Vilka ansvarsområden har era klasser?

- Car har metoder som hanterar: gas, incrementspeed, decrementspeed, move, brake, startengine, stopengine, turnleft och turnright bland annat. Dessa ska alla bilar innehålla och är en generell modell av hur en bil är gjord. Subklasserna Scania, BilTransport, Volvo240 och Saab95 har egna specifika metoder som beskriver deras unika funktionaliteter, men alla "extendar" Car-klassen. Vi använder oss av Separation of Concern-principen genom att skapa flera bilklasser/subklasser (den abstrakta Car-klassen får inte alltför många metoder inom sig), eller genom att låta ***Scania*** och ***BilTransport*** skapa ett nytt ramp-Objekt med hjälp av komposition. Då skiljer vi ramp-metoderna från bilarna och vi kan återanvända Ramp-klassen i fler bilar om det önskas. Ramp-klassen är en egen klass så därför använder vi SRP här. I subklasserna Volvo240 och Saab95 *overridar* vi movemetoden och increment-decrement -speed, eftersom dessa är annorlunda jämfört med supertypen.

- BilTransport ska kunna frakta bilar, köra och släppa av dem på en verkstad. I klassen ska man ha koll på vilka bilar som är på flaket, så därför behövs en lista.

- Verkstaden ska kunna lägga till en bil och lasta av från verkstaden. Det finns olika verkstäder som kan ta olika bilar. Bilen läggs till i en lista som finns i verkstaden.

- DrawPanel har som uppgift att måla upp grafiken och skapa programmet i en ny flik. I denna klassen måste vi också ha koll på bilens/bilarnas positioner med hjälp av HashMaps. Det används därefter i ***CarController*** så att allting hanteras där.

- Carview hanterar knapparna. Dessa används och skapas därefter i ***CarController***.

- CarController kör programmet genom en timer som uppdatera programmet hela tiden och actionTimer som vad som ska ske varje tick. Applicera de funktionerna som behövs för att bilen ska funka, för alla bilar.

Vilka anledningar har de att förändras?

- Man skulle kunna göra en ytterligare klass som använder ***CarController***, ***DrawPanel*** och ***CarView*** i en egen klass som kör själva spelet. På detta sätt blir det mer tydligt. Klassen skulle exempelvis heta CarRun och där skulle man också kunna skapa en meny med olika funktionaliteter. På detta sätt blir också ***CarController***, ***CarView*** och ***DrawPanel*** inte beroende av varandra.
- Biltransport och verkstad har båda en lista av bilar. Man skulle kunna förbättra koden genom att göra en klass som har koll på listor eller som skapar en lista för en bil. Då skulle man kunna skapa fler typer av bilar som använder listor och det blir återanvändbart.
- Vi skulle främst behöva ändra klasserna: ***CarController***, ***DrawPanel*** och ***CarView*** för att få en mer dekompositionerad kod.
-

Varför är vår nya design en förbättring?

- Vi har använt oss av Separation of Concern genom att dela upp Drawpanel, CarController och Carview till mindre delar. Vi har gjort en factory-klass som har Single Responsibility Principle (SRP) till både biltyperna (**CarFactory**) och **GraphicsFactory** samt en bild-klass (**DrawImage**) som bara har hand om att rita ut

bilderna. Sen har vi gjort en start-klass som vi har dragit ur från CarController från början, som kör programmet. Även en BunAction som bara gör så att knapparna funkar.

Refakteringsplan

1, Skapa olika Factorys, istället för att carView etc ska vara beroende av andra klasser. så skapar vi objekt av de klasser carView är beroende av.

2, Dela upp DrawPanel till DrawPanel och DrawImage. DrawImage ska skapa alla bilder och addCar funktionen. DrawPanel har PaintComponent

3, Dela upp CarView till BunAction och CarView. BunAction ska ha actionListerner knappar. CarView ritar upp bilder. Carview behöver skapa ett drawpanel objekt.

4, Dela upp CarController. Till Start och CarController. Start sköter main och skapar en carContoller objekt för att få tillgång till listan av cars.
CarController sköter Logiken av knapparna och lägger till bilerna i listan av cars.

- **CarView**

  Vi har lagt alla ActionListener till en egen klass BunAction. BunAction ska ta in en carC:CarController. BunAction behöver ha en kompositionspil till Jbutton och GraphicsFactory.

- **CarController**

  Vi har lagt huvudfunktionen till klassen Start. start får tillgång till listan av cars från CarController.
  CarController sköter alla knappar med metoden. Logiken. Den har listan av bilarna. Den får bilar från CarFactory. CarFactory skapar en bil med deras position.

- **GraphicsFactory**

  skapar CarController, DrawPanel, CarView. istället för att de klasserna är beronde av varandra så kommer de vara beronde av GraphicsFactory.

- **DrawPanel**

  I DrawPanel skapar vi nu nya bilar, vilket från CarFactory får positionen för bilarna. Med positionen skapar vi nya bilder med hjälp av klassen DrawImage, där vi använder oss av hashmaps för att måla upp en bild. Från vår förra kod har vi i den nya versionen ersatt positionerna (Point), som vi skapar med hjälp av en annan hashmap, med en position som vi får från bilen.