Peer Review av grupp JGMORs projekt

Erik Anttila Ryderup Erik Bengtsson Axel Hertzberg Jonas Nordin

Team JEEA
9 oktober 2020

Designens efterlevnad av SOLID-principer.

En viktig anmärkning till projektet, är att det inte finns någon information om hur man kör programmet. Detta då skulle behövas eftersom programmet kan köras från 5 olika ställen.

Single responsibility principle - Många paket följer inte riktigt principen då de har flera ansvarsområden, exempelvis har vissa paket hand om både modell och vy. Klasser följer däremot principen bra, till skillnad mot enstaka metoder som ibland har fler ansvarsområden.

Open closed principle - Projektet är öppet för vidareutveckling på grund av dess separation av beroenden mellan de olika spelen. Däremot är projektet även öppet för modifikation eftersom många variabler och metoder har för stor synlighet (public).

Liskov substitution principle - Subklasserna ändrar inte beteende i nuläget eftersom de inte har så mycket funktion i sig. I nuläget bryter inte programmet mot Liskov.

Interface segregation principle - Projektet följer principen. Dock utnyttjar de inte gränssnitt i java. De fallen de använder implementationsarv är det rimligt.

Dependency inversion principle - Efterlevs inte alltid.

Använder programmet en konsekvent kodstil?

Kodstilen skiljer sig mellan de olika paketen. Det finns inte många likheter mellan komponenterna i projektet. Exempel är att man använder olika slags for-loopar, ibland använder man lambda-uttryck medan i andra fall inte. Det är också intressant att projektet använder sig av både Swing och Javafx. Det verkar onödigt att göra detta och lättare att hålla sig till ett grafiskt bibliotek.

Är koden återanvändningsbar?

Alla program är sina egna och har inbördes en tät koppling. Klassen Rectangle är återanvändningsbar men det är en ganska enkel klass.

Är den enkel att underhålla?

Eftersom alla komponenter är separerade från varandra är de lätta att ändra, varje spel är ju ett helt eget program. En ändring i exempelvis Pong kommer inte ha några effekter på Frogger då det är helt olika program. Ett undantag från regeln är den konkreta klassen Rectangle, vilken både Frogger, Space Invaders och Snake beror på. Om implementationen ändras kan det helt förstöra de andra programmen.

Eftersom ingen klass deklareras som final skulle det kunna bli problem om någon i framtiden bestämmer sig för att använda implementationsarv.

Kan man enkelt lägga till eller ta bort funktionalitet?

Eftersom programmet just nu är fem olika program/spel samlade i ett projekt utan beroenden mellan varandra skulle det vara lätt att lägga till fler spel med en egen kontroll och vy. Skulle man däremot lägga till eller ta bort funktionalitet inom något av programmen blir detta svårare då man skulle behöva fler abstraktioner inom varje program.

Används designmönster?

Utöver att projektet i stora drag följer en MVC arkitektur hittades inga andra designmönster.

Är koden dokumenterad?

Koden i projektet är inte alls dokumenterad i denna version. Javadocs saknas vilket gör att det är svårt att få en överblick av vad metoder och klasser gör. Det blir också svårt att se vilka andra klasser som används. En annan svårighet är att förstå hur det är tänkt att programmet ska fungera.

I projektet har ”TODO” använts en gång. På flera olika ställen skulle projektet kunna utnyttja ”TODO” istället för att kommentera ut metoder.

Används lämpliga namn?

Klasser och metoder är namngivna på ett bra sätt. De använder sig utav ”CamelCase” överallt och namnen på klasser är beskrivande. Det finns en klass ”ProjectTemplate” som har ett otydligt namn då namnet inte säger något om klassens ansvarsområde.

I kontrollerna har man namngivit knappar till ”btn...” vilket anses vara en onödig förkortning då man lika gärna kan skriva ”button” för en tydligare namngivning. Annars är det väldigt bra namngivning i programmet.

Är designen modulär? Finns det onödiga beroenden?

Modellen är modulär. Till viss del på grund av att projektet egentligen består av 5 separata program i samma projekt. Ser man till varje program specifikt är de flesta programmen modulära då de följer en MVC arkitektur utan cirkulära- eller andra onödiga beroenden. Det finns däremot spel som inte är direkt modulära då grafik och controllers implementeras direkt i modellen, exempel är i koden till Snake- och Breakout-spelet.

