Lab 4

Deadline: 2013-03-18 19:59

Lab 4 utförs i par om två stycken elever.

Inlämning sker med en zippad fil ”Lab4.zip” mailad till david.g@jetas.se.

Zip-filen skall innehålla samtliga .rb-filer, samt samtliga FitNesse-testsidor ni skapat för denna lab. Utöver detta skall zip-filen även innehålla en textfil ”info.txt”.

- Info.txt innehållande: Fullständigt Namn + Födelsenr. för båda eleverna.
- Katalogen från FitNesse som innehåller de FitNesse-sidor ni skapat för G-uppgifterna

  (Denna katalog hittas i er FitNesse-katalog, i underkatalogen FitNesseRoot. Om er sida ligger under FrontPage, dvs. adressen till sidan är t.ex: FrontPage.Lab4Main, så måste ni gå in i FrontPage-katalogen under FitNesseRoot för att hitta katalogen med era test-sidor. Katalogen kommer ha samma namn som er startsida i FitNesse).
- Projekt-katalog för G-delarna (innehållandes MyString.rb + fixture-fil)
- Fixtures-katalogen för VG-delarna
- FitNesse-katalogen för era test för VG-delarna

  (Om ni har samma startsida i FitNesse för både G-uppgifterna och VG-uppgifterna så behöver ni inte dela upp katalogen ni lämnat in i en G-del och en VG-del. Det räcker att ni lämnar in en katalog som innehåller test för både G- och VG-uppgifter).

Tänk på att inte flytta runt de kataloger som hör till fungerande test. Ta istället kopior på de kataloger som skall lämnas in och lägg dessa kopior i en ny katalog som ni döper till t.ex. ”Lab4” – denna katalog med kopiorna kan ni sedan göra en zip av och lämna in.

Anmärkning

Jag räknar med att denna lab är relativt svår att komma igång med (jag gissar att det kommer gå snabbare när ni väl fått till de första fungerande testerna dock).

Med tanke på starten vi fick på tisdagen så kan jag se att vi behöver senare-lägga deadline för denna lab. Jag stämmer av läget kring detta på fredag.

För att komma igång: kika på mitt FitNesse-demo som finns i den nya Virtual Machine jag lagt upp. Försök med hjälp av videon jag länkade till i föreläsnings-slajden och Guiden som hittas under dokument/filer på kursbloggen hänga med i hur detta demo är upplagt och hur man skulle kunna modifiera det för att börja testa MyString i G-uppgiften.

Maila vid frågetecken.

Uppgifter för G: Max 20p – 10p för G

Syfte: Prova på utveckling att test-tabeller med hjälp av FitNesse.

Mål: Implementera klassen MyString - en egen strängklass. Denna klass skall ha 9 metoder (listas nedan). Du skall sedan skapa testtabeller i FitNesse som testar dessa 9 metoder.

Vi hade normalt inte utvecklat en egen strängklass i ett verkligt projekt. Vår implementation kommer med all sannolikhet inte bli bättre än implementationen av String som redan finns i ruby.

Anledningen att vi gör det i detta projekt är att jag vill att ni främst skall fokusera på FitNesse, och inte på själva ruby-imlementationerna.

Att metoderna i MyString baseras på metoder i den riktiga string-klassen innebär at vi, för t.ex. ”slice”-metoden som skall implementeras för MyString, kan använda oss av metoden ”slice” som redan finns implementerat för strängar i ruby på följande sätt:

```
module Lab4
        class MyString
                def initialize(string)
                        @internal_string = string
                end
                def slice(x,y)
                        @internal_string.slice(x,y)
                end
                #…Implementation av fler metoder här…
        end
end
```

Detta innebär att vi snabbt och relativt smärtfritt kan implementera en egen klass som vi skall testa.

Uppgift:

- Implementera en MyString-klass i ruby. Denna klass skall innehålla 9 metoder: ”new”, ”concatenate”, ”capitalize”, ”to_lower”, ”slice”, ”count”, ”reverse”, “include?” och

"prepend". (Max 2p) (Om du redan implementerat gsub I den tidigare uppgiften behöver du inte implementera reverse-metoden)

- Implementera en fixture-klass som gör att vi kan testa ovanstående implementation I FitNesse.

