Uppgift 2: Beroenden

Beroendena i Lab1-delen är nödvändiga, det finns inte så många beroenden där, och de flesta kommer från en arvshierarki som är ganska given. Volvo & Saab är Vehicles, som gör mer än Vehicles, och alltid kommer vara Vehicles.
Arvs Hierarkin i Lab2 känns också bra, eftersom TruckBedTruck faktiskt gör allt en Vehicle gör, och lite till, samma med Scania (som bara är ett exempel på en konkret implementation av TruckBedTruck) och CarTransport.

Det känns som att fler saker är beroende på LoaderException, än som kanske behövs? Finns det något annat sätt att göra detta som minskar beroenden på LoaderExeption.
Att CarController och CarView båda beror på varandra är inte bra, och beroendet bör minskas till bara ett håll. Detta är en produkt av att dessa klasser gör för mycket och inte är separerade enligt ansvarsområdena.
DrawPanel borde inte vara beroende på Vehicle/Car, Den borde inte bry sig om vad det är den målat ut, utan bara att den får en lista av “utmålbara”-objekt.
CarController borde inte bero på de konkreta Saab95 och Scania, vilket den gör just nu för att starta turbo och höja flaket.

Lab2 beror i stor utsträckning på abstraktionen Vehicle, istället för Car eller Saab/Volvo, vilket är bra!
DrawPanel kanske hade åtminstone kunnat göras till en Usage dependency på Vehicle, istället för nuvarande komposition, eller tas bort helt.
De flesta beroenden är antingen arv, vilka inte riktigt med nuvarande upplägg går att ta bort. Även om det hade gått att göra detta på andra sätt, med komposition och andra lösningar för att minska arv-användandet, så har inte riktigt upplägget på labbarna varit det.

Vi bryter mot Single-Responsibility Principle flertalet gånger i koden, eftersom flera klasser har många ansvarsområden.
Vi bryter mot Open-Closed Principle i exempelvis DrawPanel, eftersom den är stängd för extension då DrawPanel endast kan användas till att rita upp det vi ger den i koden. Hade man öppnat den via exempelvis generics så hade detta öppnat upp dess användning mer.
Dependency Inversion Principle säger oss att vi inte borde direkt bero på exempelvis Vehicle, utan att det borde vara interfaces däremellan.
Cohesionen i vår modell är ganska hög, men det är också Couplingen eftersom flera moduler beror på flertalet andra. Ändringar i klasserna som beros på exempelvis Vehicle, kommer därmed ha påverkan på flera andra klasser.

Klassen Car borde kanske inte finnas. Den existerar för att både Saab95 och Volvo240 skall få Loadable interfacet, men är helt tom och utan mening förutom detta. Samtidigt är funktionaliteten för Vehicles och Cars inte speciellt avancerad och vi har inte några

bilspecifika (endast modellspecifika) funktioner som inte alla vehicles kan göra. Hade man utökat funktionaliteten borde man lägga de bitar som rör bilar i cars-klassen. Så hade man velat utöka detta någon gång kan det vara en god idé att ha kvar Cars-klassen.

Uppgift 3: Ansvarsområden

Labb 1:

