2.

Analysera de beroenden som finns med avseende på cohesion och coupling, och Dependency Inversion Principle.

- Vilka beroenden är nödvändiga?
    - Superklassen Car är grundläggande till alla subklasser som ärver dess framework och har grundläggande funktion.
    - Truck är en subklass av car men den ytterligare funktionaliteten som släp, vilket inte alla typer av Car har.
    - Vidare är loadable, cartransporter och workshop beroende av Car eftresom att alla dessa använder Car i sina funktioner. Dels för lastning och avlastning.
    - CarController är beroende av Car

- Vilka klasser är beroende av varandra som inte borde vara det?
    - CarController är beroende av CarView, vilket de inte ska göra anser vi. CarView och CarGame bör istället vara beroende av CarGame.
    - CarController ska friläggas från skapandet av nya bilobjekt, detta ska skötas i en ny klass, exempelvis en CarFactory, det vill säga CarController ska inte ha med typer av Car att göra.

- Finns det starkare beroenden än nödvändigt?
    - CarController och CarView är för starkt kopplade och bryter därför mot HCLC principer

- Kan ni identifiera några brott mot övriga designprinciper vi pratat om i kursen?
    - SRP - Single responsibility principle:
        - CarController instansierar bilar och sköter dess rörelse i spelet. Vilket bör ske separat. Vi skapar därmed en CarFactory som skapar bilar och låter därför Controller enbart sköta rörelsen av instanserna
    - Open/Close - principen
        - Genom att implementera Workshop och Car i DrawPanel undviker vi hårdkodning där - däremot hade vi behövt hårdkoda Workshop i DrawPanel om vi inte hade namngivit den.

3.

Analysera era klasser med avseende på Separation of Concern (SoC) och Single Responsibility Principle (SRP).

- Vilka ansvarsområden har era klasser?
    - Car - Hur huvudfunktionaliteten för alla fordon, exempelvis turn, increment- och decrementspeed, Gas och Brake.
    - Loadable - definierar funktionalitet som är nödvändig för fordon som lastar.
    - Plattform - Skapar en påbyggnad till Truck som är höj och sänkningsbar.
    - Truckplattform - implementerar höj och sänkning som är specifik för Scania då de höjs med antal grader och kan ha olika lägen.
    - CarPlattform - Är antingen lastbar eller inte samt enbart kan vara uppe eller nere.
    - DrawPanel - speldesignen, färgläggning och förflyttning av bilderna
    - CarController - Låg cohesion då den kopplar både till volvo, saab, scania och workshop samt carview.
    -
    - Volvo240 och Saab95 innehåller information som är unika för de två modellerna såsom TrimFactor respektive Turbo
    - Truck uttrycker en specifik funktionalitet för en större bil (Truck)
    - Moveable definierar funktionalitet som avser att röra sig
    - CarTransport definierar ett större fordon (Truck) som ska kunna lasta och transportera bilar. Vilket utgör den större skillnaden gentemot Truck.
    - WorkShop implementerar en ny klass som kan lasta objekt men särskiljer sig med fast position och som är beroende av Car, kan defineras så att den enbart tar emot vissa typer av bilinstanser såsom Volvo
    - CarView - skapar alla knappar som används i programmet

- Vilka anledningar har de att förändras?
    - CarController och CarView har för många ansvarsområden och bör därför delas upp
- På vilka klasser skulle ni behöva tillämpa dekomposition för att bättre följa SoC och SRP?
    - De tre klasserna som tillkom i denna lab bör dekomposeras för att bryta ut funktionalitet på ett mer modulärt sätt

4.

Skriv en refaktoriseringsplan. Planen bör bestå av en sekvens refaktoriseringssteg som tar er från det nuvarande programmet till ett som implementerar er nya design.

- **Motivera i termer av de principer vi gått igenom varför era förbättringar verkligen är förbättringar**
    - Genom att dekomosera CarView och CarControllerer kan vi minska beroendet däremellan och uppnå HCLC principer
    - Det skulle ge en bättre struktur och en mer lättläst kod, dvs den är mer modulär
    - Genom att lägga till ett CarGame får vi bort beroendet mellan CarView och CarController, dvs bättre enligt HCLC
    - CarFactory tillåter oss att instansiera bilar och slippa göra detta i CarControllerar vilket innebär Seperation of Concern blir bättre

- **Vår refaktoriseringsplan**
    - Skapa en ny klass CarFactory
    - Skapa en ny klass CarGame
    - Bryt ut EventHandler från CarView - dvs avlyssning ska ske på annat ställe
    - Bryt ut TimeListener från CarController

- **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 plan en så att en sådan arbetsdelning är möjlig**
    - CarFactory behövs göras först efftersom att CarGame nyttjar de instansierade bilarna, således kan dessa inte implementeras parallellt
    - TIllägget CarGame och CarFactory kan göras parallellt med dekompositionen av CarView och CarController