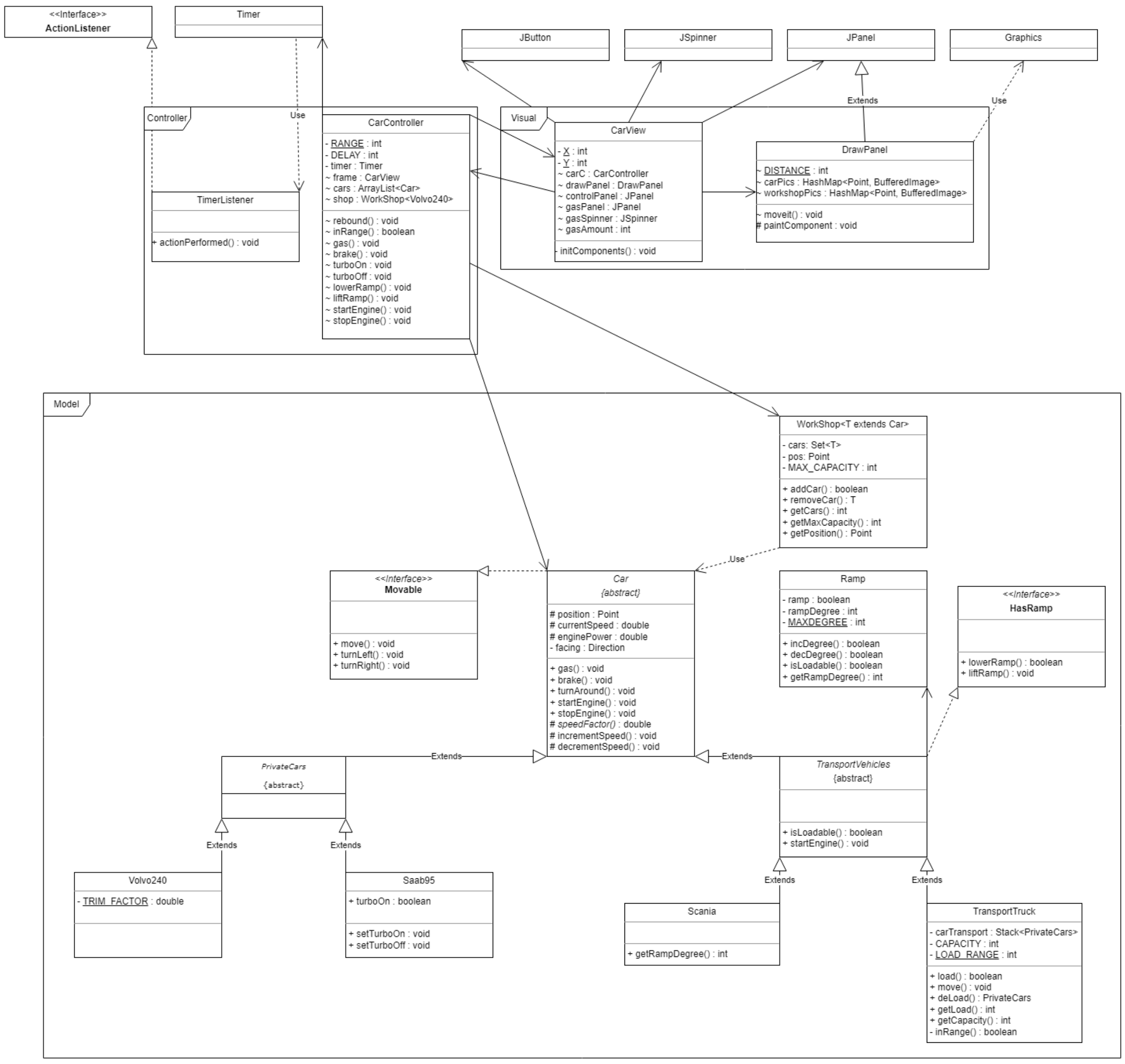
Group 28
15/02-24

# Laboration 3: Design och principer

## 2) UML-Diagram for present design: (LINK)



- Unnecessary relations between CarController and CarView -> Use main method?
- Remove hard-coding of car-/image-positions, add a relation between them.

## 3) Area of responsibility:

CarController: Has the responsibility for the changes that affect the cars such as gas/stop etc. Also contains a Timer object as a tick to update the program regularly. Also calls on DrawPanel to update the position of images to respond to updates of the positions of the car objects.

CarView: CarView handles the visual representation of the cars in the user interface as well as handling the button corresponding to an action such as “Stop All Cars”.

Car: An abstract class that defines the default attributes and behavior for all vehicles.
PrivateCars: An abstract subclass to Car whose purpose is to distinguish PrivateCars from TransportVehicles. Serves to make sure that TransportTruck cannot load TransportVehicles (size issue).

TransportVehicles: An abstract subclass to car that represents transport trucks that have unique attributes and behaviors that differ from PrivateCars.

Saab, Volvo, Scania, TransportTruck: Are all subclasses to PrivateCars/TransportVehicles that contain specific methods unique to themself and self only.

WorkShop: A generic type class to represent a workshop for cars that can store a certain amount of cars and then be returned from the shop, only containing specific methods for storing and returning a car which is good for SoC rule.

