Högskolan i Gävle

Miniprojekt 4

State och Decorator

Hanna Medén, Niklas Nordgren

2020-02-16

Miniprojekt 4 i kursen Objektorienterad design och programmering II, DVG503

Innehållsförteckning

1 Inledning

I det här projektet modifierades gammal kod som övertogs av kursansvarige för att implementera designmönstret State. Koden som ärvdes innehöll designmönstret Decorator, som i detta fall implementerades med hjälp av en 2D cirkel som under körning kan ändra utseende genom att använda en grafisk dekoration som fyller cirkeln med svart färg. Målet var att utöka koden så att det fanns fler geometriska figurer att rita ut, samt fler dekorationer för att ändra utseendet av figurerna.

2 Metod

Inledningsvis implementerades utökningar av det befintliga designmönstret Decorator. Från början fanns klassen ShapeDecorator som implementerar interface:et Shape, detta interface implementeras även av de geometriska 2D figurerna, som ursprungligen endast var klassen Circle. Detta betyder att klassen ShapeDecorator och Circle har den gemensamma typen Shape. Vidare tar ShapeDecorator in ett objekt av typen Shape som argument i konstruktorn, detta för att utföra sina egna extra operationer, som i just detta fall är att fylla den grafiska representationen av ett Shape objekt med en svart oval, för att sedan delegera vidare till det insvepta objektet som togs emot vid initiering i konstruktorn.

Klassen Rectangle adderades, likt klassen Circle implementerar den interface:et Shape för att möjliggöra dekorationer under körning. Inte helt oväntat så representerar objekt typen Rectangle grafiska 2D rektanglar, som vidare kan dekoreras tack vare Decorator designmönstret. Två klasser som används för att dekorera 2D figurer skapades, ShapeDecoratorCrosshair och ShapeDecoratorDino, där den förstnämnda dekorerar en grafisk 2D figur med ett hårkors och den sistnämnda med en bild av en dinosaurie.

Den gamla koden använde sig av olika Mode:s för att veta vad användaren valt att göra med sin ”canvas”, men detta gjordes till State. All funktionalitet låg tidigare i ShapeContainer, men detta utbyttes mot en rad kod och koden för varje funktion flyttades ut till olika State-klasser som alla har ärver från den abstrakta klassen State. Klassen State består metoderna pointerDown, pointerMoved, getState och setState. Alla State klasser gjordes till Singleton, så de innehåller även en getInstance metod. Det är metoder som nås och används av alla de specificerade State klasserna för vad de ska utföra. Varje val i menyn har en egen State-klass, som tar in en Point, en ShapeContainer och pointerMoved metoden tar även in en boolean för att se om muspekaren är nedtryckt. De innehåller sedan olika funktionalitet för att utföra det användaren valt.

3 Resultat

Nedan i **Fel! Hittar inte referenskälla.** visas programmet under exekvering där sex geometriska 2D figurer ritats ut och dekorerats med ett Decorator-objekt vardera. Vidare i **Fel! Hittar inte referenskälla.** visas den grafiska representationen av ett objekt av typen Rectangle som dekorerats med tre olika dekorations objekt.

