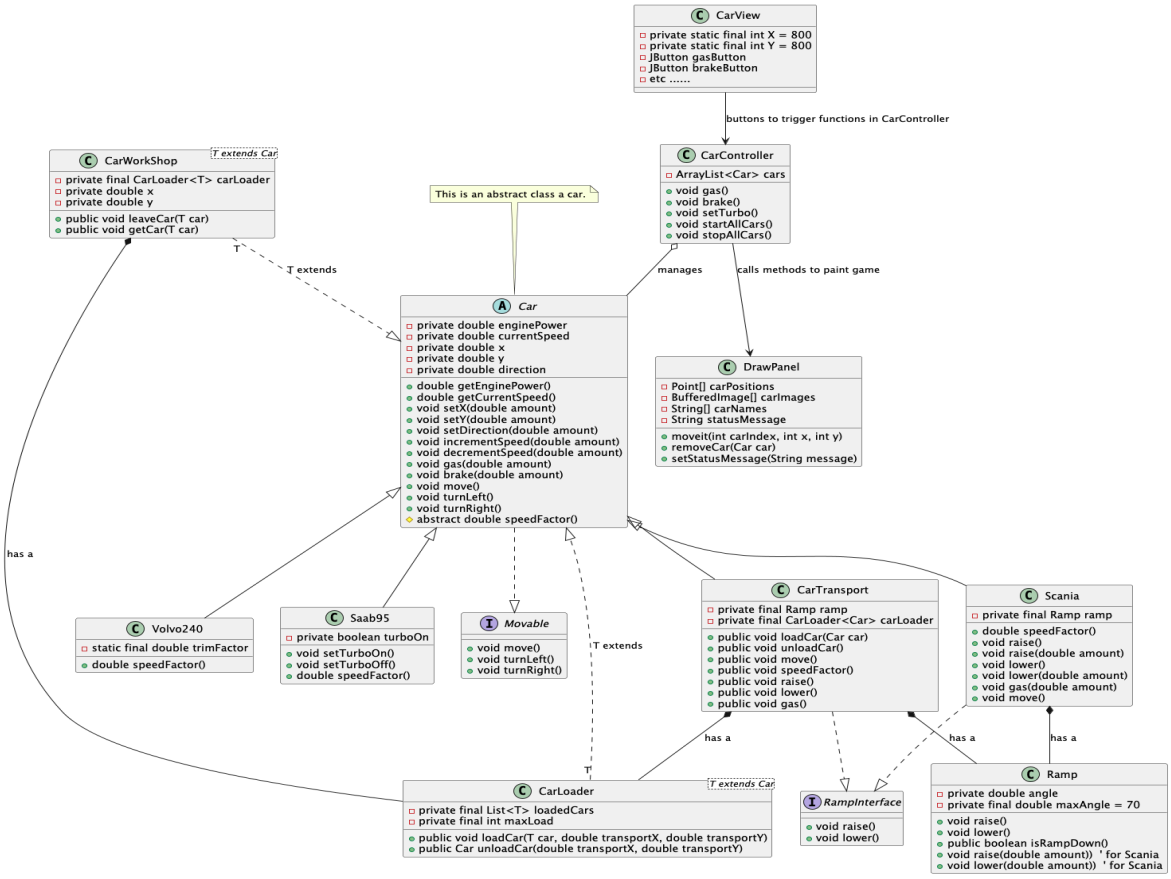
Begrepp	Typ av relation	UML-symbol	Java-exempel
Associering	"Har en"	Fylld Linje med Diamant i slutet	`private Car car;`
User-Dependency	"Använder"	Streckad pil `------>`	`CarController använder t.ex Car`
Generalisering	"Är en"	Ihålig triangelpil ▷	`class Volvo extends Car`
Realisering	"Kan göra"	Streckad triangelpil ▷	`class Car implements Movable`

Uppgift 2



1. **Vilka beroenden är nödvändiga?** : Car Transport och Scania är beroende av Ramp eftersom de behöver en ramp för att lasta bilar, vilket gör relationen nödvändig. Car WorkShop och Car Transport är även beroende av CarLoader för att lasta bilar. Car är beroende av Movable för att kunna förflytta sig. Detta är nödvändiga beroenden som skapar hög cohesion då varje klass har sitt egna ansvar.

