Uppgift 2: Beroenden

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

Typer av nödvändiga beroenden

- Cohesion
    - CarView, CarController och DrawPanel tillsammans skapar ett visuellt UI och tillåter användaren att styra "spelet"
- Coupling
    - Platform och truck - tex truck kan bara köra och loada i ett visst läge av platform
    - CarController kan styra Vehicles
    - CarView & CarController använder varandra
    - VehicleWorkshop tar in olika typer av Vehicles
- Dependency Inversion Principle
    - Vehicle är en superklass som de övriga fordonen implementerar
        - Truck är en superklass som Scania och VehicleTransporter ärver från
    - De olika flaktyperna och Workshop implementerar loadable
    - Vehicle implementar drivable

Vilka klasser är beroende av varandra som inte borde vara det?

- CarController och CarView använder båda varandra, vilket skapar en Two-Way dependency. Vi vill bara ha dessa One-Way
- CarControllers uppgift borde vara att styra bilarna, men som koden är nu är det även den som skapar bilarna. Det borde finnas en separat klass som bygger själva världen som spelet existerar i dvs bilarna och Workshop etc

Finns det starkare beroenden än nödvändigt?

- CarController och CarView som förklarat ovan

Kan ni identifiera några brott mot övriga designprinciper vi pratat om i kursen?

- Single responsibility principle
    - CarController tillverkar nu bilar och sedan styr dem, vilket vi anser kan betraktas som två separata funktioner
    - CarController sköter även inlastningen och vändningen när man krockar med väggar. Kanske bör finnas en separat spel-modul som sköter detta, och så kan CC styra gas, brake, turn osv
    - CarView både flyttar bilarna och målar upp dem genom DrawPanel

- Law-of-demeter
    - Egentligen vårt eget fel, men vi bör fixa så att fler attribut inte är publika, och istället ha getters för de attribut andra klasser behöver kunna komma åt

Uppgift 3: Ansvarsområden

Vilka ansvarsområden har era klasser?

- **Vehicle** - Är grunden för alla fordon i programmet och ansvarar för att alla fordon kan röra sig och har grundattribut som till exempel färg, storlek, position och riktning.
- **Truck** - Ansvarar för att lägga till attribut som endast lastbilsliknande fordon har.
- **VehicleTransport** - Ansvarar för att definera fordonstransporter som kan lasta på och av bilar i rätt storlek.
- **VehicleTransportPlatform** - Definerar flaket till fordonstransporten som har två lägen, uppe och nere, där flaket endast kan lastas i ett läge.
- **VehicleWorkshop** - Ansvarar för att definera Workshop som kan ta in bilar av rätt modell.
- **Platform** - Ansvarar för att definera flak med en storlek som kan höjas och sänkas.
- **Saab95** och **Volvo240** - Definerar bilar av specifika modeller med speciella attribut som Turbo och SpeedFactor
- **Scania** - Definerar lastbilar av sorten Scania vilket har ett flak som kan höjas gradvis. Lastbilen kan endast köras när flaket är helt nere.
- **ScaniaPlatform** - Definerar flaket till Scania som kan höjas gradvis.
- **Loadable -** Definerar funktionalitet som används när saker ska kunna lasta objekt.
- **Moveable** - Definerar funktionalitet som används när objekt ska kunna röra på sig.
- **DrawPanel** - Ansvarar för att rita upp programmet, dess bilder och att dess bilder flyttas.
- **CarView** - Skapar programmets knappar och bakgrund
- **CarController** - Skapar instanser av fordon och gör så de kan styras och röra på sig.

Vilka anledningar har de att förändras?

- CarContoller har för många ansvarsområden och DrawPanel och CarView ansvarar för för liknande områden. Detta skulle kunna brytas ner och delas upp på ett tydligare vis.

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

- CarController skulle behöva brytas ned för att följa SRP då de ansvarar för flera olika uppgifter just nu.
- SoC säger att en klass ska göra en sak och göra det väl. Detta är anledning till att CarView behöver brytas ned för att ha ansvar för en enskild uppgift som den gör så effektivt som möjligt.

Uppgift 4: Ny design

1. CarView och CarController bör göras om. CarView ska behålla funktionen som skapar det visuella interfacet, men andra funktioner ska flyttas ut därifrån. En annan klass bör hantera världens och bilarnas inverkan på varandra, tex krockar med väggar eller när en bil kör in i en Workshop och blir loaded. En tredje bör hantera användar-input som gas och brake. Denna refaktorisering leder alltså till tre klasser, istället för två så som det ser ut för tillfället.
    a. WorldView - hanterar det visuella interfacet
    b. ButtonController - hanterar gas, brake, raisePlatform etcetera
    c. EventHandler - hanterar krockar och laddningar i Workshop

2. En idé är att skapa en ny klass VehicleFactory i syfte att abstrahera beteendet hos alla konstruktorer som skapar vehicles. Vi skulle använda denna klass för att skapa nya instanser av Volvo, Scania och Saab i CarController klassen. Vi anser att denna förbättring ligger i linje med SOLID-principen *Single Responsibility Principle* som innebär att en klass endast ska ha ett ansvar.
    a. Det skulle även vara möjligt att ha en Factory för Workshops, alternativt döpa om ovan modul och implementera denna här

   Detta leder i sin tur att vi behöver en Game class som väver tillsammans dessa tre. Denna implementerar dessa och blir den nya klassen som ska köras när spelet spelas. Den kommer också använda VehicleFactory för att instansiera hela världen.

3. En annan sak vi kan göra är att skapa en ny abstrakt klass *Car* som extendar Vehicle - Saab och Scania ärver sedan från denna klass. Detta blir mer likt hur vi använder Truck idag, att Truck är en abstrakt klass som ärver från Vehicle, som sedan Scania ärver från. Genom detta minskar vi beroendet från Vehicle i linje med *Dependency Inversion Principle.*

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 planen så att en sådan arbetsdelning är möjlig?

Som vi beskriver det ovan ser vi ingen anledning till att refaktoriseringarna inte skulle gå att implementera parallellt med varandra.