- Vehicle
    - Håller data för fordon, sätter och ger dessa värden.
    - Flyttar på fordon, ändrar deras hastigheter. Startar/stoppar motorer.
    - Ändrar Direction på fordon.
    - Anledning att förändras:
        - Behov av förnyelse - ytterligare funktionalitet.
        - Förändrade kund- eller användarkrav - ändringar i uträkningar för hastighet osv.
- Position
    - Håller koll på en position med doubles istället för int som Points-klassen gör. Mer anpassad efter våra behov.
    - Kan räkna ut avståndet mellan sig själv och en annan Position.
    - Anledning att förändras:
        - Förändrade kund- eller användarekrav - Andra typer för värdena på x och y. Lägga till funktionalitet för positionerna.
- Saab95
    - Konkret implementation av Vehicle, lägger till funktioner för Turbo och egen hastighetsberäkning.
    - Anledning att förändras:
        - Förändrade kund- eller användarkrav - utökad funktionalitet eller förändrade fokusområden. Ex ska inte längre bara kunna köra utan också ha möjlighet att ändra inredning eller nått.
        - Ändringar i datan för bilen, ex enginePower
- Volvo240
    - Konkret Implementation av Vehicle. Har en egen hastighetsberäkning
    - Anledning att förändras:
        - Förändrade kund- eller användarkrav - utökad funktionalitet eller förändrade fokusområden. Ex ska inte längre bara kunna köra utan också ha möjlighet att ändra inredning eller nått.
        - Förändrade kund- eller användarkrav - ändringar i datan för bilen, ex enginePower
- Car
    - Säger genom att implementera interfacet Loadable, att alla subklasser är laddbara.
    - Tjänar inget annat syfte för tillfället.
    - Öppen för utveckling om fler bilspecifika funktioner önskas.
    - Anledning att förändras:

        - Förändrade kund- eller användarkrav - Utökad funktionalitet för saker som alla bilar har gemensamt.
- Movable (Interface)
    - Ger gränssnitt för funktioner rör-bara objekt skall ha.
    - Anledning att förändras:
        - Behov av förnyelse - Utökade eller ändrade metoder för att tillåta fler saker än vehicles, säg om exempelvis flygplan kanske behöver ändringar i interfacet för att kunna använda.

Labb 2:

- TruckBedTruck
    - Har en Truckbed och vikt
    - Säger till flaket att höja och sänka sig
    - Anledning att förändras:
        - Förändrade kund- eller användarkrav - ändra hur bilen samspelar med flaket. Kanske ska kunna gå att justera medan bilen körs.
- Truckbed
    - Höjer/sänker flaket
    - Ser till att flaket inte underskrider 0 grader och överskrider inte maxAngle
    - Anledning att förändras:
        - Förändrade kund- och användarkrav. Användaren vill höja/sänka på andra sätt, eller lägga till exempelvis låsning av flaket.
- Scania
    - Konkret implementation av en TruckBedTruck
    - Är Loadable
    - Anledning att förändras:
        - Förändrade kund- och användarkrav.
- CarTransport
    - Loadar/unloadar Loadable
    - Säger till de objekt som är loaded att uppdatera sin position när den själv flyttar på sig.
    - Anledning att förändras:
        - Förändrade kund- eller användarkrav - ändringar i datan för bilen, ex enginePower. Andra typer av load restriktioner samt hur nära en bil måste vara för att kunna loada den.
- LoadCarrier
    - Tillhandahåller last. Sparar objekt och kan lägga till och lasta av på olika sätt beroende på önskemål.
    - Kan även returnera information om sin last så som antal objekt och en lista på vilka på objekt som finns lastade.
    - Anledning att förändras:
        - Förändrade kund- eller användarkrav - lägga till funktionalitet för exempelvis lasta av allt samtidigt.
- LoaderException
    - Kastar en exception när en funktion hos LoadCarrier kallas som inte är möjlig. T.ex. man försöker lasta av ett fordon som inte finns.
    - Anledning att förändras:

        - ???
- Automotive Workshop
    - Lägger till eller checkar ut parametriska polymorfa <T>
    - <T> ska vara Loadable
    - Kollar så de får plats i workshop
    - Anledning att förändras:
        - Förändrade kund- eller användarkrav - användaren vill ladda annat än Vehicles, ex Kängurus. Lägg till ny funktionalitet.
- ILoader (Interface)
    - Ger gränssnitt för funktioner för saker som laddar Loadable-objects.
    - Anledning att förändras:
        - Förändrade kund- eller användarkrav - Lägg till funktionalitet till saker som kan laddas.
- Loadable (Interface)
    - Ger gränssnitt för vilka funktioner Loadable-objects skall ha
    - Anledning att förändras:
        - Förändrade kund- eller användarkrav - Lägg till funktionalitet till saker som kan bli laddade.

