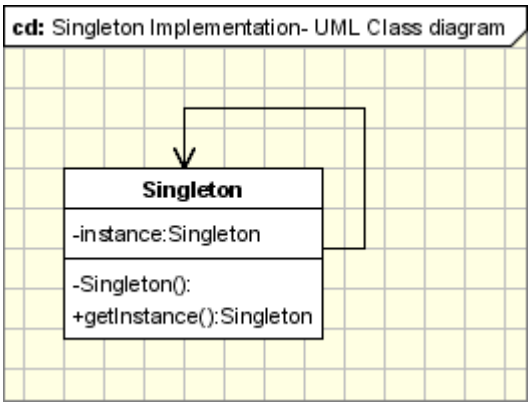
Inlämningsuppgift 3 – Design patterns

När jag föddes, 1977, gav arkitekten Christopher Alexander ut boken ”A pattern language” tillsammans med ett par andra författare. I denna bok menar de att de som använder byggnader själva vet bäst vilka byggnader de behöver. Erich Gamma, Richard Helm, Ralph Johnson och John Vlissides (Gang of Four == GoF) gillade idén och förde över denna på mjukvaruutveckling med boken ”Design Patterns, Elements of Reusable Object-oriented Software”. Ett par kapitel i denna bok handlar om objektorienterad programmering, men det som gjort den enormt populär är de sista kapitlen som beskriver 23 designmönster inom mjukvaruutveckling.

Ett designmönster är en beprövad lösning på ett återkommande problem. Det beskrivs ofta och bäst med UML och är helt plattformsoberoende. Ett designmönster ger en inte en färdig programmeringslösning, snarare ett ”programmeringstänk”. Dess fyra delar är:

1. Namn
2. Beskrivning
3. Lösning
4. Konsekvenser

Det kanske lättaste exemplet är Singleton. Om man av någon anledning endast vill skapa ett enda objekt av en klass, och inte fler, hur ska man göra då? Designmönstret Singleton är lösningen. Singletonklassen skapar ett nytt objekt endast om ett objekt inte redan finns. ”Användaren” kan vara helt ovetande om detta, och det är det som är tanken. Klassen i sig ska veta hur och vad som ska göras. Klasser / användare utanför behöver inte veta vad som försiggår inuti klassen. På https://www.oodesign.com/singleton-pattern.html visas detta genom UML (Unified Modeling Language):



Designmönster delas upp i tre grupper: creational, structural och behavioral patterns. Med detta menas hur klasser och objekt skapas, dess struktur och hur de interagerar. Singleton är således en creational eftersom den har med skapande att göra. Andra vanliga creationals är Abstract factory, Builder, Factory method och Prototype.

Structurals fungerar ofta som adaptrar. När två interfaces inte kan kommunicera med varandra behövs en adapter. Bridge skiljer abstraktion från implementation, låter klasser fungera med andra klasser utan att ändra på dess kod. Facade ger samma gränssnitt åt alla bakomliggande klasser.

Behavioral patterns består av t.ex. Observer, Iterator, Template method, Command. Observer är en helt underbar grej. Finns troligen hos varenda nyhetsbyrå eller liknande. När en nyhet kommer ut så ser Observer till att notifiera alla som behöver det. Mailtjänst, Twittertjänst, Facebooktjänst, webtjänst, SMStjänst osv. Alla får samma info samtidigt. Ett one-to-many mellan objekten.

Gemensamt för alla mönster är att de hjälper utvecklare till en bättre slutprodukt. Här finns massor med lösningar på problem. Hjulet behöver inte uppfinnas igen. Dessutom, kanske utan att man märker det, åtminstone som en junior utvecklare, finns en stor framtidstanke i koden. Det ska relativt enkelt gå att bygga ut koden utan att behöva börja från scratch. Eller förändra! Jämför med verkligheten. Skatter och avgifter ändras. Trender / moden ändras, fyrkantiga eller runda knappar? Engelska eller svenska? Det ska byggas så att sådana saker är enkla att ändra på i koden. (Nu kanske jag blandar jag lite designprinciper med designmönster här.) Slutsatsen är att som utvecklare kan det i vissa fall vara lite jobb att implementera (rätt) designmönster, men har man lyckats med det, då har man en beprövad lösning som även kommer att hålla i framtiden.

Källförteckning:

https://sv.wikipedia.org/wiki/Christopher_Alexander

https://sv.wikipedia.org/wiki/Designm%C3%B6nster

https://en.wikipedia.org/wiki/Design_Patterns

https://www.oodesign.com/

Resten kommer från ”huvvet” vad jag lärt mig från föreläsningarna samt tidigare ”slöläst” på nätet.