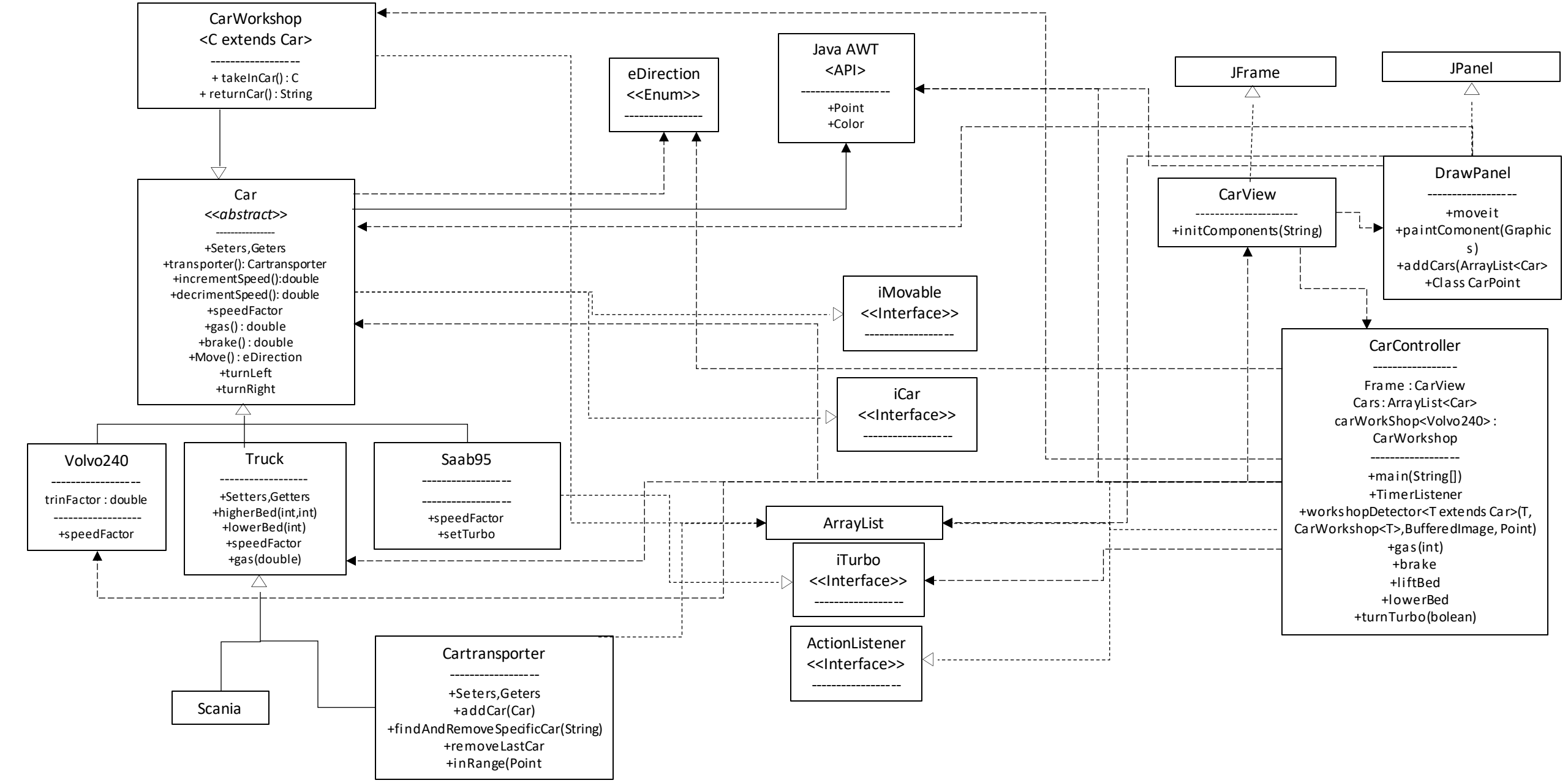
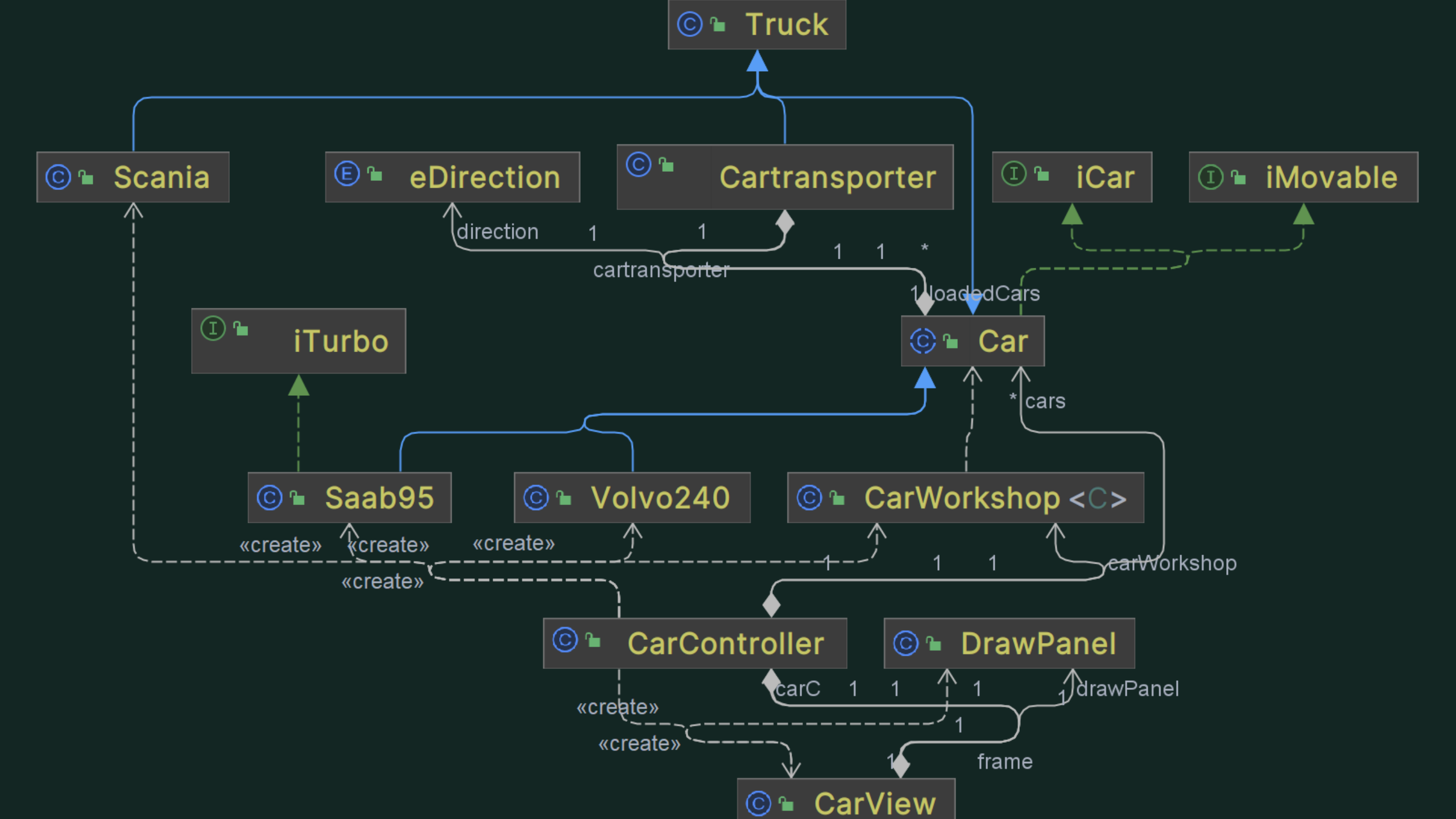

