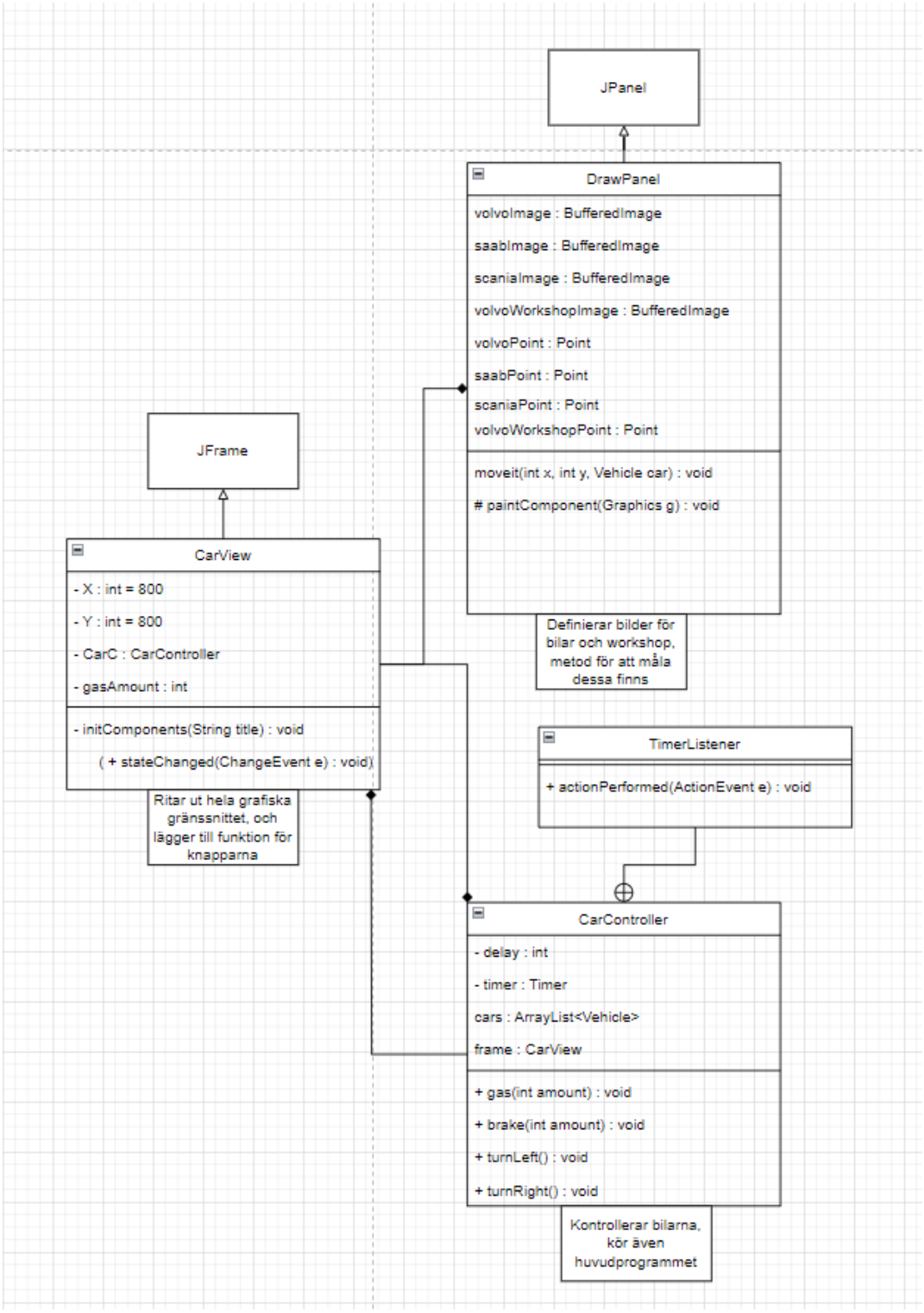
OOP Lab3

2025-02-19



Innehållsförteckning

Innehållsförteckning....1
Analys av aktuellt UML-diagram....2
Refaktoreringsplan....3
CarController Decoupling....3
DrawPanel....3
Carview....3
Gränssnitt....3
Kortfattad plan....4
Motiveringar....4
Uppdaterat Diagram....5

Analys av aktuellt UML-diagram

Huvudsakligen är diagrammet uppdelat i två tydliga arvshierarkier.
Den ena, och kronologiskt först uppbyggda, är den som har med fordonens konstruktion att göra. Där har gruppen dragit slutsatsen att hierarkin och strukturen i nuvarande läget är acceptabel; det finns inte mycket beroenden och denna struktur har mest med subclassing att göra.
Den andra har främst med simuleringen och gränssnittet att göra, då har gruppen bemärkt följande;

- CarView är beroende av CarController, detta krävs för att få knapparna att funka.
- Ansvarsområdena är relativt väldisponerade, men att CarController har main-metoden är något som bör ändras på.
    - Faktumet att main-metoden ligger i CarController har lett till att onödiga dependencies uppkommit.
- DrawPanel jobbar med hårdkodade värden, det gör det svårt att extenda programmet. En mer generell lösning erfordras.
- Dependencies mellan CarView och DrawPanel kan minskas.
- `initComponents` i CarView är väldigt lång, bör brytas ner till mindre metoder för eventuellt code reuse, samt för att göra koden tydligare och mer lättläst.
- Hårdkodade värden där det inte behövs bryter mot open-closed-principen.

Refaktoreringsplan

CarController Decoupling

1. Skapa en main-klass som tar över det ansvarsområdet `CarController` delvis hanterar, alltså försvinner `CarController > CarView` dependency.

DrawPanel

1. Passa in bilar *och* bilder som argument när objekten konstrueras, då slipper `DrawPanel` ha hårdkodade värden på onödiga ställen.
2. Points hanteras istället på individuell nivå med getters av x,y (doubles)

Carview

1. Bryt ner metoden `initComponents` till flera mindre metoder för att potentiellt få code reuse, samt för att göra koden tydligare och mer läsbar
2. Metoder som behövs:
    a. private Jpanel createPanelWithSpinner(int min, int max, int, step, int tiedVariable)
    b. Private Jpanel createControlPanel(int rows, int cols, Color color, int width, int height, ArrayList<JButton> buttons)
    c. Addlistener() ska lägga till funktion till knapparna

Gränssnitt

1. Formulera en klass som kombinerar aspekter av DrawPanel och CarView för att huvudsakligen skapa gränssnittet, med syfte att minska antalet dependencies mellan DrawPanel och CarView, i led med single responsibility principen.

Kortfattad plan

1. Flytta main-metoden från CarController till separat klass, relevanta instansvariabler bör också flyttas.
2. Refaktorera DrawPanel så att den tar bilder och bilar som argument i konstruktorn, därmed undviker vi hårdkodade värden. `getpos` för points.
3. Refaktorera `initComponents` i CarView till flera metoder för framtida code-reuse och tydligare kod
    a. En metod för att skapa en panel med en spinner
    b. En metod för att skapa en kontrollpanel
    c. En metod för att lägga till funktion till knapparna

Motiveringar

Att flytta ut main-klassen samt att skapa en ny gränssnittsklass gör att dessa klasser får singulära och individuella ansvarsområden, vilket följer SRP.
Att klasserna är mindre beroende av varandra leder till låg coupling och högre cohesion.
Att metoderna bryts ner till mindre bitar gör att vi får mer återanvändbar och tydligare kod.

Uppdaterat Diagram