- Skapa en Test-sida i FitNesse som innehåller 1 tabell per metod i MyString-klassen (Totalt 9 tabeller/metoder). Varje tabell bör innehålla minst 3 rader utöver tabellrubrik och kolumnrubriker. Tabellerna skall vara körbara test (Max 2p per tabell)

Ovanstående tre punkter utgör hela G-uppgiften. Nedan följer ytterligare beskrivningar och förklaringar.

I de fall det finns gränsvärden eller avvikande input för någon metod sker poängsättning enligt följande:

- Endast 3 rader i testtabellen, som alla testar normal input: 1p – 1.5p
- Endast 3 rader i testtabellen, varav 1 rad testar normal input, 1 rad testar värde innanför gränsen, 1 rad testar värde utanför gränsen: 2p

Jag gissar att ni talat om gränsvärden innan, men för att förtydliga. Om vi skulle sätta begränsningen "max 16 tecken" på ett input-fält av typen sträng så skulle det göra att strängar av längden 16 (innanför gränsvärdet) och strängar av längden 17 (utanför gränsvärdet) är extra intressanta att testa.

Vi kan testa att ett visst värde ger ett error/exception med ex. RSpec: "expect {object.some_method(0)}.to raise_error" – där "some_method" är en metod på objektet/variabeln "object". I detta exempel förväntar vi oss ett error om vi ger metoden argumentet "0".

Listning + Exempel för de 9 metoder som skall finnas i MyString:

Efter varje metod nedan, beskriver jag beteendet för varje metod med några exempel som ser ut som följande:

- str: "string" - str.concatenate("test") => "stringtest"

Ovanstående exempel innebär:

- str: "string" =>

  str är ett objekt av typen MyString som är initierat med "string"

- str.concatenate("test") => "stringtest"

  om concatenate-metoden anropas med ett argument "test" (av typen String) på ovanstående MyString-objekt så skall metoden returnera "stringtest" (av typen String)

Hett tips: på http://www.ruby-doc.org/core-2.0/String.html kan ni hitta fler exempel och beskrivningar för samtliga dessa metoder, utöver det som listas nedan.

Exempel för de 9 metoder ni skall implementera

new(str)

- str: "string" - MyString.new(str) => ...ny MyString-instance skapad

concatenate(string_arg)

- str: "string" - str.concatenate("test") => "stringtest"
- str: "string" - str.concatenate("") => "string"
- str: "" - str.concatenate("test") => "test"
- str: "string" - str.concatenate(33) => ”string!”

capitalize

- str: "string" - str.capitalize => "String"
- str: "String" - str.capitalize => "String"
- str: "STRING" - str.capitalize => "String"

to_lower

- str: "string" - str.to_lower => "string"
- str: "String" - str.to_lower => "string"
- str: "STRING" - str.to_lower => "string"

slice

- str: "string" - str.slice(1,3) => "tri" (0-indexering gör att (1,3) innebär tecken 2-4 i strängen)
- str: "string" - str.slice(2,5) => "ring"
- str: "string" - str.slice(1,6) => “tring”

count

- str: "string" - str.count => 6
- str: "23" - str.count => 2
- str: "" - str.count => 0

reverse

- str: "hello" – str.reverse => “olleh”
- str: "you" – str.reverse => “uoY”
- str: "first,second" – str.reverse => “dnoces,tsrif”

include?(string)

- str: "hello" - str.include?("lo") => true
- str: "hello" - str.include?("asdf") => false
- str: "hello" - str.include?("elo") => false

prepend(other_string)

- str: "World!" - str.prepend("Hello") => "HelloWorld!"
- str: "World!" - str.prepend("Hello ") => "Hello World!"
- str: "World!" - str.prepend("") => "World!"

Överkurs för er som vill ha en utmaning/lära er mer ruby:

Implementera egna versioner av de metoder som skall implementeras för MyString (dvs. använd er inte av metoderna som redan finns för String-klassen).

En bra strategi för detta är att ni börjar med att utveckla metoderna med hjälp av de String-metoder som inbyggda i ruby. Gör sedan VG-uppgifterna och återvänd till G-uppgifterna efter du är klar med VG-uppgifterna. Försök nu att ersätta metoderna med egna implementationer

Du kommer inte få några extra poäng för detta, men du kommer troligtvis lära dig en hel del ruby.

Tips: Se och behandla strängarna som arrayer av characters om du ger dig på detta, så kommer det gå enklare.

Uppgifter för VG: Max 26p – 11p för VG + minst 10p på G-delen.

Syfte: Utveckla FitNesse-tester för ett mer verklighetsnära program, Läsa befintlig källkod.