CarWorkshop
<C extends Car>
+ takeInCar() : C
+ returnCar() : String
eDirection
<<Enum>>
Java AWT
<API>
+Point
+Color
JFrame
JPanel
Car
<<abstract>>
+Seters,Geters
+transporter(): Cartransporter
+incrementSpeed():double
+decrimentSpeed(): double
+speedFactor
+gas() : double
+brake() : double
+Move() : eDirection
+turnLeft
+turnRight
CarView
+initComponents(String)
DrawPanel
+moveit
+paintComonent(Graphics)
+addCars(ArrayList<Car>
+Class CarPoint
iMovable
<<Interface>>
iCar
<<Interface>>
CarController
Frame : CarView
Cars : ArrayList<Car>
carWorkShop<Volvo240> :
CarWorkshop
+main(String[])
+TimerListener
+workshopDetector<T extends Car>(T,
CarWorkshop<T>,BufferedImage, Point)
+gas(int)
+brake
+liftBed
+lowerBed
+turnTurbo(bolean)
Volvo240
trinFactor : double
+speedFactor
Truck
+Setters,Getters
+higherBed(int,int)
+lowerBed(int)
+speedFactor
+gas(double)
Saab95
+speedFactor
+setTurbo
ArrayList
iTurbo
<<Interface>>
ActionListener
<<Interface>>
Scania
Cartransporter
+Seters,Geters
+addCar(Car)
+findAndRemoveSpecificCar(String)
+removeLastCar
+inRange(Point


