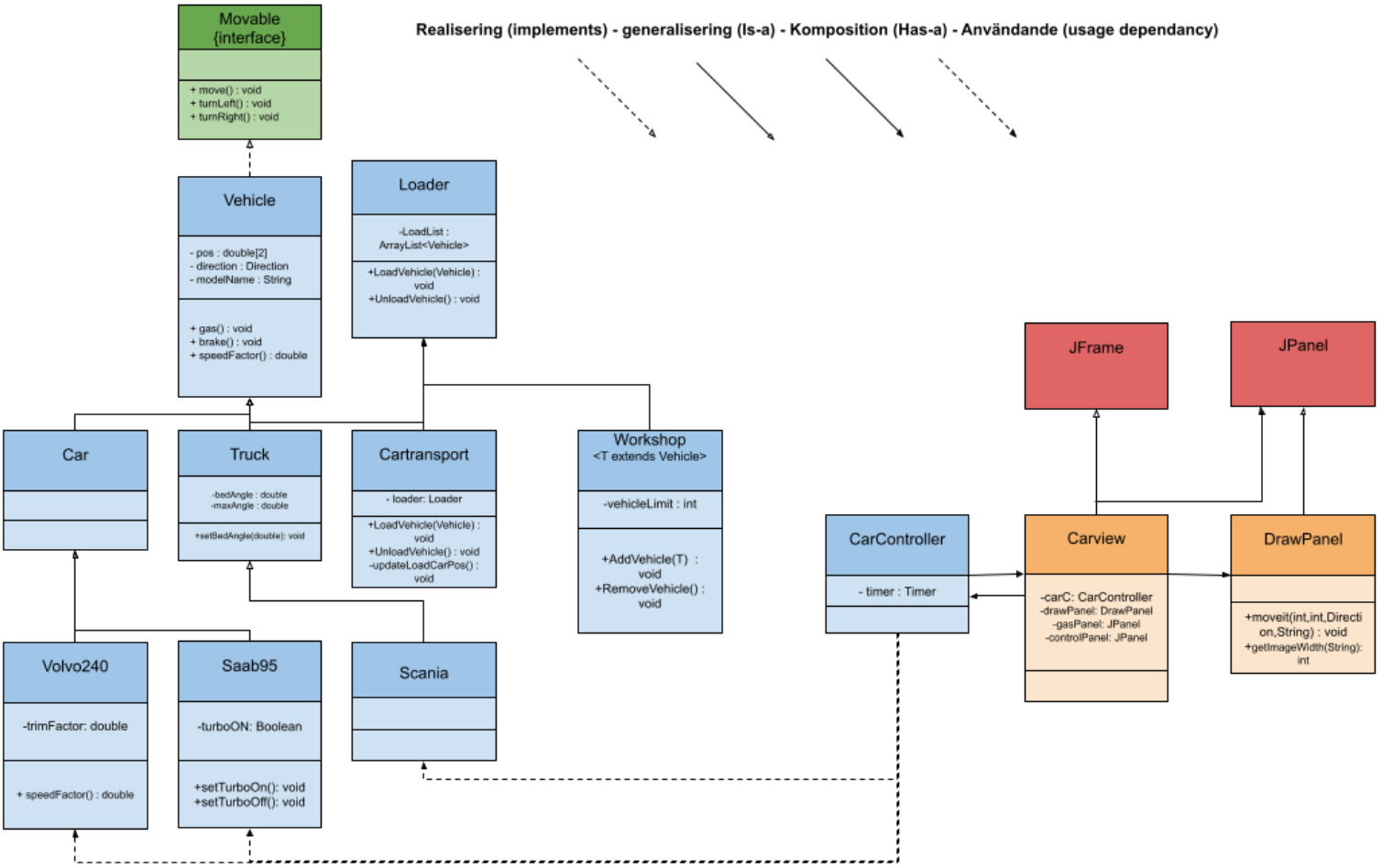

Realisering (implements) - generalisering (Is-a) - Komposition (Has-a) - Användande (usage dependancy)
Movable
{interface}
+ move() : void
+ turnLeft() : void
+ turnRight() : void
Vehicle
- pos : double[2]
- direction : Direction
- modelName : String
+ gas() : void
+ brake() : void
+ speedFactor() : double
Loader
-LoadList : ArrayList<Vehicle>
+LoadVehicle(Vehicle) : void
+UnloadVehicle() : void
JFrame
JPanel
Car
Truck
-bedAngle : double
-maxAngle : double
+setBedAngle(double): void
Cartransport
- loader: Loader
+LoadVehicle(Vehicle) : void
+UnloadVehicle() : void
-updateLoadCarPos() : void
Workshop
<T extends Vehicle>
-vehicleLimit : int
+AddVehicle(T) : void
+RemoveVehicle() : void
CarController
- timer : Timer
Carview
-carC: CarController
-drawPanel: DrawPanel
-gasPanel: JPanel
-controlPanel: JPanel
DrawPanel
+moveit(int,int,Direction,String) : void
+getImageWidth(String): int
Volvo240
-trimFactor: double
+ speedFactor() : double
Saab95
-turboON: Boolean
+setTurboOn(): void
+setTurboOff(): void
Scania

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

Vilka beroenden är nödvändiga?

De beroenden som är nödvändiga är de som ligger inuti våra delkomponenter. T.ex. de beroenden mellan Car, Truck, Vehicle och Movable samt de klasserna som ärver utifrån Car och Truck. Likt detta kan vi se en komponent med nödvändiga beroenden mellan Loader, CarTransport, Workshop och Vehicle. Dessa två komponenter ligger i ett package och vi har ett annat package som bygger på det första. De har beroenden mellan sig men dessa är endast nödvändiga om vi vill ha ett grafiskt gränssnitt.