2. **Vilka klasser är beroende av varandra som inte borde vara det?**
   Vi anser att alla klassernas relationer är nödvändiga men de beroenden som finns skulle kunna förbättras då några är onödigt starka.

3. **Finns det starkare beroenden än nödvändigt?**Ja det finns starkare beroenden än nödvändigt. Till exempel borde det inte finnas ett direkt beroende mellan volvo, saab, scania, car transport med Car. Hade blivit lägre coupling om det

fanns ytterligare två klasser som en passengerCar (volvo,saab) och sedan en Truck (Scania, Cartransport). På detta sätt kommer inte en ändring i koden påverka alla bilar ifall man vill ändra på t.ex passenger cars.

4. **Kan ni identifiera några brott mot övriga designprinciper vi pratat om i kursen?**
    a. **Single Responsibility Principle** (SRP)?: Car fyller nu två funktioner, en för förflyttning i x och y led, och för att lagra information om bilen som, cardoors, color etc. Vi kan minimera bilens uppgifter och skapa högre cohesion genom att skapa en ny klass som Car ärver av, exempelvis "Vehicle” som har ansvar för förflyttning och koordinater, samt lagring av information generell för alla typer av fordon.
    b. **Open/Closed Principle (OCP) ?:** Om man skulle vilja lägga till ett annat typ av fordon så hade det ej varit möjligt då programmet endast är byggt för bilar.
    c. **LSP-brott (Liskov Substitution Principle Violation):** Till exempel ärver Car Transport från Car och har ett annat beteende än en vanlig bil: den transporterar andra fordon och har en ramp. CarTransport måste dessutom förhindra vissa metoder i Car, som att köra när rampen är nere vilket bryter mot principen.

