Uppgift 2:

Avvikelser från MVC I originalet:

1. I det ursprungliga användargränssnittet fanns ingen application klass, istället använde man controllern I detta syfte vilket inte är korrekt enligt MVC
2. Controllern användes här också I ett större syfte än att bara tunt koppla funktionalitet mellan användaren och model. Denna bör ha varit tunnare.
3. Modellen hade även olika beroenden med tanke på att controllern nu var smartare än den borde vara.
4. Det fanns ingen Model klass I den ursprungliga
5. Timern som bör ligga i modellen ligger istället i controllern
6. Det som den ursprungliga viewn gör är att skapa widgets som inte används för att ändra något i utseendet utan dessa widgets ändrar bara saker som finns i modellen. Detta gör att dessa widgets bör ligga i controller.

Brister vi åtgärdade:

1. Vi skapade en Application klass där main funktionen ligger.
2. Denna Application klassen skapar model, controller och view objekten.
3. Vi la över koden som skapar fönstret i en ApplicationWindow klass. CarView gjordes om till MotorVehicleWidgets. Den ligger tillsammans med MotorVehicleController i Controller paketet eftersom widgetsen påverkar modellen och inte viewn.
4. Vi la timern i Model klassen eftersom “världen” ska inte vara fast i tiden bara för att vi inte har en controller eller view.
5. Vi isolerade DrawPanels beroenden av de andra klasserna. Nu är den beroende av Application klassen som skapar den och så får den information från modellen via en observer.
6. Vi delade upp klasserna i packages.

Brister vi inte åtgärdade?

1. Vi kan inte se någon tydlig brist med MVC vi inte redan åtgärdade under laboration 3.

Uppgift 3:

Observer:

Redan använder: Vi använder en observer för att båda ge information och säga till DrawPanel att göra saker så att den inte har ett direkt beroende på DrawPanel.

Kan använda:

Vi kan inte hitta ett till ställe där vi kan använda en observer

Factory:

Redan använder: Vi använder ingen Factory I nuläget

Kan använda: Vi kan använda en factory för att skapa bilar. Detta hade underlättat när vi senare ska lägga till knappar för att skapa och ta bort bilar.

State:

Redan använder: Vi använder inte något State Pattern.

Kan använda: T.ex.

Vi kan använda State Pattern för att kolla villkor hos PlatformVehicle som ifall den rör sig eller om platformen är uppe eller nere. Vi kan också använda State Pattern för att kolla om fordon är Active (inte lastade) och därmed får röra sig.

Composite:

Redan använder: I drawPanels function så använder vi addVehiclesToArray(modelNames);

Kan använda: I flera utav våra widgets så använder vi oss av for-loops för att t.ex. starta alla bilar. Istället I dessa widgets hade vi kunnat använda composite pattern och kalla på en metod I model som inehåller loopen. Just nu dock så kan man säga att vi redan använder composite pattern ifrån användar perspektivet eftersom de trycker på en knapp och de påverkar alla bilar.