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

Det man skulle kunna argumentera för är att man tar bort Car klassen och gör så att Volvo240 & Saab95 extend:ar Vehicle istället. Varför vi valt att ha kvar Car klassen trots att den är tom är för att bibehålla möjligheter inför framtiden (OCP), dessutom om vi tar bort Car nu kan det vara jobbigare att implementera den i framtiden.

Finns det starka beroenden som är nödvändigt?

De till Loader samt de till Vehicle. Diskutabelt om Loader’s beroenden är starka.

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

Vi bryter mot SrP (Single responsibility principle) som säger att varje klass endast skall ansvara för en sak. Till exempel skulle varje egenskap Vehicle har (turnRight/Left, gas, brake, styrenhet) vara en egen klass som instansieras i Vehicle. Loadern skulle kunna splittras i två delar: en loader och en unloader. Dock på den här skalan fungerar arv “good enough”.

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

Vilka ansvarsområden har era klasser?

Klasserna CarController, Carview och Drawpanel ansvarar tillsammans för programmets grafiska representation. Klasserna Volvo240, Saab95, Scania och Cartransport används för att skapa instanser av olika typer av bilar, deras gemensamma superklass är Vehicle. Komponenten loader används av klasserna Cartransport och Workshop för hantering av en lista som kan förvara Vehicle objekt.

