Analys

Coupling och Cohesion:
Diagrammet har relativt hög coupling, extra problematiskt är när vi i “onödan” skickar datastrukturer (deque:n) mellan olika klasser. Cohesion är snarare en definitionsfråga då man kan argumentera för att all funktionalitet i t.ex. Car jobbar “åt samma håll” men också för att man kunnat dela upp funktionaliteten i mindre beståndsdelar.

Dependency Inversion Principle:
Diagrammet uppfyller Dependency Inversion Principle, ett exempel på detta är Loadable-interfacet som används för att koppla hög nivå (Loader) med låg nivå (CarTransport och Workshop).

Separation of Concern:
Det finns ett antal klasser som har ett tydligt avgränsat ansvarsområde bland annat: loader och objectwithposition. Objectwithposition definierar beteendet för alla objekt som har en position och loader sköter endast laddning av ett objekt i ett annat, beroende på distansen mellan dem (kan dock övervägas om distansen ska vara definierad vid instansiering av en loader). Själva simuleringsloopen borde förmodligen inte vara en del av Carcontroller, utan av en yttre klass som i varje tick nyttjar vissa av Carcontrollers metoder. Carcontroller bör rimligtvis endast innehålla funktionalitet för manipulation av modellen d.v.s bilarna i vårt fall. D.v.s. att till exempel timer och instansiering av objekt bör ske utanför CarController i någon form av Simulationsklass. På det sättet minskar beroendet mellan CarView och CarController då CarController inte behöver använda sig av Carviews metoder längre. Ett exempel är en collisioncheck som finns i Carcontroller som kallas på varje tick i Simulation och CarController kallar på en egen lämplig metod såsom turn.

Modellen borde ansvara för sina egna bilder, d.v.s. DrawPanel borde inte vara ansvarig för att tilldela bilder till olika objekt. Låt därför istället ObjectWithPosition bli ett interface som tvingar modulerna att bestämma en bildlig representation av sig själva.

Single responsibility principle:
Carcontroller ansvarar både för att hålla koll på världen/bilarnas status och att utföra förändringarna på dem och detta följer inte principen eftersom dessa metoder har olika anledning att förändra och de ansvarar för olika saker. Detta ansvar borde minskas eller bli ett mer tydligt ansvar.

DrawPanel ansvarar både för att skapa bilderna och att måla ut dem och de har båda samma anledning att förändras ur ett perspektiv men om en bil skulle ändra sitt utseende skulle förändringar på flera platser behöva ändras eftersom vi specificerar exempelvis hur många dörrar bilden har i bilklasserna. Drawpanel skulle kunna ha enbart ansvaret att måla ut de objekt den får in och på så vis tydligare ha ett single responsibility.

Refaktoriseringsplan

1. Skapa en ny klass Simulation som innehåller “världen” och alla objekt som befinner sig i den. Låt Simulation skapa instanser av både CarController och CarView och berätta för CarController vad världen innehåller (för tillfället). Simulation innehåller endast en metod main, som startar en timer och låter CarController hantera all logik för manipulationen av bilarna
2. Skapa nya metoder loadVehicles(ArrayList<Vehicle> vehicles) samt moveAllCars() i CarController för att låta CarController få information om spelvärlden respektive möjlighet att flytta alla bilar via deras movemetod. Ta även bort beroendet av Carview, då det inte längre är nödvändigt.
3. Ändra ObjectWithPosition till ett mer allmänt interface, DrawableWithPosition som också definerar metoder för att hämta bilden av objektet samt att välja en ny bild.
4. På grund av föregående, definera en bild alternativ en temporär bild för samtliga Vehicles som nu är DrawableWithPosition.
5. Ändra DrawPanels funktionalitet, så att dess enda funktion är att ta in objekt som kan bli målade (ArrayList<DrawableWithPosition>) via en loadObjects metod. DrawPanel kan nu måla rätt bild i rätt position.
6. Låt Loader ta hand om all funktionalitet för laddning, d.v.s. låt även Loader vara den enda modulen som innehåller Dequen, inte modulerna som använder sig av loader. Detta innebär också att loader måste innehålla en getter för load.