NULÄGET

Något vi ser just nu är att det är väldigt hög coupling mellan CarView och CarController. De håller instanser av varandra i klasserna och metoder studsar mellan dem. Det är även låg cohesion i CarController, eftersom den både hanterar saker från CarView och försöker delegera vidare arbetsuppgifter på detaljerad nivå. Detta bryter mot Single Responsibility Principle samt Dependency inversion principle.

- Dubbelpil mellan CarView och CarController
- MoveIt onödigt komplicerad implementering
- Måste “måla om” i TimerListener istället för att bara flytta objekten
- Klasserna CarView, CarController och DrawPanel. Klasserna är beroende av instanser av en eller båda av de andra klasserna. Det innebär hög coupling.
- Låg cohesion i CarController. Den hanterar både kommunikation mellan anrop - objekt, grafik - anrop och grafik-uppdateringar. Bryter mot single responsibilty principle.
- MoveIt är en väldigt specifik implementering som därför inte är utökbar i framtiden (man måste lägga till en MoveIt-funktion för varje ny bilmodell och carshop)

ÄNDRINGAR

1.) Lägga till funktioner “drawVehicles” & “drawShops” i DrawPanel, de itererar listor med alla objekt och ritar ut respektive bilder vars adresser finns sparade i objekten, ta bort “moveIt” funktionerna. För att göra detta möjligt måste det skapas ett interface “MoveImage” mellan VehiclesAndShops och DrawPanel. I DrawPanel så kan man då endast få tillgång till getX, getY och getImage, vilket förhindrar den från att komma åt andra metoder. (Måste delat upp CarController först)

2.) Bryta ner CarController till 3 klasser; VehiclesAndShops, TimerListener och ButtonControl. (går att göra parallellt)
    a.) VehiclesAndShops ska hantera anropen som användare gör och applicera dessa på objekten. (Exempel: Om användaren matar in gas med amount 100 i spinnern är det i klassen VehiclesToMove some gas kallas på objekten)
    b.) TimerListener är utbruten från att vara en privat klass i CarController till en egen separat klass
    c.) ButtonControl ska vara kommunikatören mellan CarView, som får anropen från användaren, och VehiclesToMove. CarController meddelar alltså en VehiclesToMove att användaren vill att saken ska hända.

Denna förändring görs för att bryta ner den komplexiteten i CarControllern och låta klassen följa Single Responsibility Principle.

3.) Skapa en klass “RunProgram” som innehåller main-metoden och kör hela programmet. Denna är ganska lik main-metoden i originalet fast nu med hänvisningar till nya klasser (Måste ha gjort alla ändringar först)
4.) Refaktorisera en hel del i “actionsPerformed” i “TimerListener”

Fixa interface mellan VeheiclesAndShops och DrawPanel, med getX, getY, getImage, få interface som en typ i lista? (Måste ha gjort ändringar först)

Våran ändringsplan kan anses vara relativt sekventiell. Detta eftersom att ändringarna i steg 2, att bryta ner CarController i flera funktioner, är essentiellt för att på ett adekvat sätt kunna göra de andra ändringarna. Vad som kan däremot göras parallellt (efter steg 2 är gjort) är ändringarna i 1 och 4, eftersom ändringarna i de stegen inte kommer påverka varandra.
För steg 3, RunProgram, måste alla andra ändringar har gjorts. Detta eftersom att våran main-metod måste kunna skrivas på det aktuella programmet och därför måste alla andra ändringar göras innan RunProgram implementeras.