Uppgift 2:
Analysera de beroenden som finns med avseende på cohesion och coupling, och Dependency Inversion Principle.
Vilka beroenden är nödvändiga?
Saab95, Volvo240, Truck → Vehicle
CarTransport, Scania → Truck
Vehicle → Movable, Positionable
CarWorkshop → Positionable
LifoStorage, FifoStorage → Storage
RampWithAngle, RampWithStates → TruckPlatform
Storage → Loadable

CarController → Vehicle, Saab95, Scania
TimeListener → CarController
// Drawpanel → vehicleImage, vehiclePoint → Cars

Vehicle, CarTransport, Saab95, Scania, Volvo240, Truck:

- Dessa klasser verkar relaterade till fordonsfunktioner och egenskaper. Det finns en hög sammanhållning inom denna grupp av klasser eftersom de alla är relaterade till fordon. Detta indikerar hög inre sammanhållning.
- Beroendet mellan dessa klasser verkar vara nödvändigt och rimligt, eftersom de alla representerar olika typer av fordon och dess egenskaper.
- Saab och Volvo är fördelaktiga enligt SRP då vi bara har anledning att ändra i dem om någon viktig info ska ändras, annars behöver vi inte röra dem. Även Scania och CarTransport har dessa fördelar.

CarWorkshop, CarTransport, FifoStorage, LifoStorage, Storage:

- Dessa klasser verkar relaterade till lagring och hantering av fordon. Deras sammanhållning bör vara hög då de alla handlar om att lagra fordon. Beroendet mellan dessa klasser och Loadable är nödvändigt för att implementera lagringsfunktioner.

Två exempel på Dependency Inversion Principle:
CarWorkshop, FifoStorage, LifoStorage, Storage:

- Dessa klasser implementerar Loadable-gränssnittet, vilket är ett exempel på Dependency Inversion Principle, där högre nivåer i systemet är beroende av abstraktioner istället för detaljer.

CarTransport, CarWorkshop:

- CarTransport utökar Truck och implementerar Loadable-gränssnittet. CarWorkshop utökar CarWorkshop och implementerar Loadable-gränssnittet. Dessa relationer tyder på att de har en liknande funktion inom systemet, men de har olika specialiseringar eller ansvarsområden.

Vilka klasser är beroende av varandra som inte borde vara det?
CarView → CarController bör inte vara beroende enligt MVC.
carController och carView innehåller objekt av varandra.

carView bör enligt MVC-pattern interagera bättre med modellen i något avseende (typ genom vehicle?) interagerar just nu med modellen via CarController

CarWorkshop, CarTransport → Loadable
- Får dem via komposition från fifostorage och lifostorage som får från storage.
- DIP, dependancy inversion prinsiple

Finns det starkare beroenden än nödvändigt?
CarView → CarController?

Kan ni identifiera några brott mot övriga designprinciper vi pratat om i kursen?
hög coupling mellan carController och carView – kan vara dåligt ifall vi vill använda den ena men inte den andra. bryter även mot HCLC.

– anteckningar –
timeListener behöver gå via carView för att använda drawPanel, känns dåligt spontant

konstigt med ramptyper , truck och scania. Scania har nu två ramper, använder endast 1. hur lösa?
- ta bort truck? Dock smidigt med instance of Truck när man vill gruppera Scania och CarTransport

Uppgift 3:

Ansvarsområden för våra klasser:
CarController
- sköter logistiken, dvs flytta bilar genom att lyssna på carView och därmed uppdaterar

CarTransport
- *Man kan lasta på vehicles men inte instanser av Truck samt kan sätta en ramp i olika lägen (på/av) som Truck kan.*
- ÄR en Truck
- Implementerar Loadable
- HAR LifoStorage<Vehicle>

CarView
- *Klass representerar allt som har med fönstret att göra där den har en actionlistener som kollar om knapparna trycks på.*

CarWorkshop
- *Är en klass som har ett fifostorage som tar in Veichles.*
- Implementerar Loadable(T)
- HAR FifoStorage<Veichle>

DrawPanel
- *Denna klassen ansvarar för att rita upp objekt och bild.*
- *Raderar carbilder och position i bildlistan när en vehicle raderas i CarController*
- ÄR JPanel

FifoStorage
- *Är en klass first in first out storage*

- ÄR Storage

LifoStorage
- *är en klass Last in first out storage*
- ÄR Storage

Loadable
- *interface för load(objekt) och unload: objekt*

Movable
- *interface för definition av move, turnleft, turnright.*

Positionable
- *interface för getters av xpos och ypos*

RampWithAngle
- *en klass som har en ramp vars uppgift är att ändra status på rampen med en vinkel eller läge.*
- implementerar TruckPlattform

RampWithStates
- *en klass som har en ramp vars uppgift är att ändra status på en ramps läge (på/av)*
- implementerar TruckPlattform

Saab95
- *skapar objekt av typ saab95*
- ÄR en vehicle

Scania
- *skapar objekt av typ scania har ett flak med vinkel overridar det som Truck har då den har RampWithAngle istället (men har också RampWithStates pga är en Truck)*
- HAR RampWithAngle
- ÄR Truck

Storage
- *Abstrakt klass som definierar ett lager för objekt har maxcapacity*
- Tar in objekt av typen T
- implementerar Loadable

Truck
- *Är en abstrakt klass som är en vehicle men har ett flak*
- ÄR en Veichle
- HAR RampWithStates

TruckPlatform
- *interface som identifierar en plattform*

Vehicle
- *Definerar ett fordons genom attributen nrDoors, enginePower, currentSpeed, Color color, modelName, xpos, ypos, direction*
- Är en abstrakt klass (alltså kan inte skapa ett objekt av sig själv)
- Implementerar Movable
- Implementerar Positionable

Volvo240
- Skapar Volvo240 objekt
- ÄR en vehicle

Uppgift 4
Förbättringar att göra:
CarView kan förbättras, köra lite dekomposition och skapa två klasser istället för en. En klass för knapptryck och en för fönstret. Lättare och bättre kodmässigt, samt bättre med avseende på SoC.

collisionFrame i actionperfomed-metoder bör flyttas på, ligga annanstans. enligt SoC. samma med workshop collision. Vi skapar en ny klass för att hantera kollisoner enligt SRP.

Skapa en ny klass “main” som startar upp programmet. Dvs flytta ut main från carController.

Refaktoriseringsplan Main:

1. Vi vill skapa en ny klass Main som ansvarar för att starta upp programmet genom att delegera uppgifter till CarController, CarView, samt skapa objekt av typen Volvo240, Saab95, Scania och CarWorkshop.

2. Uppdatera CarController så att den inte längre är ansvarig för att starta upp programmet och CarView.

Refaktoriseringsplan “ButtonPanel”:

1. Skapa en klass för knapptryck (ButtonPanel):

2. Uppdatera klassen för fönstret (CarView):

3. Anpassa CarView för att använda de nya klasserna:

Refaktoriseringsplan “CollisionHandler”:

1. Skapa en ny klass kallad CollisionHandler:

2. Flytta collisionFrame-metoden till CollisionHandler:

3. Flytta workshopCollision-metoden till CollisionHandler:

4. Anpassa CarController för att använda CollisionHandler: