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

a. Vilka beroenden är nödvändiga?
Truck/CarTransport till dess Cargos
(CarTransport - - - > RampCargo
Truck —> Cargo
Scania - - - > TipCargo

CarWorkshop och CarTransport beroende till CarStorage

Beroende-kedjorna från (Truck och) Car till MovableObject
(Truck) -> Car -> Vehicle -> MovableObject

CarController är beroende på våra "Bilmodellerna" (Volvo, CarWorkshop<Volvo>, Saab95 osv)

b/c. Vilka klasser är beroende av varandra som inte borde vara det? Eller är starkare än nödvändigt?
Truck —> Car
Detta beroende vill vi undvika - En Truck är inte "en typ av bil" - deras gemensamma beteende är mer eller mindre att de båda har motorer

DrawPanel ← → DrawableObject.object
I vår nuvarande design har vi brutit mot MVC-patternet och vi har lagt in logiken för att hantera kollision med vägg/verkstad - detta skall naturligtvis skötas genom CarController.
DrawableObject skall egentligen inte behöva en referens till objektet den representerar, då det är "Controllerns" ansvarsuppgift.

d. Kan ni identifiera några brott mot övriga designprinciper vi pratat om i kursen?
Vi har under projektet gått tillbaka och ändrat i klasser och därmed brutit mot open/close-principen.
Exempel: För att hålla koll på om ett fordon är transportabelt krävdes ändringar i klassen Car.

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

a. **Vilka ansvarsområden har era klasser?**

- CarController hanterar logik kring att skapa bilar/workshops och hanterar logik/funktionalitet vid knapptryck.
- CarView Skapar knappar, visar knappar och skickar knapptryck till CarController.
- DrawPanel ritar upp allt, men hanterar också kollision med väggar och Workshop-logik.

b. **Vilka anledningar har de att förändras?**

DrawPanel bryter mot SRP eftersom den har tre ansvarsområden. Klassens bör endast rita, och inte hantera någon form av logik om kollisioner eller workshop → måste flyttas till CarController.

VarView bryter SRP och ansvarar både för att skapa samtliga knappar, samt att rita ut samtliga knappar och skicka vidare dess event. Denna logik kan brytas upp i mindre klasser (en klass för en knapp)

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

DrawPanel behöver "dekomposteras" och flytta sin logik till CarController så att DrawPanel endast ansvarar för att rita. P.s.s. behöver DrawPanel enbart ansvara för att rita ut en Image, och klassen DrawableObject kommer enbart att behöva en Bild-instans och en Point instans som representerar varje ritbart objekt (Bil/Workshop)

Logiken i CarView kan brytas upp (dekomposition) i subklasser som ansvarar för att skapa en knapp i taget, koden blir mer lättöverskådlig och lättare att följa koden. Nu är det väldigt svårt att följa och förstå koden.

Genom att bryta ut motor-logik ur Car klassen med dekomposition, till en ny klass Engine kan vi ex. använda denna klass om vi vill skapa nya motordrivna fordon, som inte nödvändigtvis är en bil. Till exempel Truck (som just nu ärver från Car), som har en motor men inte "är" en bil.

Vi implementerar Engine klasserna i både Car och Truck klasserna via Komposition har vi fördelat ansvaret enligt SRP (Engine får specifikt motorlogik (speed, gasa, bromsa), och därmed även Car en mer specifik uppgift) + SoC (En motor finns i en bil, bilen har ingenting med motorimplementationen att göra).

Le plan

1. Göra en ny klass Engine.
    a. public class Car extends MovableObject {Engine engine = new Engine;}
    b. Följaktligen → Truck extends MovableObject { … }
2. Flyttar kollisions- och Workshop-logiken från DrawPanel till CarController (ta bort object fält ur DrawableObject → Rename till Drawable?)
3. Bryter upp knappar i CarView klassen till mindre klasser