Reflektion – Left To Do – Johannes Jakobsson

Program.cs - I den här klassen skapas en ny instans av klassen Application som i sin tur anropar metoden RunStartMenu från klassen Application som kör igång all funktionalitet.

Application.cs - innehåller startmenyn, instruktioner samt felhantering som användaren behöver för att denne ska mata in rätt uppgifter. Det är den här filen som i slutändan anropar all funktionalitet från de andra klasserna och knyter ihop säcken så att användaren ska få rätt upplevelse.

Task.cs - är en abstract superklass som innehåller två variabler, en string där det sparas en beskrivning av uppgiften som användaren matar in. Samt en bool som håller koll på om uppgiften är färdig eller inte. Klassen innehåller även metoder som sätter värden till variablerna och hämtar värden från dessa. En metod som returnerar en string är virtuell som de underliggande subklasserna ska kunna "overridea".

PlainTask.cs, Deadline.cs, Checklist.cs och ChecklistSubTask.cs är alla subklasser till Task.cs och ärver dennes egenskaper. Alla dessa subklasser har även olika egenskaper som är unika för just den klassen. Exempelvis Deadline.cs har utöver superklassen Tasks egenskaper även en deadline variabel. Denna variabel tillsätts ett värde(slutdatum) som användaren matar in och sen räknar programmet ut via en metod hur många dagar det är kvar tills uppgiften ska vara färdig. I slutändan returneras en sträng via en override metod. Som "kör över" superklassens virtuella metod så att användaren får ta del av rätt uppgifter utifrån vilken "att-göra-uppgift" de vill använda sig av. Alla subklasser fungerar på liknande sätt men med sina unika egenskaper för att just den metoden ska returnera korrekt information.

Storage.cs - I den här klassen lagras listorna todoList, archivedList samt checklistList. Alla metoder för att fylla på listor och arkivera uppgifter finns i denna klass. Metoder för att exempelvis hämta listors värden och tillsätta värden i dessa är känt för den här klassen. För att Application.cs ska ha tillgång till metoderna i denna klass så skapas en ny instans av Storage.cs i just Application.cs. Därav kan man komma åt all funktionalitet som behövs i klassen Application.

Använda principer

Arv: Uppgiftens huvudprincip är att man ska skapa nya "uppgifter/tasks/att-göra" i olika former. Därför är det bra att göra en superklass som innehåller de delar som vidare är gemensamt för alla uppgifter som ska skapas. Därför skapar jag en egen abstract superklass för detta som jag namnger Task. Alla uppgifter/klasser som ärver av Task och har denne som superklass kommer därav att ärva dess egenskaper och tillåter oss att återanvända den koden. Genom att sätta variabler i superklassen till protected kan subklasserna även ta del av dessa variabler. Jag skapar även en virtuell metod som varje subklass som ärver från superklassen kommer att behöva "köra över"(override) för att returnera en string i önskat format.

Enkapsulation: Alla variabler i klasserna är satta till private, förutom i superklassen där dessa är protected. För att komma åt variabler i de olika klasserna behöver man skapa en ny instans av klassen och sedan via den instansens kalla på metoderna för att antingen "Setta(set)" eller "Getta(get)" variablerna, eller på ren svenska för att kunna tillsätta ett värde eller hämta ett värde på/från variablerna. Variabler satta till protected istället för private i superklassen gör så att även subklasser kan komma åt de variabler som har den "protection-leveln". Mitt program är bra enkapsulerat enligt mig för att inte obehöriga klasser ska komma åt onödig information.

