Objektorienterad Analys & Design:Individuell Uppgift Joakim Gröhn

Jag har valt att jobba med följade patterns, från creational patterns **factory** och **singleton**, **adapter**, **composite** och **facade** på från structurals och slutligen valde jag **command** från behavioral. Dessa valdes för att de var de som tilltalade mig mest och som i förväg kunde föreställa mig hur de skulle passa ihop.

Användandet av **singleton** är det enklaste valet att förklara, då programmet har en och endast en databas så finns det helt enkelt bara bruk för ett interface till den. Det blir också klarare programkod då man inte behöver skicka vidare den klassen som sköter interaktionen med databasen upp och ner i trädet av klasser. Klassen **DbAdapter** tjänar även som en **adapter** mellan SQLite databasen och det övriga programmet.

Dessutom används command-patternet för att kommunicera mellan övriga programmet och adapterna. Det finns inte mindre än **6 command-klasser i DbCommandskatalogen**. Poängen med dessa är att organisera att hålla alla SQL queries för sig och även göra det enkelt att reversera ändringar i databasen, detta är dock inte implementerat pga brist på tid. Men det var tanken med att använda just commandpatternet här.

Den största komplexiteten som abstraheras bort är den gällande de olika menyerna. **TeacherMenu** är den klass som agersa som **facade** för en rad menyklasser, men även **ScheduleFactory** som likt namnet skvallrar om, är en factory som producerar de olika schemamenyerna.

Valet av fasad ser jag i skrivandes stund som ännu mer självklart än singleton, på sätt och vis så får man alltid något sådant när man programmerar då man har ett antal olika resurser som man måste anropa på ett eller annat vis, gör man en webbsida så är användargränssnittet som en stor fasad.

Angående **ScheduleFactory** vill jag säga att valet kanske inte är lika självklart, kanske hade det varit bättre att fokusera på de olika klassernas enskilda konstruktörer. Men då de alla på något sätt är någon form av kommandotolksmeny så finns det rätt mycket gemensamt för dem.

Det sista design patteret jag har använt är **composite**, när man ska uppdatera en schemaläggd lärare så väljer man i menyn att visa alla schemalagda lärare för en dag, sen så väljare man ut en att hantera och kan ge den en ny arbetstid, man måste alltid mata in både start och sluttid även om man bara vill ändra det ena. Jag nämner composite här för att om man väljer att visa hela veckoschemat, så kan man precis som innan välja en lärare för att ändra dess schemaläggning, men här så gör man en ändring över hela arbetsveckan. Klassen **WorkWeek** låter en hantera varje **Workday** i ett svep. Kanske lite onödigt då SQL är ganska bra till att göra alla uppdateringar som behövs i detta fallet i en enda query, men det finns fördelar också med att abstrahera bort det.