Uppgift 2: Beroende

1. **Movable-interface och Vehicle:**
    - **Cohesion och DIP:** Hög cohesion eftersom Movable-interface och Vehicle samarbetar för rörelse och båda är beroende av varandra. DIP upprätthålls eftersom båda är beroende av abstraktion (Movable).
    - **CarView och CarController:** Hög coupling och inte helt enligt DIP. CarView är starkt kopplad till CarController och vice versa. Minskning av beroendet mellan dem kan förbättra DIP.
    - **Fordonstyper och Vehicle:** Fordonstyperna (Volvo240, Saab95, Scania) är beroende av Vehicle, vilket är rimligt eftersom de delar gemensamma egenskaper.

2. **CarController och DrawPanel:**
    - **Coupling med Vehicle:** CarController har hög koppling till konkreta fordonstyper. Detta kan bryta mot DIP. Minska kopplingen genom att använda abstraktioner kan förbättra designen.
    - **Coupling mellan CarController och DrawPanel:** CarController har hög koppling med DrawPanel, och detta kan ses som en möjlig förbättring.

3. **Dependency Inversion Principle (DIP):**
    - **Följs:** Saab95 och Volvo240 följer DIP genom att vara beroende av abstraktionen (Vehicle). Scania och Transport följer också DIP genom att vara beroende av abstraktionen (Truck).
    - **Brott mot DIP:** Beroendet mellan CarController och konkreta fordonstyper kan betraktas som ett brott mot DIP. Minskning av kopplingen här kan förbättra DIP.

4. **Dependency Inversion Principle (DIP) brott:**
    - Beroendet mellan CarController och konkreta fordonstyper (Volvo240, Saab95, Scania) kan ses som ett brott mot "Dependency Inversion Principle". Minskning av kopplingen här genom användning av abstraktioner kan förbättra DIP.

Uppgift 3: Ansvarsområden

1. **Vehicle Class:**
    - **Ansvarsområden:** Allmänna egenskaper och beteenden för alla fordon.
    - **Orsaker att ändras:** Förändringar i gemensamma attribut eller metoder för alla fordon.
    - **Dekomposition för SoC och SRP:** Överväga att skapa ytterligare abstraktioner om klassen blir för stor.
2. **Saab95 och Volvo240 Classes, Scania Class:**
    - **Ansvarsområden:** Specifika detaljer för varje fordonstyp.
    - **Orsaker att ändras:** Förändringar i specifika detaljer för varje fordonstyp.
    - **Dekomposition för SoC och SRP:** Överväga att dela upp i mindre specialiserade klasser om ansvarsområdena växer.
3. **Transport and CarRepairShop Class:**
    - **Ansvarsområden:** Gemensamma egenskaper för transportfordon och CarRepairShop.
    - **Orsaker att ändras:** Förändringar i gemensamma attribut eller metoder för lastbilar.
    - **Dekomposition för SoC och SRP:** Dela upp i mindre ansvariga klasser om ansvarsområdena växer.
4. **CarController Class:**
    - **Ansvarsområden:** Hantering av kommunikationen mellan model och view, styr logik för fordonen.
    - **Orsaker att ändras:** Ändringar i fordonens beteende eller interaktion med användargränssnittet.
    - **Dekomposition för SoC och SRP:** Dela upp i mindre ansvariga klasser om ansvarsområdena växer.
5. **DrawPanel Class:**
    - **Ansvarsområden:** Hantering av visningen av fordon på en panel.
    - **Orsaker att ändras:** Förändringar i hur fordonen visas eller ritas.
    - **Dekomposition för SoC och SRP:** Dela upp i mindre ansvariga klasser om ansvarsområdena växer.
6. **Turbo och Trim Classes:**
    - **Ansvarsområden:** Tillhandahålla funktionalitet för trimning och turboladdning.
    - **Orsaker att ändras**: Förändringar i turbofunktionaliteten och förändringar i trimfaktorn.
    - **Dekomposition för SoC och SRP:** Dela upp i mindre ansvariga klasser om ansvarsområdena växer.
7. **Car och Truck Classes:**

    - **Ansvarsområden:** Gemensamma egenskaper för personbilar och lastbilar.
    - **Dekomposition för SoC och SRP:** Dela upp i mindre ansvariga klasser om ansvarsområdena växer.
8. **CarView Class:**
    - **Ansvarsområden:** Hantering av användargränssnittet och kommunikationen med kontrollern
    - **Orsaker att ändras:** Förändringar i användargränssnittet eller visuella aspekter.
    - **Dekomposition för SoC och SRP:** Dela upp i mindre ansvariga klasser om ansvarsområdena växer.

Uppgift 4: Ny design

Refaktorisering Plan:

Refaktoriseringsplan:

1. **Skapa egna klasser för varje knapp:**
    - **Åtgärd:** Skapa nya klasser för varje knapp, t.ex., StopButton, StartButton, LowerBedButton, LiftBedButton, TurboOffButton, TurboOnButton, BrakeButton, GasButton.
    - **Motivering:** Detta steg hjälper till att implementera separation of concern (SoC) och single responsibility principle (SRP) genom att delegera specifika knappbeteenden till separata klasser.
2. **Ny design för Scania och Plattform:**
    - **Åtgärd:** Införa en ny klass för plattform och låt Scania ärva eller delegera till denna klass.
    - **Motivering:** Överensstämmer med open/closed principle (OCP) genom att möjliggöra skapandet av nya biltyper utan att ändra befintlig kod.
3. **Skapa gemensamt gränssnitt för Transport och CarRepairShop:**
    - **Åtgärd:** Skapa ett gemensamt gränssnitt för Transport och CarRepairShop och låt båda implementera det.
    - **Motivering:** Främjar interface segregation principle (ISP) och gör att vi kan hantera både transportfordon och verkstad med gemensamma egenskaper.
4. **Införa CarAssembler-klassen:**
    - **Åtgärd:** Skapa CarAssemblerklassen för att skapa instanser av fordon och hantera deras konstruktion.
    - **Motivering:** Minskar kopplingen och främjar OCP genom att möjliggöra läggande till nya fordonstyper utan att ändra befintlig kod.
5. **Separera CarController och App:**
    - **Åtgärd:** Dela upp CarController i två klasser: App och CarController.
    - **Motivering:** Följer separationen av ansvar (SoC) genom att separera huvudmetoden och TimerListener från fordonens logik.

6. **Användning av abstraktioner:**
    - **Åtgärd:** Implementera abstraktioner som interfaces (Movable, Drawable) för att minska kopplingen mellan olika komponenter och göra koden mer flexibel.
    - **Motivering:** Följer Dependency Inversion Principle (DIP) genom att använda abstraktioner för att minska beroenden.

Vilka steg eller delar av laben kan man jobba med Parallell?

- Steg 1 och 2 kan utföras parallellt av olika utvecklare eftersom de berör olika delar av koden (knappklasser och Scania-design).
- Steg 3 och 4 kan utföras parallellt eftersom de handlar om att skapa nya gränssnitt och klasser för konstruktion av fordon.
- Steg 5 och 6 kan utföras parallellt eftersom de berör olika aspekter av koden (uppdelning av CarController och användning av abstraktioner).

Drawpanel måste inte ha så mycket (.....), och carController inte