Polymorfism: Jag har i så god mån som möjligt, på de ställen och områden jag kan, försökt att bygga mitt program så polymorfismiskt som möjligt. Det hör ju även ihop med arvet av superklassen. I superklassen finns exempelvis en virtuell metod som heter GetTaskString, alla subklasser har också denna metod som overridar superklassens metod, det betyder att alla uppgifter är "ganska" likadana men kan se olika ut i slutändan. Jag har även försökt baka ihop vissa metoder så man följer konceptet DRY för att samma metod ska kunna göra fler saker. Exempelvis metoden GetAllTasksFromStorageLists var i början två olika metoder som hade samma funktionalitet för olika listor i början, men nu är det ihopsatt till en metod som kan hämta olika strängar beroende på vilken lista man vill komma åt. Dock så återupprepas kod I vissa delar av mitt program fortfarande, som förmodligen kan göras bättre i framtiden när kunskapen är djupare. Samtidigt så är mitt program polymorfiskt byggt då jag har avstått så mycket jag kan från att bygga det med att involvera "Console-kommandon" i metoderna av uppgifts-klasserna och storage klassen. Så att man på ett så enkelt sätt som möjligt skulle kunna byta UI från konsolprojekt som det är nu till ett annat.

Tester för klasser

CheckIfArchivedTaskIsCorrectTask: Lägger till två nya uppgifter i todoList. Sätter den andra av uppgifterna till "färdig" och arkiverar alla färdiga uppgifter i todoList. Sedan kallas en metod som hämtar värdet på den första platsen i den arkiverade listan och dennes sträng för att sedan jämföra med det förväntade värdet.

Är relevant för att se att det verkligen är rätt uppgift som arkiveras eller inte. Det hade varit förödande om programmet arkiverar uppgifter som inte är markerade som klara. Fungerar detta test vet man att rätt uppgift blir arkiverad.

FillToDoListWithCheckListTaskAndCheckIfCorrect: Testar att lägga till en Checklist uppgift i en todolista och kontrollerar sedan att det är rätt string som finns i todlistan.

Är relevant för att Checklistan är det mest invecklade "bygget" i hela programmet och det kan vara bra att kontrollera att man via rätt metoder kan komma åt den här platsen och se att rätt sak hamnar på rätt plats.

SetSubTaskAsDone: Lägger till en checklista i todoList, lägger även till en ChecklistSubtask i checklisList och kontrollerar att det går att sätta subtasken som done.

Det här är relevant för att man behöver kunna göra det här för att senare kunna kontrollera att alla delmål/subtasks är markerade som färdiga så att man i slutändan kan markera hela checklistan som färdig i todoList.

CheckIfAllSubtasksInListIsDone: Lägger till en checklista med två subtasks. Sätter sedan båda som färdiga och kontrollerar om alla subtasks är färdiga.

Är relevant för att utan den här funktionen kan man inte kontrollera om alla subtasks är färdiga. Då kan man inte heller sätta hela Checklistan som färdig i todoListan och senare arkivera uppgiften.

Kontrollerar även i AddDifferentTasksToList att även en Deadline uppgift går att lägga till i listan. Det är bra att veta att alla nya klasser går att lägga till.

Egna kommentarer

Har några få egna kommentarer som jag ville få med. Själva checklistan kämpade jag något extremt med. Det tog många timmar att enbart komma fram till en lösning så att programmet fungerar tillsammans med Checklistan. Jag är medveten om att metoden PrintAllTasksAndSetToDone i klassen

Application är aningen rörig med alla try-catch och if-satser. Eftersom jag fastnade så länge på checklistan och den delen så ansåg jag resterande tid fick gå åt de viktigaste delarna, att få igång funktionaliteten och att programmet i slutändan fungerar. Det här med två listor som tar hand om allt som har med checklistan att göra, en i Checklist som heter subTaskList och en i Storage som heter ChecklistList blir aningen rörigt anser jag. Det hade man också kunnat få till lite smidigare på ett eller annat sätt. Men det är samma här, i brist på kunskap och tid så fick jag prioritera funktionalitet före “fin kod”. En grej till som jag är lite missnöjd med angående checklistorna är att det går ju att få till så att checklistor där alla delmål har blivit kryssade automatiskt ska bli ikryssade, men det var lite klurigare för mig att reda ut under tidspress och med den uppbyggnaden jag har fått till. Jag har helt enkelt mycket jag kan förbättra. Men samtidigt är jag nöjd med det jag presenterar och hoppas att man kan få konstruktiv kritik i en handledning här framöver!

Repo: https://github.com/campus-varnamo/left-to-do-jajo21