Labb 3:

- CarController
    - Sparar en lista med Vehicles.
    - Skapar dessa Vehicles.
    - Gasar, bromsar, startar motorn osv för dessa Vehicles
    - Sparar ett CarView objekt för att TimerListener skall använda detta.
    - Anledning att förändras:
        - Förändring av användarkrav - användaren vill ändra uppdateringsfrekvensen.
        - Behov av förnyelse - någon metod i Vehicle ändras som gör att denna koden måste anpassas.
        - Behov av förnyelse - bilarna får ytterligare funktionalitet, exempelvis Volvo som i en senare version kan flyga, då måste CarController ha en metod för att anropa denna metod för alla Volvos.
- TimerListener
    - Uppdaterar programmet och lyssnar efter events.
    - Uppdaterar positioner och säger till objekt att flytta på sig.
    - Säger också till DrawPanel att rita om sig.
    - Anledning att förändras:
        - Förändring av användarkrav - användaren vill inte längre avrunda värdena utan skriva ut dem som de är.
        - Behov av förnyelse - Vehicles är inte längre bara bilar, och därmed bör namn uppdateras.
- CarView
    - Lägger till UI:n (för- och bakgrund) från DrawPanel. Skickar med bilarna till DrawPanel

    - Konstruerar knapparna och ser till att något händer när användaren trycker på dem.
    - Säger till CarController att köra ex gas, brake osv.
    - Anledning att förändras:
        - Behov av förnyelse - uppdatera storleken på X och Y.
        - Förändring av användarkrav - lägga till fler knappar eller ändra text/placering/utseende på dem.
        - Behov av förnyelse - ändra utseendet av knapparna.
        - Behov av förnyelse - ändra värdet man kan gasa med, samma för flak eftersom de alltid är 10 så som koden är nu.
        - Behov av förnyelse - knappar för ytterligare funktionalitet som bilar i CarController kan tänkas ha.
- DrawPanel
    - Skapar UI:n och ritar bakgrunden som bilarna rör sig på.
    - Kopplar bilder till objekt.
    - Kopplar punkter till objekt.
    - Flyttar på sina punkter
    - Ritar ut bilderna som representerar fordonen som skapas
    - Anledning att förändras:
        - Förändringar av hårdvara - användaren vill använda andra filformat, sökvägar eller namn för bilderna.
        - Förändring av användarkrav - användaren vill använda andra färger eller storlekar på UI:n.
        - Förändring av användarkrav - användaren vill flytta bilarna närmare varandra eller motsvarande.
        - Behov av förnyelse - UI:n är för gammaldags och behöver nytt utseende.
        - Förändringar av hårdvara - programmet skall köras på exempelvis en mobiltelefon, och behöver ändra storlek på fönster eller annat.

På vilka klasser skulle ni behöva tillämpa dekomposition för att bättre följa SoC och SRP?

De klasser som i nuläget borde prioriteras att dekompositioneras är:
CarController - Den gör alldeles för mycket i nuläget. Behöver delas upp i en del för datarepresentation, en del för actions.
TimerListener - den gör för mycket, hade varit bra om den delegerade sina actions till andra klasser. Så exempelvis kollar inte den om en bil är i närheten av kanterna, och flyttar på dem.
CarView - Dela upp i en del som ritar/konstruerar knapparna, och en annan del som använder/lyssnar på dem.
DrawPanel - dela upp ansvarsområdet på nya metoder och/eller klasser så att inte lika mycket görs direkt i konstruktorn för DrawPanel-objekt.

De klasser som hade kunnat dekomposioneras är:

Vehicle - Dela upp så att den har en Engine. Då kolla på subklasser så de har all funktionalitet fortfarande. Detta hade betytt enklare och mer extensibility senare, eftersom exempelvis en ändring i hur motorn fungerar inte hade behövt påverka alla Vehicles.