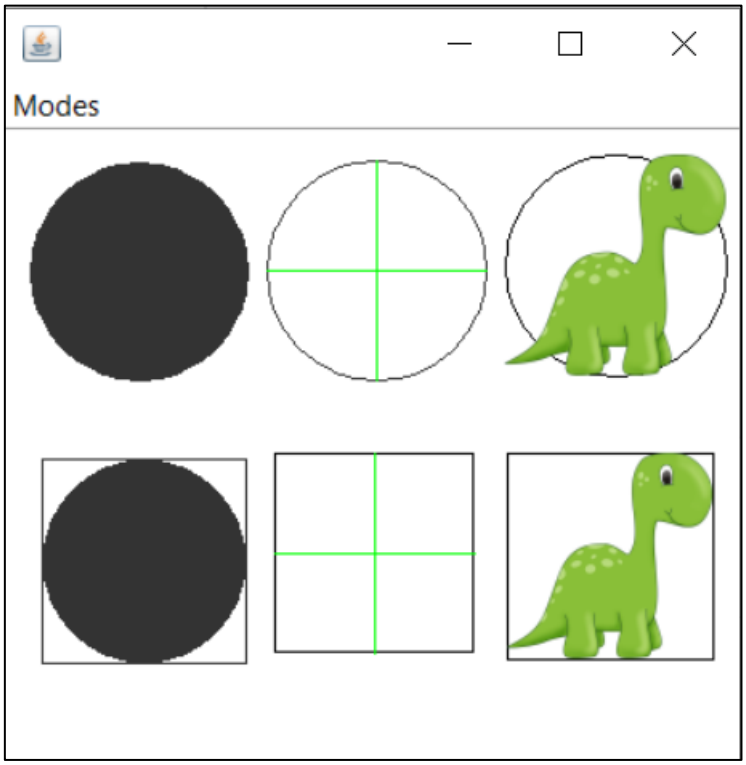

Figure 1: Programmet vid exekvering. 2D figurer en decoration vardera.

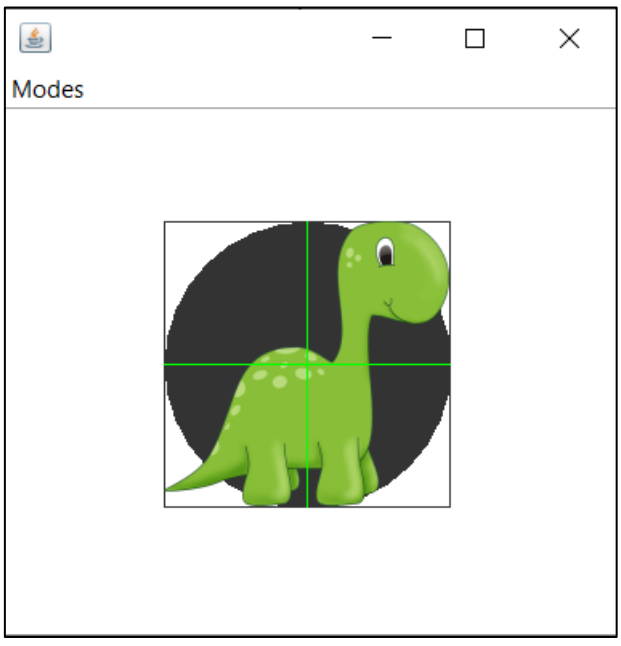

Figure 2: Programmet vid exekvering. Rektangel med staplade dekorationer.

I menyn får man välja mellan nio olika lägen, där det finns två olika geometriska 2D figurer som kan ritas ut, och tre olika sätt att fylla dem på. Det är även möjligt att välja att avmarkera fyllningarna, ändra storlek, flytta och ta bort figurer.

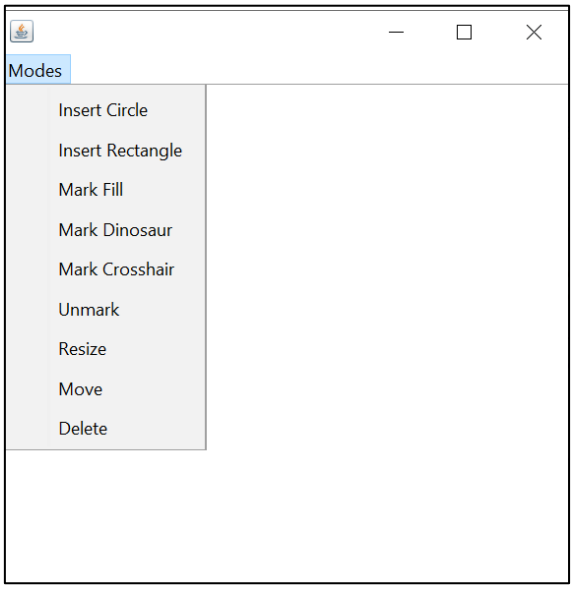

Figure 3: Valmöjligheter i menyn

4 Diskussion

Efter genomfört projekt har vi fått förståelse för att det kan finnas stora fördelar med att använda designmönstrena Decorator och State då de ger applikationer större flexibilitet samt underlättar underhållsarbetet vid utökning/ändring av koden.

