Uppgift 2: Beroenden

1. Movable-interface och Vehicle har hög cohesion, för att Movable metoder (move, turnLeft, turnRight) är ansvar om rörelse, medan Vehicle metoder är är ansvar om beteende och egenskaper. Dessutom har Turbo och Trim också hög cohesion, eftersom Turbo är ansvar om turbo-relaterade funktioner, och Trim är också ansvar om trim-relaterade funktione.

2. CarController och DrawPanel har hög coupling med Vehicle, eftersom CarController är coupling med Vehicle beteende, och är beroende av dem, medan DrawPanel använder Vehicle pictures för att rita bilarna. I andra hand CarView har hög coupling med CarController, för att den visar information från CarView och skickar till CarController.

3. Dependency Inversion Principle (DIP): Saab95 och Volvo240 följer den för att de bero på abstraction klass (Vehicle). Scania och transport är också beror på abstraction klass (Truck).Beroenden mellan Vehicle, Movable och Direction är nödvändig för att de är ansvar om rörelse för bilen. Beroenden mellan CarController och Vehicle är nödvändig för att styra beteende för bilar. CarController och CarView har hög koppling och borde bero på varandra mindre.

4. Dependency Inversion Principle (DIP) brott för att det finns en viss koppling mellan CarController och CarView, så vi behöver att minska Dependency mellan dem.

OCP: Klassarna är inte helt öppen för **extension** och stängd för modification. Tillexempel är Scania som har brott OCP, och vi kan flytta platfom into its own class.

Uppgift 3: Ansvarsområden

Vehicle Class:

- **Ansvarsomraden:**
    - Representera vanliga egenskaper och beteenden hos alla Vehicle.
    - Hantering av rörelse, att svänga till höger och vänster, hastighet.
- **Anledningar att ändra:**
    - Tydlig åtskillnad mellan lastbilar och vanliga bilar som ärver Vehicle, genom skapa ett klass som heter Car som ha alla vanliga bilar.
    - Utökning av fordon i framtiden

· **Volvo240 and Saab95 Classes:**

- **Ansvarsomraden:**
    - Representerar specifika egenskaper och beteenden hos Volvo- och Saabbilar.
    - Hantera trimfaktorer och turbo, som är beroende av klasserna trim och turbo.
- **Anledningar att ändra:**
    - Ingen anledning för att Jag har flyttat turbo och trim till ny klass, för om vi kunde skapa en ny bil med trim eller turbo utan att ändra eller duplicera kod)

· **Scania Classes:**

- **Ansvarsomraden:**
    - Representerar specifika egenskaper och beteenden hos Scania.
    - Inkludera funktionalitet för att höja och sänka plattformen.
- **Anledningar att ändra:**
    - Det kan vara bättre att lägga in koden för plattform i en egen klass som Scania ärver från eller delegerar till. Då kunde vi skapa en ny lastbil med plattform utan att ändra eller duplicera kod.

Transport and CarRepairShop Class:

- **Ansvarsomraden:**
    - Transport presentera en transportbil med förmåga att transportera andra fordon.
    - CarRepairShop presentera en verkstad för fordon.

    - Både hanterar load och unload av bilar..
- **Anledningar att ändra:**
    - För att undvika kodduplicering och främja, kan skapa ett gemensamt gränssnitt. (Innehåller load and unload)

CarController Class:

- **Ansvarsomraden:**
    - Represents the controller in the MVC pattern.
    - Innehålla metoderna som körs.
- **Anledningar att ändra:**
    - Klassen har många responsibility.(SRP att vi kan delar klass till två klass, och varje klass har nu ett mer fokuserat ansvar)
    - kod i klassen är svart att underhåll, så för att följa MVC dela upp ansvaret till två klassarna, och det är bra för kodunderhåll och skalbarhet.

Draw Panel Class:

- **Ansvarsomraden:**
    - Ansvar om grafisk
- **Anledningar att ändra:**
    - Ingen anledningar att ändra.

Car View Class:

- **Ansvarsomraden:**
    - Ansvar om knapparna och ansvar för knapparna.
- **Anledningar att ändra:**
    - Många responsibil som har klass, och det bättre att flytta knappr till ny klass.
    -

Uppgift 4: Ny design

Vi vill organisera vår kod på ett mer modulärt sätt för att följa principen om open/close princip (OCP) och för att förbättra strukturen och underhållbarheten.

Vi börjar med klassen Vehicle genom att skapa en klass som heter Car och volvo och scania arver av Car, båda som är subklasser till en ny abstrakt klass som vi kallar Car, och Car är abstrakt klass till Vehicle. Detta möjliggör enklare expansion med fler fordon i framtiden och ger oss möjlighet att tydligt särskilja lastbilar från vanliga bilar.

Det kan vara bättre att lägga klass som involverar Plattform i en egen klass som Scania ärver från eller delegerar till. Så kunde vi skapa en ny bil med Plattform utan att modifiera eller duplicera koden. (OCP)

För att undvika kodduplicering och främja, kan skapa ett gemensamt gränssnitt för Transport and CarRepairShop. **(ISP)**

För att minska antalet direktberoenden till bilklasserna och göra koden mindre modifierbar, inför vi en ny klass kallad `CarAssembler`. Denna klass kommer fungera som en konstruktor för våra fordon och kommer att hantera skapandet av nya instanser av fordon. Detta hjälper till att upprätthålla öppen/sluten princip och gör att vi kan lägga till nya fordonstyper utan att behöva ändra befintlig kod.

För att förbättra modulariteten och göra koden mer strukturerad, delar vi upp klassen CarController i två klasser: App och CarController. App kommer att hålla huvudmetoden, hantera TimerListener, samt refera till CarController. Medan CarController kommer bara att innehålla metoderna som körs. Detta följer separationen av ansvar (SoC) och bidrar till en mer modulär struktur, som kan relateras till modell-visnings-kontrollern (MVC)-mönstret.

Att skapa för varje knappar en egen klass (`// stopButton startButton lowerBedButton liftBedButton turboOffButton turboOnButton brakeButton gasButton`)