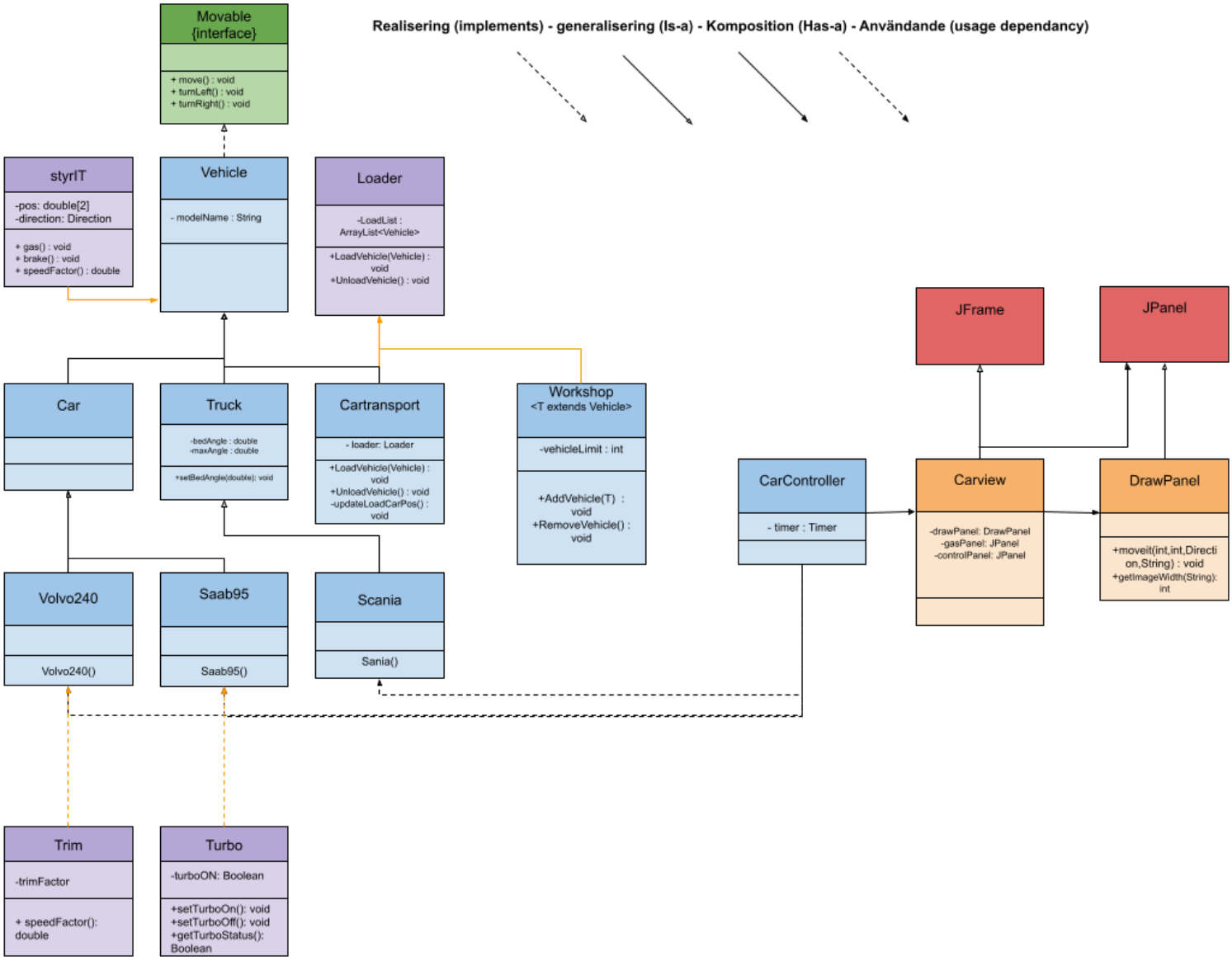
Vilka anledningar har de att förändras?

Genom att förbättra/förändra vår kod kan vi skapa en mer utvecklar-vänlig miljö där flera kan arbeta parallellt. Detta skulle enklast göras genom att stärka SRP i vår kodbas, till exempel skulle vi kunna införa fler komponenter och bryta ned större klasser till fler mindre klasser som har mer specificerade användningsområden.

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

På Vehicle klassen skulle dekomposition kunna tillämpas där dess egenskaper som styrenheten och funktionalitet som gas och break kan skrivas om till komponenter som Vehicle sedan använder. Dessa förändringar skulle främja både SoC och SRP.

Refraktionsplan



Vår plan är att refaktorisera vår kod enligt UML-diagrammet ovan. Där utgår vi då från SoC & SRP och övergår till ett mer kompositionsbaserat system. Planen är fool-proof den går absolut att genomföra parallellt, dessutom så bidrar den nya strukturen till att parallellt arbete är enklare än någonsin! CarController kommer här efter innehålla all funktionalitet för knapparna vilket tar bort vår “double dependency”.