Testdriven utveckling i Java

Inlämningsuppgift 3

Design patterns

Design patterns är fördefinierade, industri-standardiserade lösningar på återkommande problem. Det finns 23 olika patterns, däribland singleton, facade, abstract factory m.fl. Dessa delas in i tre olika kategorier, creational (skapande), structural (struktur) och behavioral (beteende). Dessa används när man har flera problem som kan lösas på samma sätt, eller genom samma mönster. Designmönster är dock inte en metod man klipper och klistrar, som man gör med hjälpklasser och bibliotek, utan snarare instruktioner hur man strukturerar sin kod för att lösa problem, likt en ritning. Här måste man också ta i beräkning vad för typ av problem man har så man kan tillämpa rätt typ av designmönster för lösningen. Väl använda designmönster kan också underlätta kommunikation mellan utvecklare, genom att man kan helt enkelt förklarar vilket designmönster man använt för sin produkt. Detta gör det därför lätt för en ny medlem i teamet att, med hjälp av kommentarer och mönster, förstå hur programmet är uppbyggt. [1]

Creational patterns

Det finns flera olika creational patterns, men bland de mer kända är factory, abstract factory, builder, singleton och prototype. Creational pattern hanterar objektskapande på lämpligt sätt, för att undvika att koden blir onödigt komplex eller får designproblem längre fram under utvecklingsprocessen. Man kan också dela in creational patterns i klass-skapande och objektskapande där klass-skapande instansierar en klass, medan objektskapande gör ett objekt från en vald klass.

[1] "Refactoring Guru - Design Patterns." https://refactoring.guru/design-patterns. Öppnades 3 mars. 2021.

Structural patterns

De här mönstren hjälper till vid strukturering av programmet, hur alla små delar ska bilda det stora hela men utan att minska flexibiliteten. Fokuset ligger på hur en klass ska ärva från andra klasser och hur de kan vara komponerade av flera klasser. Ett exempel på strukturellt mönster är bridge, som kan dela upp en stor hierarki till flera mindre för att i sin tur förhindra hierarkin från att bli enorm i takt med att man lägger till objekt. Bron det här mönstret är döpt efter är den mellan hierarkierna, t.ex. carBrand contains carColor. Bridge är en av sju structural patterns enligt Gang of Four, författarna av boken *Design patterns*. [2]

Behavioral patterns

Beteendemönster hanterar hur klasser och objekt ska kommunicera med varandra samt eventuella algoritmer. Enligt GoF finns det elva behavioral patterns, däribland iterator och observer. Iterator används för att iterera igenom en collection, likt en for-loop. Skillnaden är att en iterator kan vara mer flexibel eftersom den kan anpassas till hur man vill se sin collection. En for-loop hårdkodas oftast vilket minskar dess flexibilitet. Observer fungerar som så att man bara returnerar data till observers som finns sparade i en array eller liknande, t.ex. prenumerationer.

Antimönster

Användningen av designmönster minskar avsevärt risken för att antimönster ska förekomma i systemet. Antimönster är lösningar på problem som fungerar, men löser problemen på ett sådant sätt att de blir väldigt svåra att felsöka och testa, m.m. Några antimönster är spaghettikod och blobprogrammering. Den förstnämnda är väldigt känslig för svårupptäckta buggar, medan den sistnämnda är i väldigt svår att testa, eftersom det är ett enda stort objekt som sköter all funktionalitet. Med hjälp av designmönster blir det istället lättare att göra allt det här eftersom man kan se vilken del av programmet som får buggar och liknande.

Jonatan Ljung

[2] "Gang of Four Design Patterns - Spring ...."
https://springframework.guru/gang-of-four-design-patterns/. Öppnades 5 mars. 2021.