Använder koden lämpliga abstraktioner?

I den version som har analyserats så är spelen i stort sett helt separerade så därför finns det ingen abstraktion spelen emellan. Det verkar som att klassen ”Rectangle” är skapad som en abstraktion för en rektangel som är tänkt att användas i mer än ett spel vilket är bra. Det skulle vara bra att se denna bakom ett interface istället om tanken är att använda en rektangel på mer än ett ställe. I spelet Space Invader använder projektet en abstrakt klass ”Movable object” som håller gemensamma saker för ”spaceInvader”. Detta kan finnas i ett interface istället, tex i ”IMovable” som redan finns i programmet. Programmet har väldigt många olika paket som kan utnyttjas bättre, ett exempel är att utnyttja javas ”package-private” istället för att ha alla klasser som public. Vilket skulle ökat inkapslingen i programmet. För att öka mängden abstraktion i programmet borde gemensam funktionalitet faktoriseras ut till ett gränssnitt så att det blir möjligt att använda polymorfism. Projektet skapar många objekt i olika delar av programmet. Detta skulle man kunna abstrahera genom att använda ett designmönster tex. ”Factory Method Pattern”. I ”snake” är ”GUI” en publik statisk variabel. Det är oklart varför den ska kunna ändras överallt i programmet. Det skulle vara en bättre lösning att använda ”Singleton Pattern” om det är nödvändigt att den kan manipuleras överallt. Eller om tanken är att det ska vara möjligt att ändra instansen som ”GUI” pekar på, erbjuda setters och getters.

Är koden vältestad?

Koden använder väldigt få tester. Testerna är inte placerade efter vilken klass de testar utan alla test ligger i samma java-fil. Projektet testar om två rektanglar skär varandra. Rörelse testas på spelen frogger och Space Invader. Här skulle det vara bra med fler tester.

Finns det säkerhetsproblem, finns det prestandaproblem?

Programmet använder inte internet och verkar inte skriva till datorn heller. Så det finns inga uppenbara säkerhetsproblem. Prestandaproblem hittades inte men värt att notera är att programmet inte utför särskilt mycket.

Är koden enkel att förstå? Har den en MVC-struktur och är modellen isolerad från andra delar?

Programmet är indelat i flera olika mindre program som alla har sin egen modell, vy och controllern. Just nu är följer alla delar i stora drag MVC-strukturen förutom vissa undantag. Ett exempel finns i ”SnakeController” där det verkar som att det är tänkt att en kontroll ska hålla både tid och spelarens poäng. I enighet med MVC borde detta ansvar istället ligga på modellen. Det finns även andra ställen i projektet där MVC bryts. Ett ställe är i ”Breakout”-modellen där vyer initialiseras, samma sker i ”snake”-modellen.

Kan designen förbättras? Finns det bättre lösningar?

En fundering är ifall det är en bra idé att strukturera modellen, vyn och kontrollen så det är gjort nu. I dagsläget ligger alla spelens modeller, vyer och kontroller i samma paket. Då alla spel har en egen main-klass funderar vi på ifall alla spel inte hade kunnat ligga i separata paket med en egen modell, vy och kontroll istället för att dela med andra. Vi tror att det hade gjort koden mer lättöverskådlig.

Det skulle var fördelaktigt ifall projektet utnyttjade inkapsling. Alla klasser bör inte vara publika. Samt det som nämndes i abstraktionsstycket.

RAD och SDD

RAD-dokumentet har gruppen kommit långt på och det märks att gruppen har lagt tid på det. Vi tycker att gruppen använder ett bra språk genomgående i dokumentet. Dock bör ni fundera på om det verkligen är lämpligt att använda både svenska och engelska i dokumentet då både 1.1 och 1.2 är skrivna på svenska medan resten av dokumentet är på engelska. Innehållet är bra och speciellt är domän-modellerna både snyggt ritade och klassernas ansvarsområden tydliga. Dock kan det vara en bra idé att dela upp bilderna för att det ska bli lättare för läsaren för att se vad. Under 2.3 bör det förklaras vad det är som visas eller eventuellt lägga domänmodellen som en bilaga.

SDD-dokumentet är svårt att ge feedback på då det fortfarande befinner sig i ett för tidigt stadie.