Uppgift 3

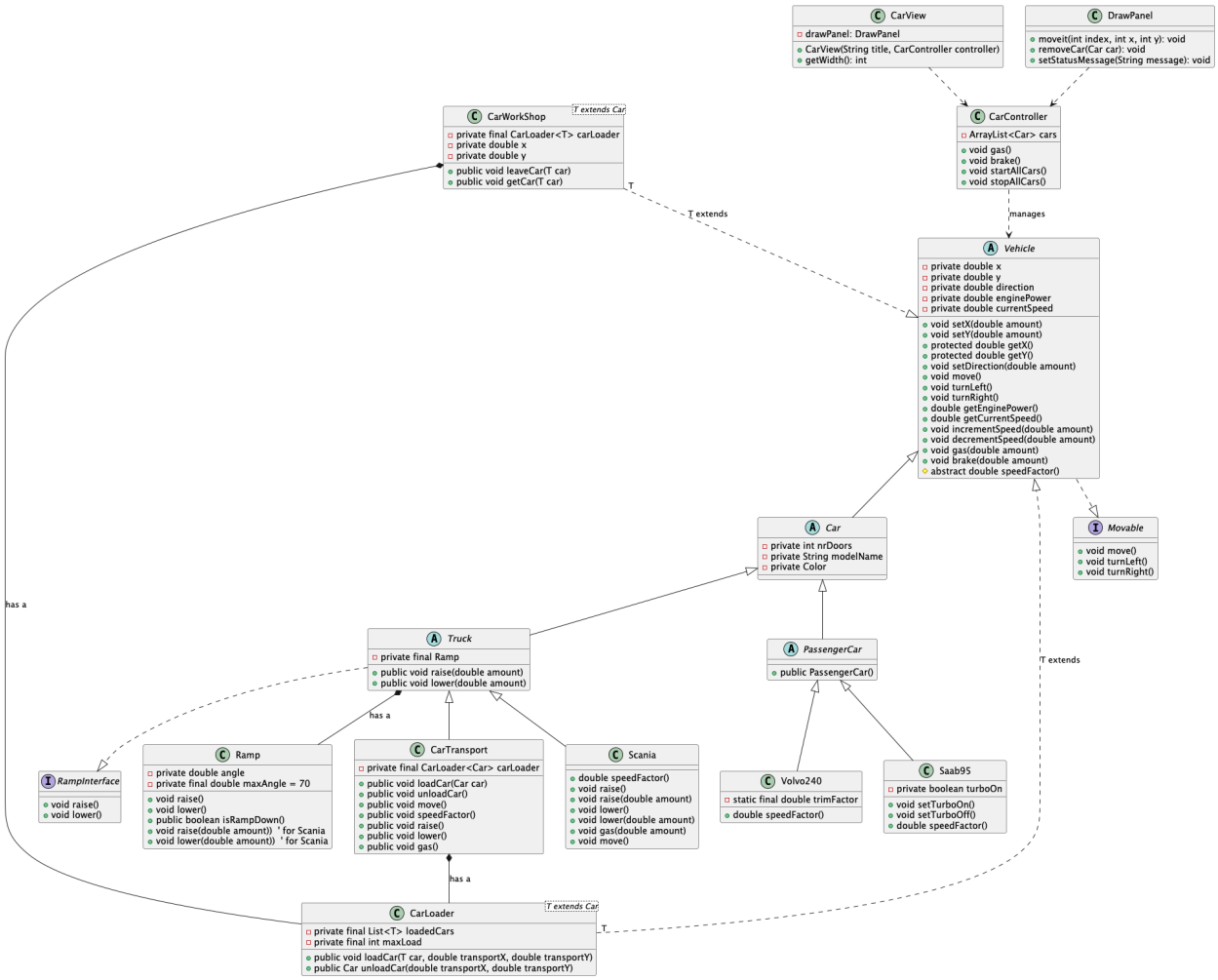
Separation of Concerns (SoC) och Single Responsibility Principle (SRP)

SoC handlar om att dela upp systemet i olika områden som var och en hanterar en specifik funktionalitet. SRP handlar om i princip samma sak då varje klass endast ska ha ett ansvar.
I ditt system kan vi identifiera följande concerns/ansvarsområden:

- **Hantering av bilar**: `Car`, `Volvo240`, `Saab95`, `Scania`, `CarTransport`.
- **Hantering av lastning/avläsning**: `CarLoader`.
- **Hantering av lagring av bilar:** `CarWorkShop`
- **Hantering av ramper**: `Ramp`, `RampInterface`, `Scania, CarWorkshop .`
- **Hantering av rörelse**: Car/`Movable`.
- **Användargränssnitt**: `CarView`, `DrawPanel`.
- **Kontroll och logik**: `CarController`.

Som förklarat i tidigare uppgift hanterar Car nu både förflyttning och information om bilar. Det vill säga två ansvar som kan delas upp till två olika klasser. Även en ytterligare en klass kan läggas till (Truck) som endast sköter Ramp, raise/lower som CarTransport och Scania ärver.

Uppgift 4



Rerefaktoriseringsplan

1. **Skapa Vehicle klass - som sköter förflyttning/position egenskaper (flytta över de Car funktionerna/variablerna till Vehicle, samt gör så Vehicle implementar movable interface)**
2. **Skapar en Truck-klass som ärver Car, vidare ärver CarTransport och Scania Truck. Truck har alla rampfunktioner + interfacet**

3. **Skapa en PassengerCar klass som nu Volvo/Saab kommer ärva ifrån.**

Uppdelning av plan:

1. Jubin skapar Vehicle, Car klassen/nya kopplingarna.
2. Mats skapar Truck, PassengerCar klassen/nya kopplingarna

På detta vis kan båda vi arbeta parallellt, då inget arbete som görs nu kommer påverkan den andra.