Truck
Scania
eDirection
Cartransporter
iCar
iMovable
direction
1
1
1
1
*
cartransporter
1
loadedCars
iTurbo
Car
* cars
Saab95
Volvo240
CarWorkshop <C>
«create»
«create»
«create»
«create»
1
1
1
carWorkshop
CarController
DrawPanel
carC
1
1
1
1
drawPanel
«create»
«create»
1
1
frame
CarView

Uppgift 2 frågor:

1. Vilka beroenden är nödvändiga?
2. Vilka klasser är beroende av varandra som inte borde vara det?
3. Finns det starkare beroenden än nödvändigt? -
4. Kan ni identifiera några brott mot övriga designprinciper vi pratat om i kursen? -

1: Dem beroende som är nödvändiga är:

- Car och dess sub och super typer som vi utväcklat enledigt DIP
- Interface beroende samt beroende till uteligande classer, ArrayList, Jframe mm.
- carController är beroende på Car så att vi kan styra bilarna

2: car controller och carview är har beroende pilar på båda hållen, samma med cartransporter och car så vi kan kopla positionen av cartransporter för den bil som förflyttas.

3: carcontroller är ju den som är beronde på mest, men även den som styr alla komplingar mellan bilaran, så nej dem beroende vi har är

4:
I drawpanels har vi redan fixat open close problemet, där man inte kunde lägga till flera bilar utan att hård koda in nya punkter samt bild och liknande så vi har gett att alla bilar har en bild med sig samt använder vi os av array för att lättare lägga till nya bilar. Samt gjorde vi en ny klass CarPoint som är en touple och tar in en point samt en car sedan har vi en arraylist med carpoints

Uppgift 3

- Analysera era klasser med avseende på Separation of Concern (SoC) och Single Responsibility Principle (SRP).
  - Vilka ansvarsområden har era klasser?

Car-hanterar bilens dörrar, motorstyrka, hastighet, färg, position och riktning ect.

Cartransport-lasthantering, rampkontroll, rörelsekontroll med last, kapacitetkontroll och räckviddsbedömning.

CarWorkshop-bilhantering i verkstaden, kapacitetskontroll och sökning efter specifik bil

Truck-rampkontroll, rörelsebegränsningar, vinkeljustering av flaket.

CarController-hanterar huvudflödet i applikationen,användarinmatning och timerhändelser, direkt manipulation av bilar och interagerar med "CarWorkshop".

  - Vilka anledningar har de att förändras?

