Svar på frågor

Uppgift 2: Model-View-Controller

Vilka avvikelser från MVC-idealet kan ni identifiera i det ursprungliga användargränssnittet? Vad borde ha gjorts smartare, dummare eller tunnare?

- smartare
    - Vi saknade en övergripande modell som hanterade logiken i koden. Innan hanterades logiken när t ex en bil krockade i väggen i CarController. En CarController ska vara dum och bör därmed inte göra någon beräkning. Vi resonerade att lägga all beräkning i Model som skulle vara smart.
- dummare
    - Vår CarView hade alla knappar i sig vilket gjorde den smartare än den behövde vara.
- tunnare
    - CarController körde spelet, skapade bilar och hanterade kollisioner. Detta var för mycket och den behövde tunnas ut och bara hantera user inputs.

Vilka av dessa brister åtgärdade ni med er nya design från del 3? Hur då? Vilka brister åtgärdade ni inte?

- Det vi gjorde var att skapa en application, en modell (eventhandler), en controller och en view.
- I vår applikation hade vi vår main-funktion och när vi körde den skapade vi instanser av eventhandler, controller och view.
- Vi åtgärdade inte riktigt problemet med att contollern var för tjock och att viewen var lite för smart.
- Vi tyckte dessutom att DrawPanel hade för mycket ansvarsområden. I enlighet med SIR bör en klass endast ha ett ansvarsområden och denna klass skulle innan både måla upp komponenterna och läsa in bilderna. Det vi gjorde var att skapa en ny klass ImageLoader som läser in bilderna som vi sedan anropar i Factory när nya vehicles ska skapas.

Rita ett nytt UML-diagram som beskriver en förbättrad design med avseende på MVC.

Uppgift 3: Fler designmönster

Observer, Factory Method, State, Composite. För vart och ett av dessa fyra designmönster, svara på följande frågor:

Finns det något ställe i er design där ni redan använder detta pattern, avsiktligt eller oavsiktligt? Vilka designproblem löste ni genom att använda det?

- Observer: Vi använder en observer som kollar på vår controller hela tiden och skickar signaler till viewen som repaintar spelet med jämna mellanrum.
- Factory: från labb 3 hade vi redan planer på att ha en Factory som skapar alla instanser av vehicles och workshop. Denna anropas i modellen.
- State: användes redan i applikationen. Till exempel hade varje vehicle två olika states - engineOn och engineOff. Om motorn på vehiclen inte var på så gick det inte att gasa i applikationen. Ett annat exempel är CarTransport - om rampen inte är nere så går det inte att gasa, CarTransport har därmed två tillstånd.
- Composite: vi har använt detta mönster rätt mycket i vår kod redan oavsiktligt. Till exempel i vår kod när vi kollar collisions så kan vi hantera alla typer av vehicles, även om det är en car eller truck. Detta mönster kan även ses för CarTransport som hanterar bilar som en grupp, även om det är olika typer.

Finns det något ställe där ni kan förbättra er design genom att använda detta design pattern? Vilka designproblem skulle ni lösa genom att använda det? Om inte, varför skulle er design inte förbättras av att använda det?

- För att förbättra abstraktion för CarTransport skulle vi på något sätt kunna implementera Movable och Loadable samtidigt - för en CarTransport är antingen movable eller loadable, aldrig båda sakerna samtidigt.

Uppgift 5: Utöka användargränssnittet

Implementera funktionalitet för att lägga till och ta bort bilar via GUI:t. Kan ni implementera detta utan att behöva förändra existerande klassfiler?

- Kan något designmönster vara relevant att använda för denna utökning?
  - Vi utökade den befintliga koden med samma mönster som tidigare kod var uppbyggt.