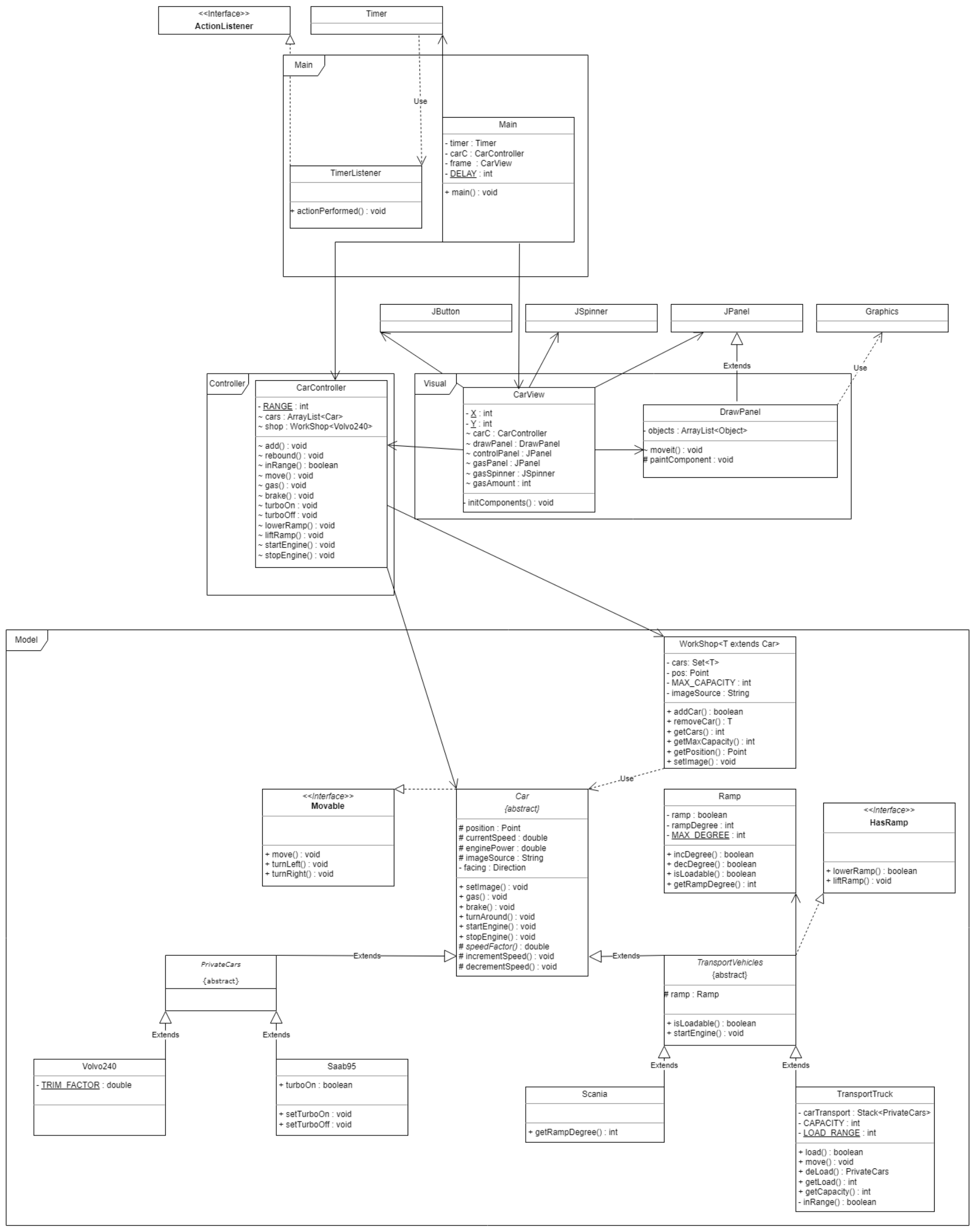
What could happen that would require a change?
If we wanted to add more cars to our program we would have to update both CarController and DrawPanel to set their positions by hand.

if we wanted to add an implementation of a truck without a Ramp, we would have to redefine our current composition between Ramp and transport vehicles. Maybe create a superclass Ramp that has different types of ramps as subclasses, then the new creations of trucks would only need to specify which kind of ramp they have/none or create a new Ramp subclass to fit the specific criterias.

Group 28
15/02-24

## 4) UML-Diagram for improved design:

LINK



Group 28
15/02-24

**Refactorization plan:**

Improving link between image and object position, removing hard-coding:

- Add String variable *imageSource* to *Car* class.
- Add String variable imageSource to WorkShop class.
- Add setters to Car and WorkShop to set an image source.
- Redesign *DrawPanel* constructor to take an ArrayList of Objects as argument.
- Have moveit() in DrawPanel take a Car as argument
- Have CarView take a DrawPanel as argument in its constructor

Removing unnecessary relations, decomposing *CarController*:

- Add Main-class to hold the constructor for running the program
- Add Timer-, CarController- and CarView-objects to main.
- Refactor the code of the actionPerformed method of the TimerListener, so that calls to the appropriate components happen in Main, but the code to be run lives in the components (for example, add a move() function to CarController that moves every car, which can be called from Main).
- Add intermediate functions to CarView that pass the call on to DrawPanel, such as moveit() and repaint(), as these are currently accessed via CarView.DrawPanel.moveit() etc.