Car-rörelselogik, interaktion med andra objekt, grafiskrepresentation

Cartransport-lastkapacitetsregler, rampenfunktionalitet, rörelsebeteende med last och interaktionen med car-Objekt.

CarWorkshop-policyförändringar för verkstadens kapacitet och optimering av sökfunktionaliteten.

Truck-Säkerhetsbestämmelser, rörelsemakaniska förbättringar och ändringar i föräldrarklassen "Car".

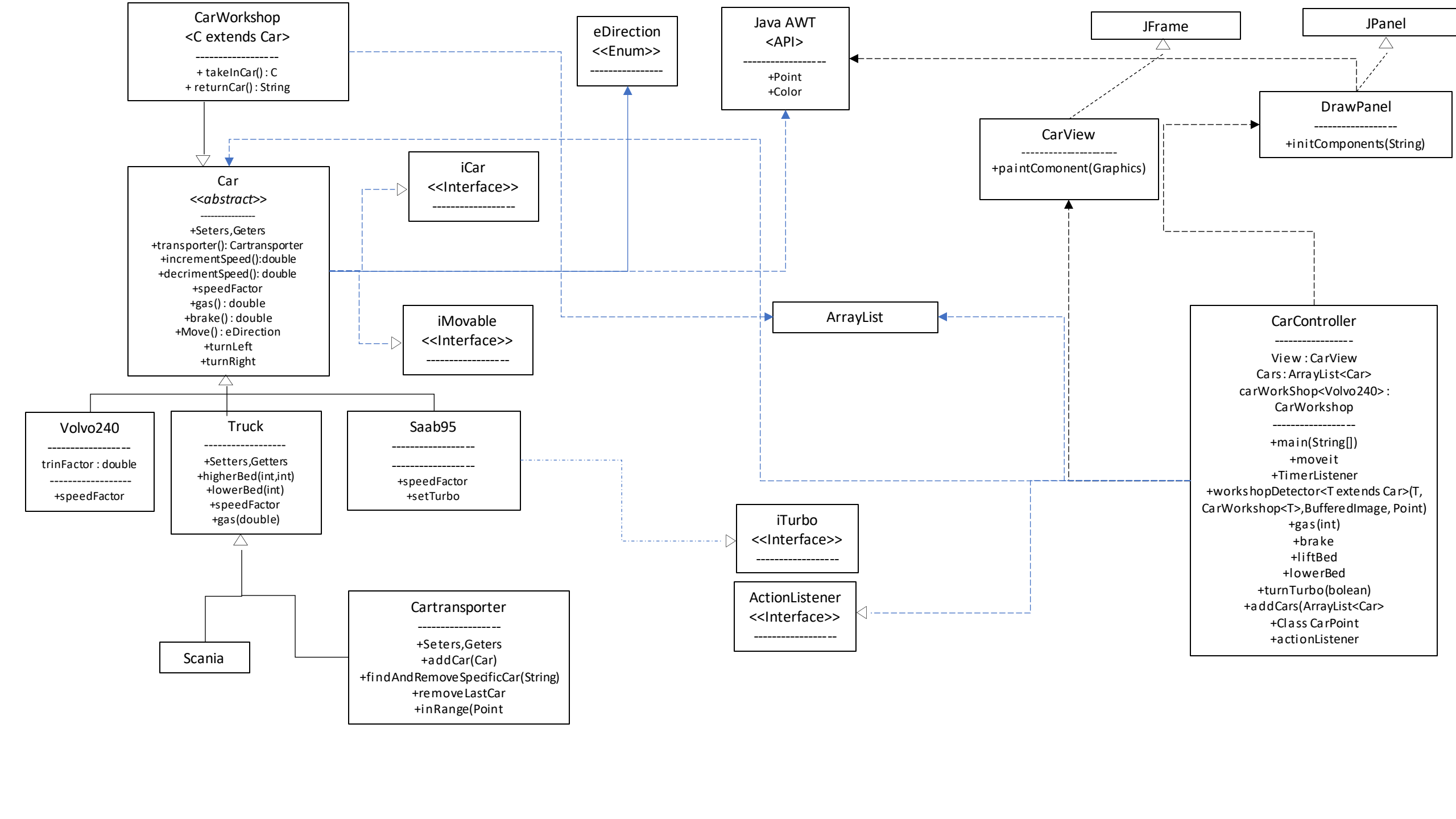
CarController-lägga till nya typer av användarinteraktioner, ändra logiken för hur bilar uppdateras och modifiera hur verkstaden hanteras.

