LINKÖPINGS UNIVERSITET

TDP007: Konstruktion av datorspråk

Utvecklarblogg
Seminarieuppgift 3

Författare

Viktor Rösler, vikro653@student.liu.se
Jim Teräväinen, jimte145@student.liu.se

INSTITUTIONEN FÖR DATAVETENSKAP

Vårterminen 2021
Version 1.0

8 februari 2021

1 Uppgift 1

1.1 Tolkning

Uppgiften var att skapa ett `DSL` för att kunna skriva tabeller och regler som sedan används för att räkna ut försäkringspremier. Vi tolkade det som att själva `policy.rb` filen skulle vara helt utbytbar och all logik ligger i Ruby koden.

1.2 Tillvägagångsätt

1.2.1 Person klassen

Vi valde att göra två generella metoder. En `table` metod som hanterar alla tabeller i vårt DSL, och en `rule` metod som hanterar alla regler. Vårt mål var att minimera hur mycket som behöver ändras i `Person` klassen om `policy.rb` utökas med nya tabeller och regler. Med vår lösning behöver man endast lägga till en ny medlemsvariabel i `Person` för varje ny tabell i `policy.rb`, och man kan lägga till nya regler i `policy.rb` utan att ändra i `Person`.

`table` metoden tar två argument. Det första är namnet på tabellen i form av en sträng eller en symbol. Det andra argumentet är en Hash som innehåller datan i tabellen. Namnen på tabellen måste matcha en medlemsvariabel i Person klassen.

`rule` metoden tar tre argument. Det första argumentet är en Array med villkor, det andra argumentet är en operator som används för att ändra säkerhetspoängen, och det tredje argumentet är värdet som ska ändra säkerhetspoängen. Varje villkor i det första argumentet ska vara en sträng uppdelad i tre delar. Den första delen är namnet på en medlemsvariabel i `Person`, den andra delen är en jämförelseoperator, och den sista delen är ett värde.

1.2.2 DSL

Varje rad i vårt DSL börjar med något av metodnamnen `table` eller `rule`, följt av argument till metoden separerade med komman.

1.3 Lärdomar

Den mest intressanta delen av uppgiften var att försöka tänka ut de mest generella sättet att skriva funktioner så att de fungerade med flera tabeller av olika slag. Från början planerade vi att ha 1 funktion för varje tabell men kom snabbt på att det blev koduppprepning och inte alls så elegant.

Vi har fått en större förståelse för `eval` funktionerna och då specifikt `instance_eval` som vi använder för att läsa in och tolka `policy` filen.

2 Uppgift 2

2.1 Tolkning

Uppgiften var att använda en `parser` som vi blev givna och med denna hantera indata så att användaren kan göra enkla logiska beräkningar och spara resultatet av detta eller booleska värden i variabler.

2.2 Tillvägagångsätt

Vi implementerade språket genom att byta ut namn och den grammatiska specifikationen i `DiceRoller` klassen i `rdparse.rb`. Vi ändrade även hur tokens är definierade, och skapade en Hash att spara variabler i. Det som blev lurigt var att ändra värdet på en variabel som redan hade blivit tilldelad ett värde. Vi ville att

variablerna skulle bete sig som i Ruby så att värdet på variabeln skrevs ut om den matades in ensamstående. Samtidigt ville vi inte heller att programmet skulle krascha om variabeln inte fanns när användaren matade in namnet. Denna funktion var lätt att fixa med en Hash och logik i `:var`. Problemet uppstod när vi sedan ville tilldela ett värde till samma variabel med hjälp av `:assign` då den nyttjar `:var` och istället för att returnera namnet på variabeln gav oss värdet på variabeln.

För att lösa problemet så var vi tvungna att flytta logiken för att returnera värdet på en variabel till `:term`. I efterhand kändes detta helt självklart men det var svårt att förstå i stunden.

2.3 Lärdomar

Ingen av oss hade arbetat med varken `tokens` eller `parsning` innan så allt var väldigt nytt. Att läsa in rätt tokens var inte svårt då vi redan är vana vid reguljära uttryck men att förstå i vilken ordning parsern arbetade var klurigare. Igenom att titta på exemplet med tärningsprogrammet och testa lite själva så kom vi närmare att förstå hur allting fungerar.