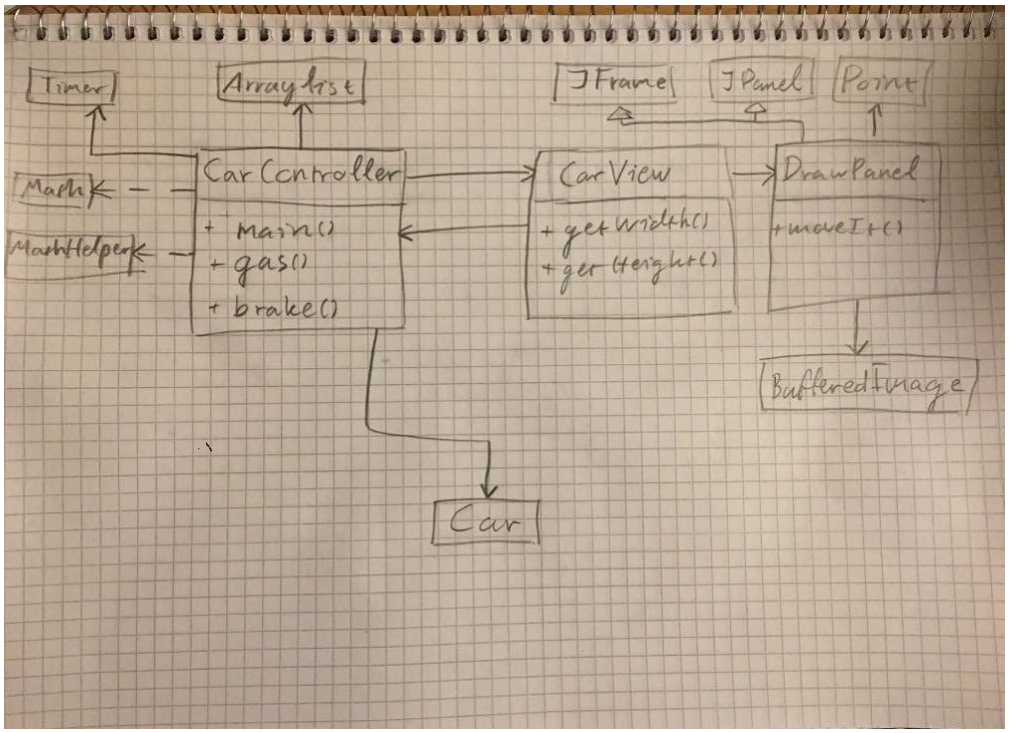
TDA550 Laboration 2, del A

Rickard Ström, Samuel Sandelius, Elmer Lingestål

4 december 2020

2.



Figur 1: UML innan förändringar

CarViews beroende till CarController är nödvändigt eftersom vi knappast kan ha grafiska bilar utan dess betende. Samtliga beroenden till externa bibliotek som Timer, JFrame och JPanel behöver nog vara kvar.

CarController är beroende av CarView, vilket inte borde vara nödvändigt. Att CarView dessutom har en instansvariabel av CarController gör beroendet cirkulärt vilket ska undvikas.

CarControllers beroende till ArrayList är starkare än det borde vara. Detta eftersom ett beroende av List istället hade varit mer flexibelt om man vill byta ut ArrayList. Detta skulle följa principerna av modulär design bättre. Dessutom är det DrawPanel som i nuläget skapar respektive bil samt bilarnas bilder. Detta bör allokeras till en annan del av programmet, antagligen till main där bilarna skapas i början av exkiveringen. DrawPanels enda funktion bör vara att rita enligt SRP och open-closed principle. I nu läget skulle man behöva ändra i DrawPanel för att ändra vilka bilar man ska använda, vilket är ologiskt.

Vidare skulle man inte kunna använda drawPanel för någonting annat än just dessa bilar eftersom det är hårdkodat in i klassen.

3.