Uppgift 4: Ny design

Varför Bättre?

Vi har signifikant minskad Couplingen mellan modulerna för datan (Vehicle mm) och klienten (grafiken från labb 3) genom att ta bort och göra om beroenden. Detta leder oss mot HCLC (High Cohesion Low Coupling). Vi har även tagit bort ett par beroenden, vilket är bra eftersom vi vill sträva efter en så låg mängd som möjligt.

Vi har fördelat ansvaret mer över flera klasser, vilket följer både SRP och SOC, vilket har gett oss enklare möjligheter att förstå vad delar av koden gör, men också att modifiera delar utan att påverka/förstöra andra.

Vi har följt DIP genom att se till att DrawPanel beror på abstraktionen Movable istället för den konkreta implementationen Vehicle.

Vi har följt (delar av) designmönstret Model-View-Controller vilket gett tydligare uppdelning av koden (SRP + SOC) samt enklare modularitet.

Vi har tagit bort beroendet från TruckBedTruck på en konkret TruckBed implementation, och lagt det på den abstrakta AbstractBed istället som implementerar interfacet ITruckBed. Detta följer DIP genom att vi nu är beroende av en mer abstract implementation av en TruckBed. Detta liknar det man åstadkommer med en Bridge-pattern, vilket är det vi försökt eftersträva.

Vi har följt SOC på CarView eftersom vi flyttat ansvaret med att skapa knapparna till en separat klass. Med en ButtonFactory har vi även skapat ett interface mellan vår modell och de konkreta knapparna, vilket minskat vår Coupling.

Refaktoriseringsplan

- Skriv om DrawPanel så att den inte beror på Vehicle genom att istället använda Movable-interfacet. Passa även på att abstrahera metoderna bättre så att de är uppdelade i att skapa fönstret, respektive måla objekten.
- Bryt ut metoder och attribut gällande Datamodellen från CarController till en separat model-klass. Lägg alla metoder för att styra bilar i denna.
- Ändra anrop i TimerListener till att kalla på CarModel istället för bilarna direkt.
- Låt CarView och CarModel kommunicera (Model skickar listan med bilar till View) istället för View och CarController.
- Bryt isär CarView till delar som hanterar knapparna, och delar som ritar upp planen.
- Implementera en Factory som skapar knapparna till CarView.
- Ersätt TruckBed med en abstract implementation och tillämpa Bridge-Pattern för att skapa de konkreta implementationerna ScaniaBed och CarTransportBed.
- Byt namn på metoder och attribut för att representera deras funktion bättre.

- Abstrahera delar av metoder för att göra koden mer förståelig och öka möjligheten för återanvändning.

Hur man hade kunnat dela upp refaktoriseringen?

Att ändra namn på metoder och attribut går att göra helt självständigt om man gör det på ett smart sätt och använder sig av inbyggda refaktoriserings-verktyg i sin utvecklingsmiljö.

TruckBed abstraktionen kan göras helt separat från resterande ändringar till projektet, eftersom detta inte påverkar funktionaliteten som används just nu, utan bara gör den mer lämpad för framtida utveckling och förändring.

Refaktoriseringen av DrawPanel, dvs att göra den beroende på Movable istället för Vehicle, är ganska lätt arbete, och påverkar inte heller resterande del av projektet så mycket. Ändringarna till grafik-modulen är ganska sammanknutna, och bör nog göras av en person/mindre arbetslag för att inte orsaka förvirring. Det hade gått att dela upp denna refaktorisering mer, om så hade behövts, se exempel nedan. Dock är det inte jättemycket ändringar, så det kanske är bättre att behålla de inblandade utvecklarna till en låg nivå.

Skapandet av CarModel går att göra separat, men när detta skall implementeras in i CarFrame (tidigare CarController) kommer det att medföra förändringar på metodanrop i CarFrame.

På samma sätt som med CarModel så kan man göra utökningen av CarView enskilt först, och sedan när den är klar koppla samman den med resterande av modulen.