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

Car-Separera funktionalitet relaterad till rörelse och riktning från bilens beteende tex bilens rörelsekontroll och riktningshantering till en eller flera separata klasser för att minska klassernas storlek och komplexitet vilket gör det lättare att underhålla och utveckla bilrelaterade funktioner.

Cartransport-Skilja ut hantering av rampfunktioner och belastningskapacitet i en egen klass eller modul för att isolera transportfunktioner från billogik, vilket gör det enklare att hantera ändringar i lastningsmekanismen och tillämpa olika strategier för lasthantering.

Truck- flakhanteringen kan eventuellt brytas ut i en separat klass. Detta skulle kunna inkludera all logik för att hantera flakets vinkel och rampens position. Detta för att förfina ansvarssättningen inom lastbilshantering och underlätta för nya funktioner eller beteenden som rör lastbilens lastområde utan att påverka fordonshanteringen.


CarWorkshop
<C extends Car>
+ takeInCar() : C
+ returnCar() : String
eDirection
<<Enum>>
Java AWT
<API>
+Point
+Color
JFrame
JPanel
DrawPanel
+initComponents(String)
CarView
+paintComonent(Graphics)
Car
<<abstract>>
+Seters,Geters
+transporter(): Cartransporter
+incrementSpeed():double
+decrimentSpeed(): double
+speedFactor
+gas() : double
+brake() : double
+Move() : eDirection
+turnLeft
+turnRight
iCar
<<Interface>>
iMovable
<<Interface>>
ArrayList
CarController
View : CarView
Cars : ArrayList<Car>
carWorkShop<Volvo240> :
CarWorkshop
+main(String[])
+moveit
+TimerListener
+workshopDetector<T extends Car>(T,
CarWorkshop<T>,BufferedImage, Point)
+gas(int)
+brake
+liftBed
+lowerBed
+turnTurbo(bolean)
+addCars(ArrayList<Car>
+Class CarPoint
+actionListener
Volvo240
trinFactor : double
+speedFactor
Truck
+Setters,Getters
+higherBed(int,int)
+lowerBed(int)
+speedFactor
+gas(double)
Saab95
+speedFactor
+setTurbo
iTurbo
<<Interface>>
ActionListener
<<Interface>>
Scania
Cartransporter
+Seters,Geters
+addCar(Car)
+findAndRemoveSpecificCar(String)
+removeLastCar
+inRange(Point

refraktoreseringsplan.

- 1. Skapa en ny class "Tuple" som är en tuple i Drawpanel, som innehåller variablerna car och point. Lägga till de bilar i en Arraylist av typen CarPoint. Med så blir det lättare att läga till nya bilar och minimerade kod längden.
- 2. Lyfta ur BufferedImage för objekt i DrawPanel, och skapa det när vi skapar ex. en Car. Open-closed, DrawPanel kan nå bilderna men inte ändra.
- 3. DrawPanel hanterar statiska element. Vi flyttar initcomponent metoden till Drawpanel. (För att följa Single resp. Principle.)
- 4. CarView får Drawpanels paintcomponent. hanterar dynamisk, där instansobjekts bilder uppdateras under programmets körning.
- 5. Carcontroller får actionlistener från CarView. Följa SRP så att all typ av actionlistening sker här. Kallar CarView så att våra dynamiska instansobjekt uppdateras.
- 6. ta bort movit from drawpanel och ha den i carcontroller i och med att drawpanel enda uppgift är de statiska gränssniten och inte position för bilarna