Group 12

Freja Bermann, Sebastian Kölbel, Patrik Wahlgren

Refactoring plan (Lab 2A):

(I)Movable/Positionable:

Change the relationship between Movable and Car/Boat to a bridge pattern. (One - way from car to movable)

Extract all the position related attributes from Movable to a new class Positionable.

Movable should have a “has-a” relationship with Positionable.

There should be a way to update the Positionable, either create new Positionable or update old one not sure which would be best.

Create setters/getters as necessary.

(I)Transportable/(I)Transporter:

Change the relationship between Transporter/Transportable and Workshop/Ferry/CarTransport/PassengerCar to a bridge pattern. (One - way, as before)

ITransporter should have as few methods as possible and they should only represent the basic functions of a transporter and any specific implementations such as first in last out should be handled solely by the Transporter.

Transporter/Transportable should be able to match each other’s position and direction as well as unloading to a specific position (ex. a few coordinates behind / ahead of the Transporter) solely based on data sent from a Workshop, Ferry, etc \*without\* relying on how any other module works.

Idea currently to achieve the above point is that both Transporter and Transportable have a “has-a” relationship with a Positionable that the Workshop/Ferry/etc can write onto while PassengerCar can both write onto and read theirs. The Transporter can update the Transportables Positionable.

Workshop can’t move but should till have a position, so it should have a “has-a” relationship directly with Positionable.

Create setters/getters accordingly.

Workshop/Ferry/CarTransport/PassengerCar:

Recreate any specific implementations that were previously inside Transporter/Transportable.

Setters/getters.

CarController/CarView/DrawPanel:

Having two references to the same list in both CarController/DrawPanel feels bad but it does “match” each Point to a Car effectively. Maybe we can find a better way to do that?

Less sure about this one overall.

Car/Boat/ Model:

Create a bunch of setters/getters to allow for the multiple bridges.

(Optional) Extract all the tertiary info (like nrDoors, color, etc) about a Car/Boat into a new class Model that also uses a bridge pattern. This probably isn’t necessary, but it would be consistent with the other changes. It would also allow us to get rid of the distinction between Boat and Car and just make it a vehicle class without any awkward “nrDoors” fields on a boat.

CarController/CarView/DrawPanel

Try to separate out timer and main into a new “Main” class.

Create a new class to hold Actors and allow for ObserverPattern.

DrawPanel is the observer and repaints the images on screen according to the data it observes.

CarView creates/handles GUI and listens for actions then relays these actions to CarController.

CarController updates the actors held in the new class above.

Parallel development:

It should be safe to refactor each part separately if the representation of coordinates and direction remains as 3 doubles and setters/getters are well made enough. While our code is a bit messy it is still somewhat modular.