varit bättre att splittra dessa klasser och dela upp ansvaren i mindre klasser. På så sätt följer vår kod Separation of Concern-principen och Single Responsibility-principen. För att minska klassernas beroende av varandra (dependency inversion principle) kan man även skapa en ny klass som skapar objekt av CarController, CarView och DrawPanel.
- CarTransport har två olika ansvar (är en Car och kan lasta bilar) vilket strider mot SRP. För att fixa detta skulle man kunna skapa en ny klass Loading som innehåller metoder för att lasta och avlasta bilar.

Refaktoriseringsplan:

1. Loading: Ny klass Loading skapas för att ta hand om lastning av bilar i en CarTransport (SRP).
2. FactoryGraphics (fabriksklass) skapas för att minska beroenden mellan ovanstående klasser. I denna ska det finnas metoder för att skapa objekt av CarController, CarView och DrawPanel. Klasser som behöver något av dessa objekt är då beroende av CreateGraphics.
3. FactoryCar (fabriksklass) skapas för att kunna skapa bilar åt CarController (minskar beroende mellan Car och CarController)
4. FactoryCarShop: skapas för att kunna skapa en CarShop åt DrawPanel.
5. DrawPanel: En ny klass DrawImage skapas med metoder från DrawPanel (behåller dock PaintComponent för att uppdatera panelen under exekvering). Denna klass skapar alla bilder och lägger till bilar.
6. CarController: Splittras för att skapa Main, resten är kvar i CarController. I Main finns nu main som kör programmet samt en ArrayList (+ kopia) för bilarna som den får från ett CarController-objekt den skapar. CarController sköter nu knapparnas funktion samt gör så att

det läggs till bilar i listan.

7. CarView: Allt som använder ActionListener tas ut och sätts in i en ny klass ActionListener. Denna har en CarController. CarView har ansvar för fönstret som skapas vid exekvering, detta behöver ett DrawPanel-objekt.