När vi implementerade designmönstret State, valde vi att göra varje enskild funktion från menyn till en egen State, för att slippa if-else-satser eller State i ett State och då det kändes lättast när det inte var så många funktioner av samma ”typ”. Vi diskuterade om att först bara ha ett InsertState och MarkState för att ta in en ”nyckel” av typen String för att avgöra vilken 2D figurer som skulle ritas ut eller vilken Decoratorklass som skulle användas vid dekorering.

5 Bilaga – UML för systemet

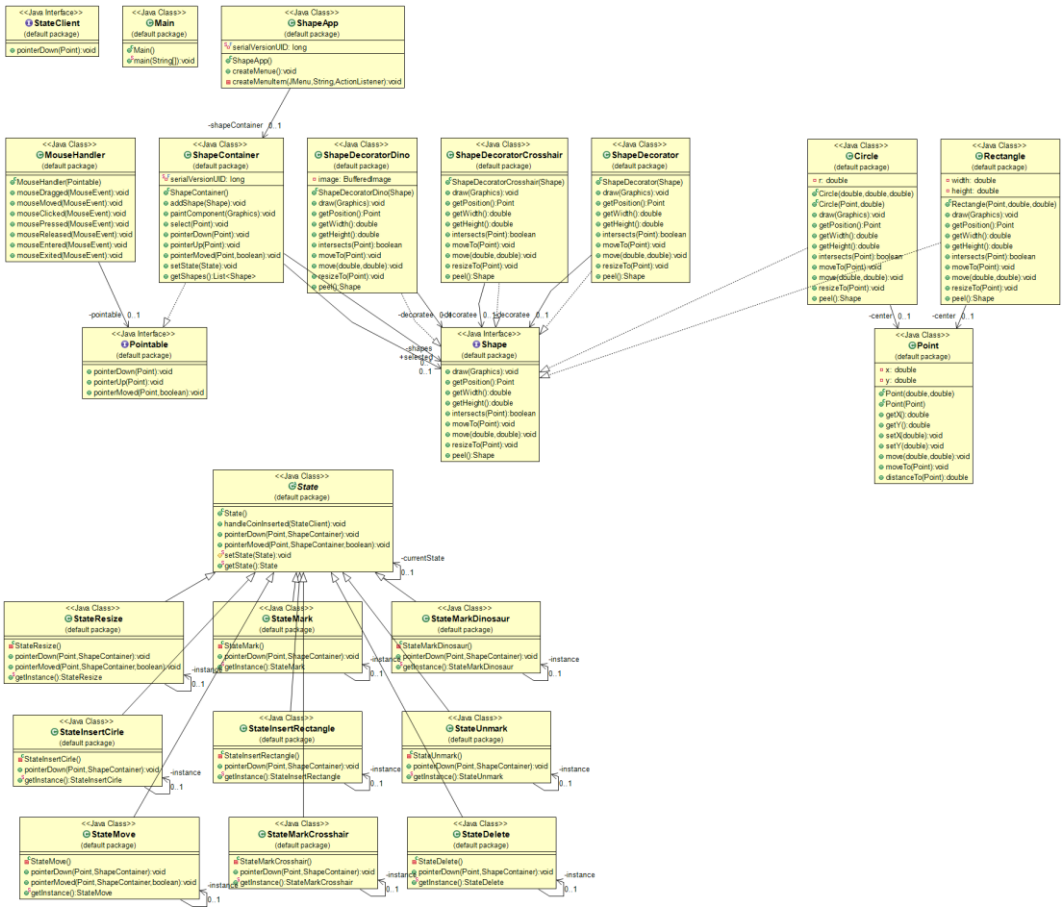

Figure 4: UML av systemet.