Refaktoriseringsplan

Cars-Vehicle

Att införa en Vehicle-klass innebär att samma beteenden som delas mellan olika typer av fordon (bilar, lastbilar, etc.) nu är inkapslade i Vehicle-klassen. Detta främjar high cohesion eftersom varje klass ansvarar för sin specifika typ av fordon.

Workshop-Container

Skapa en generisk Container-klass som kan återanvändas i olika sammanhang. Klassen ‘Workshop’ använder den här containern för att hantera lastning och lossning av bilar, vilket främjar code reusability och separation of concerns.

VehicleFactory:

Att införa en Vehicle factory för att skapa fordon ger ett sätt att centralisera skapande logik. Om det till exempel är specifika konfigurationer eller steg involverade i att skapa olika typer av fordon, förenklar fabriken denna process.

Directions

Lägg Directions i en Enum för att hantera riktningar istället för att inkludera dem i Cars (Separation of Concerns)

Application (Superklass)

Representerar den övergripande applikationen eller systemet
Inkluderar huvudmetoden för att starta applikationen
Fungerar som ingångspunkt och knyter ihop de olika modulerna

CarController

Skapa en klass som representerar applikationens kärnlogik. CarController fungerar som en mellanhand mellan användarinmatningen och modellen, och anropar lämpliga metoder beroende på input. I den ursprungliga designen var CarController direkt kopplad till specifika fordonsklasser (Volvo, Saab, Scania). Styrenheten var tätt kopplad till detaljerna i individuella fordonsimplementeringar. Den hade dessutom för många ansvarsområden som hantering av användarinmatning, uppdatering av vyn och hantering av simuleringslogiken.

Modulär design

Moduler: Vehicle, Container, View, Controller, Application

Parallel Execution

Uppgifter relaterade till Vehicle-klassen, Workshop, Container, VehicleFactory, Directions, Application, CarController och Model-klassen involverar distinkta aspekter av systemet och kan därför arbetas parallellt av olika personer.