Att lämna in:

- fixtures-katalog

  (denna skall heta ”fixtures” och vara tänkt att ligga i projektets ”lib”-katalog.

- katalog med FitNesse-sidor som innehåller körbara testtabeller.

  FitNesse-sidorna skall ha en Suite-sida som har 4 andra Suite-sidor under sig. De 4 Sub-Suite-sidorna skall vara ” ArticleManagerList”, ” ArticleManagerImport”, ” ArticleManagerDetails”, ”ArticleRepository”. Under respektive Sub-Suite skall det finnas ett antal test-tabeller enligt beskrivningen nedan.

- Totalt ingår det i ovanstående punkt att skapa 11 FitNesse-tabeller (Se nedan)

Notera: Om vi lagt till möjligheten att spara Articles i en databas och lagt till ett grafiskt webbinterface istället för det nuvarande text-interfacet, så hade detta program varit väldigt snarlikt t.ex. ett lagerhanteringsprogram. Att lägga till en databas och ett grafiskt gränssnitt för detta projekt är en relativt enkel sak att göra.

Note: Vi kommer testa samma saker som redan finns test för i Cucumber-features och RSpec-examples. Detta är ingenting vi normalt gör.

> (Vi väljer det ena eller andra– det finns dock untantag. Det är generellt sett smidigare att beskriva Scenarios i Cucumber och smidigare att sätta upp beslutstabeller i FitNesse – detta kan göra att vi i vissa fall väljer att använda båda verktygen för att testa olika aspekter av en applikation).

Anledningen till att det redan finns cucumber-test i detta projekt är framförallt för att jag var tvungen att specificera programmets beteende på något sätt för att ni skall kunna testa det - som en bonus får ni se lite fler cucumber/rspec-tester.

…btw, vad vi kan utläsa från ovanstående är att ni kommer ha stor hjälp av implementationerna av Cucumber- + RSpec-testerna då ni utvecklar era fixture-filer och era testtabeller.

Uppgift:

Utveckla tester i FitNesse för programmet ”ArticleManager”. Källkoden för ArticleManager hittas här: http://www.loudelou.se/BDDArticleManager-master.zip (lägg märke till att det finns en README.md-fil som beskriver programmet i zip-filen. Eventuellt kommer du också få en varning vid nedladdning p.g.a. att ingen annan laddat ned filen tidigare.).

Du skall skriva testtabeller fixture-klasser som gör att vi kan testa följande i FitNesse:

2 tabeller som testar beteendet som beskrivs I 2 av de 4 scenarios som hittas i articlemanager_list.feature-filen (vilka 2 scenarios du väljer är valfritt).

- Dessa tabeller skall ligga i en Sub Suite i FitNesse: ArticleManagerList

3 tabeller som testar beteendet som beskrivs I 3 av de 9 scenarios som hittas i articlemanager_import.feature-filen. (vilka 3 scenarios du väljer är valfritt).

- Dessa tabeller skall ligga i en Sub Suite i FitNesse: ArticleManagerImport

3 tabeller som testar beteendet som beskrivs I 3 av de 7 scenarios som hittas i articlemanager_details.feature-filen. (vilka 3 scenarios du väljer är valfritt).

- Dessa tabeller skall ligga i en Sub Suite i FitNesse: ArticleManagerDetails

3 tabeller som testar beteendet för 3 metoder som _spec.rb-filerna beskriver för ArticleRepository

- Tillgängliga metoder: add_array, insert, delete, update_article_with_id, find_by_id, find_by_url, find_all, exists? (vilka 3 metoder du väljer att testa av dessa 3 är valfritt).
- Dessa tabeller skall ligga i en Sub Suite i FitNesse: ArticleRepository

Poängsättning: 2p per tabell. 11 tabeller. Max 22p.

Bonusuppgift:

Hitta buggar i min kod - 2p per bugg - max 4p

- En bugg är i det här fallet något oväntat beteende som inte är dokumenterat i något test. (Dvs. en bugg behöver inte innebära att programmet kraschar eller ger felaktiga resultat).
- Dvs. om du tycker att något beter sig oväntat men beteendet är dokumenterat i något test så är det inte en bugg (Om du inte kan motivera varför testet är felaktigt).
- Jag är medveten om minst en bugg som finns i programmet- det bör dock finnas gott om andra med tanke på hur lite tid jag lagt på projektet.