Utvecklarblogg Seminarium 3

Den här dokumentationen är för utvecklarblogg till seminarium 3 i TDP007, meningen med att skriva detta dokument är för att redovisa hur vi har tänkt/tolkat och löst uppgifterna.

- **UPPGIFT 1**

Vi tolkade uppgiften så att vi måste ha alla nödvändiga funktioner för att beräkna försäkringspremien för vem som helst genom att skapa funktionen "**car_model()**" och testa den. Vi trodde först att "**policy.rb**" d.v.s reglerna ska finnas för att använda men vi märkte sedan att den inte finns och däremot så behövde vi skapa den själva, vilket ledde till olika tankar på hur man ska skriva det faktiskt. Därefter gjorde vi samma sak för de andra funktioner som vi tänkte att ska va med.

I funktionen **evaluate_policy()** evaluerar vi innehållet i filen **"policy.rb"** genom att använda **instance_eval()**.

I funktionerna som **car_model()**, **post_number()** och **sex()** har vi loopat i varje Array som finns i **"policy.rb"** filen för att hitta den specifika regeln. Detta gjorde det möjligt att lägga till index 1 i **result** data medlemmen.

Till skillnad från de andra funktioner så har vi i funktionerna **license_year()** och **age()** använde vi Rubys inbyggda metoden **between?** som kollar om värdet ligger i mellan första och andra indexet i Arrayen eller inte.

I "**policy.rb**" valde vi att skriva alla reglerna i ***Arrayer*** för att komma åt dem genom deras index. Till exempel i "**car_model**", "**post_number**" och **"sex"** har vi skrivit i första indexet i citattecken och den andra index var det som vilket poäng den personen ska ha. Däremot i "**license_year**" och "**age**" så har vi första två index som indikerar tidsskalan och tredje indexet är det själva poängen som personen kommer få. Därefter skapade vi de speciella reglerna och då funkade dem, det tog lite tid på att hitta ett sätt att lösa den del av uppgiften. Vi använde oss av ***eval*** som kan köra det som finns inuti en citattecken som kod, då kunde vi skriva reglerna i **policy.rb** vilket var mycket användbart i detta fall.

Vi tycker att koden har blivit för långt då flera funktioner gör samma sak, det enda skillnaden var vilket objekt den ska behandla dvs (age, sex, etc.). Med detta menar vi att "**method_missing**" skulle användas i koden för att slippa funktions upprepning samt ge mer frihet till försäkringsbolaget att göra ändringar i "**policy.rb**". Men tyvärr så lyckades vi inte med den trots att vi gjorde många försök.
Vi testade varje funktion i taget där kunde vi se om de fungerar som de ska eller inte. Sedan kom vi fram till den slutliga testfallet som matchar med det som står på själva uppgiften.

Vi hade svårt att komma igång med att skriva koden enligt beskrivningen eftersom DSL är ett helt nytt ämne för oss. Så vi tänkte läsa på ytterligare om ämnet kanske kommer vi komma på något. I stort sett så hade vi svårt att tolka/förstå uppgiften trots att vi satte oss flera timmar på den.

- **UPPGIFT 2**

I denna uppgift tolkade vi så att vi måste skriva ett eget språk för enkla logiska uttryck med hjälp av befintliga parsern **“rdparse.rb”**.

Vi började med att läsa ***DiceRoller***-klassen och försöka förstå innehållet och hur den är implementerat. Med hjälp av ***DiceRoller***-klassen fick vi skapa klassen ***LogicLang*** som sedan plockade vi tokens via ***lexer*** och använde vi befintliga parsern för att parsa den.
Vi fick skriva reglerna enligt angiven grammatiska specifikationen i uppgift beskrivningen.
Sedan skapade vi en Hashen **our_var** för att senare ska kunna spara variabler och dessa värde i den.

Funktionen **done()** tar in en sträng som några specifika ord vilka är skriven i koden.
Funktionen **activate()** gör det möjligt till användaren att skriva på terminalen. Om användaren skriver något ord som finns i den strängen som finns i **done()** funktionen då ska terminalen skriva ut “**Bye.**” för användaren och stänger ner själva programmet. Annars kommer den funktionen parsa den strängen som användaren matar in.

Vi hade svårt att skriva enhetstester i början men efter hjälp från assistenten så kom vi fram till att vi behöver skriva funktionen **for_test()** som tar in en sträng från själva testerna och returnerar den strängen efter parsning.

Ett misstag vi gjorde under tiden vi skrev koden var att använda ***Array*** istället för ***Hash***.
Detta passar bättre med ***Hash*** eftersom man kan spara både “variabeln” och “värdet” i dem.

En av de sakerna som var relativt svårt att ta sig in i var BNF-grammatiken i sin helhet då det var ganska stort ämne och förmodligen kommer användas senare i TDP019 kursen också.