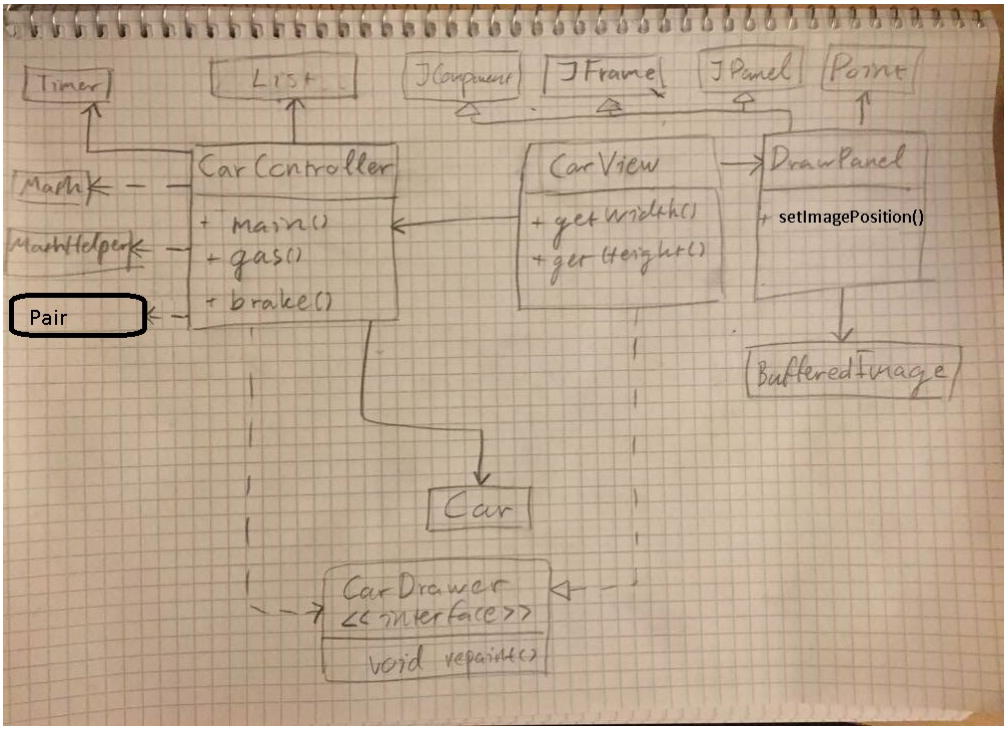
CarController ansvarar för all funktionalitet och kontroll över bilarna och deras beteende. CarView reglerar koppling mellan bilarna och vad som visas. DrawPanel ansvarar för utseende och grafiken som visas.

Att CarController är beroende av CarView strider mot objektorienterade principer och att dessutom beroendet är cirkulärt är ett mycket dåligt tecken. Det cirkulära beroendet kan hävas genom att införa ett interface CarDrawer som overridar JComponents repaint(). CarView kan sedan implementera detta interface och CarController kan anropa repaint() genom det. Då är endast CarView beroende av CarController och inte tvärt om. Detta ger oss lägre koppling och följer bättre SoC och SRP.

CarController borde vara beroende av List istället för ArrayList eftersom det öppnar upp för andra typer av listor i framtiden och är ett svagare beroende.

Main klassen borde skapa bilarna samt deras bilder och CarController borde därefter skicka dem tillsammans in i DrawPanel för att rita ut dem. Funktionerna i DrawPanel skulle då behöva modiferias för att kunna ta emot en bild samt x och y position.

4.



Figur 2: UML efter förändringar

1 Nya Designen

I den nya designen i figur 2 har vi tänkt på vissa designprinciper, och haft som mål att uppfylla dessa.

Enligt principen "High cohesion, low coupling"vill man uppnå hög sammanhållnign inom klasser, men få kopplingar däremellan. Det har vi uppnått i högre grad med den nya designen genom att flytta ansvar som att koppla ihop bilder med rätt bil ifrån DrawPanel

till controller genom att skapa ett Pair av en bil och en bildreferens. Metoden moveit i DrawPanel ändras också så att den tar in en position, en bildreferens och lägger dessa i en lista innan den till slut ritar ut alla positioner och bilder den fått ifrån CarController. Det gör att DrawPanel slipper hålla reda på vilka bilar som finns, och att koppla bilder till rätt bil.

Det blir även lägre grad av koppling mellan klasser när cirkelberoendet försvinner mellan CarController och CarView. Detta görs, som beskrivet i uppgifte 2, genom att ta bort CarControllers beroende av CarView. CarController kommer då att deklarera ett interface CarDrawer och utnyttja dess metod repaint() istället för att deklarera en frame av typen CarView. Denna åtgärd samt flera som till exempel utbytt beroende från ArrayList till list, ger tillsammans en högre grad av moduläritet, så att klasserna kan användas (byggas ihop) med andra klasser på ett enklare sätt, och man kan till exmpel byta ut klasser så som CarView eller ArrayList utan att behöva ändra i andra klasser. Dessa åtgärder främjar även Dependency Inversion Principle eftersom Interfaces till exempel är mer abstrakt än en konkret klass.

Det första som skulle behöva genomföras vid omstrukturering av koden är att flytta koden som relaterar till bilarna och bilderna till main klassen, dessutom lägga dem i Pairs. Därefter måste dessutom funktionerna i DrawPanel modifieras för att kunna hantera bildparametrar. Även mycket klientkod i alla klasser måste modifieras eftersom vi kommer att använda Pairs. Alltså om en klass tidigare bara skrivit "Car", måste den nu skriva Pair.getKey()"eftersom "Carnu befinner sig där. Slutligen måste även interfacet för CarController och CarView skapas.

1.1 Parallell utveckling

Refaktoreringen består ändringar på flera olika ställen, men det bör vara möjligt att olika utvecklare refaktorerar olika klasser oberoende av varandra. Efter refaktoreringen har varje klass bestämd uppgift, och enligt planen vet utvecklarna vilken data som skickas mellan klasserna och via vilka metoder. Till exempel kan en utvecklare ändra den tidigare moveit() metoden i DrawPanel till den nya setImagePosition, som tar in position och en filepath till en bild utan att samma utvecklare måste ha kallat på metoden i CarController.