TEST DRIVEN DEVELOPMENT

Marco Console
console@dis.uniroma1.it

LABORATORIO

ESEMPIO

- **Vogliamo implementare una classe che gestisce espressioni aritmetiche**

- Funzionalità
  - Gli oggetti sono inizializzati con una stringa di interi divisa da virgole
  - Il metodo **`validate`** valida le stringhe
  - Il metodo **`calculate`** calcola la somma dei numeri

- Implementiamo con la tecnica TDD classe **`ArithmeticExpression`**

ESEMPIO -- SETUP

- Generiamo un progetto rails
  - **`rails new arexpr`**

- Aggiungiamo Gemfile le seguenti gemme in **`test`**
  - **`rspec-rails`**
  - **`rexml`**

- Lanciamo
  - **`bundle install`**
  - **`rails g rspec:install`**

ESEMPIO -- SETUP

- Creiamo il file **`./ar-expr/lib/arexp.rb`**
  - La classe da implementare

- Creiamo il file **`./ar-expr/spec/arexp_spec.rb`**
  - La definizione dei test

ESEMPIO – CLASSE E SPEC

- Partiamo dallo spec file

```
require 'rails_helper.rb'
require `arexpr.rb'

describe ArithmeticExpression do

end
```

- Il test fallisce. Definiamo la classe

```
class ArithmeticExpression

end
```

ESEMPIO – TEST BASE

- Con stringa vuota calculate restituisce 0

```
context "Given an empty string" do
   it "returns 0" do
      ae = ArithmeticExpression.new("")
      expect(ae.calculate).to eql 0
   end
end
```

- Cosa definiamo?

```
class ArithmeticExpression
   def initialize ...
   def calculate ...
end
```

ESEMPIO – IL CODICE

- Definiamo il codice minimo affinché il test abbia successo

```
class ArithmeticExpression

   def initialize(expr)
      @expr = expr
   end

   def calculate()
      0
   end
end
```

ESEMPIO – TEST 1

- Con stringa 1 calculate restituisce 1

```
context "Given 1 ; Calculate" do
   it "returns 1" do
      ae = ArithmeticExpression.new(“1")
      expect(ae.calculate).to eql 1
   end
end
```

ESEMPIO – IL CODICE

- Definiamo il codice minimo affinché il test abbia successo

```
def calculate()
      i = @expr.to_i
      i
   end
```

ESEMPIO – TEST 1, 2

- Con stringa 1, 2 calculate restituisce 3

```
context "Given 1, 2; Calculate" do
   it "returns 3" do
      ae = ArithmeticExpression.new(“1,2")
      expect(ae.calculate).to eql 3
   end
end
```

ESEMPIO – IL CODICE

- Definiamo il codice minimo affinché il test abbia successo

```
def calculate()
      i = 0
    @expr.split(",").each do |v|
       i += v.to_i
    end
    i
  end
```

ESEMPIO – TEST BASE

- Con stringa vuota validate restituisce **true**

```
context "Given an empty string; Validate " do
   it "returns true" do
      ae = ArithmeticExpression.new("")
      expect(ae.validate).to eql true
   end
end
```

ESEMPIO – IL CODICE

- Definiamo il codice minimo affinché il test abbia successo

```
def validate()
    if @expr.eql?("")
     return true
    else return false
    end
  end
```

ESEMPIO – UN ALTRO TEST

- Validiamo una stringa più complessa

```
context "Given 1,2,3; Validate " do
   it "returns true" do
      ae = ArithmeticExpression.new("1,2,3")
      expect(ae.validate).to eql true
   end
end
```

ESEMPIO – IL CODICE

- Definiamo il codice minimo affinché il test abbia successo

```
def validate()
      return not(@expr.match(/\A([0-9]+,)*[0-9]*\Z/).nil?)
end
```

ESEMPIO – MODELLO

- Aggiungiamo un modello per salvare le espressioni nel database

```
rails g model myexpr expr_str:string
rake db:migrate
```

- **Nota**: il comando aggiunge un fixture file per **`myexpr`**
  - **`test/fixtures/myexprs.yml`**

ESEMPIO – UN ALTRO TEST

- Creiamo delle fixture per i test
  - **spec/fixtures/myexprs.yml**

```
one:
  expr_str: "1,2,3"

two:
  expr_str: "MyString"
```

ESEMPIO – TEST CON FIXTURES

- Validiamo e calcoliamo stringhe a partire dalle due fixture definite

```
fixtures :all

context "Given fixture_one; Validate " do
   it "returns true" do
      me = myexprs(:one)
      ae = ArithmeticExpression.new(me.expr_str)
      expect(ae.validate).to eql true
   end
end
```

ESEMPIO – TEST CON FIXTURES

- Validiamo e calcoliamo stringhe a partire dalle due fixture definite

```
context "Given fixture_two; Validate " do
  it "returns false" do
    me = myexprs(:two)
    ae = ArithmeticExpression.new(me.expr_str)
    expect(ae.validate).to eql false
  end
end
```

ESEMPIO – TEST CON FIXTURES

- Validiamo e calcoliamo stringhe a partire dalle due fixture definite

```
context "Given fixture_one; Calculate " do
   it "returns 6" do
      me = myexprs(:one)
      ae = ArithmeticExpression.new(me.expr_str)
      expect(ae.calculate).to eql 6
   end
end
```

ESERCIZIO 1

- Una espressione aritmetica è definita dalla seguente grammatica

- **`expr = bin-expr | n-expr | ( n )`**
- **`bin-expr = bin-op(expr_1, expr_2)`**
- **`n-expr = n-op(expr_1, expr_2, … , expr_n)`**
- **`n-op = + | *`**
- **`bin-op = - | /`**

- La semantica delle espressioni è quella usuale.
- **EX**: +(1, 2, / (4, 2))
  - 1 + 2 + (4 / 2) = 5

ESERCIZIO 1

- Tramite la metodologia TDD, definire una classe che rappresenta espressioni aritmetiche

- Funzionalità.
  - Validazione delle stringhe.
  - Calcolo del valore associato.

- Test
  - La validazione avviene solo una volta
  - Il calcolo avviene solo una volta

ESERCIZIO 1 -- SOLUZIONE

- Tramite la metodologia TDD, definire una classe che rappresenta espressioni aritmetiche

- Funzionalità.
  - Validazione delle stringhe.
  - Calcolo del valore associato.

- Una possibile soluzione.

```
git clone https://gitlab.com/console.marco/
          lassi-tdd-calculator.git
```

ESERCIZIO 2

- Aggiungete il numero di review e la media dei voti alla homepage di Spoiled Potatoes
  - Partite dalla User Story
  - Definite test di accettazione e integrazione con Cucumber+Capybara
  - Definite test unitari con Rspec
  - Partite dal commit con tag **`base-test`**

  `https://gitlab.com/console.marco/spoiled-potatoes.git`
- Se i test non partono, nel test environment
  - Installate webpacker
    - **`RAILS_ENV=test rails webpacker:install`**
  - Aggiungete la gemma rexml
    - **`gem 'rexml'`**

ESERCIZIO 2 -- SOLUZIONE

- Una possibile soluzione è visibile al commit seguente.
  - **`base-autenticazione`**

    `https://gitlab.com/console.marco/spoiled-potatoes.git`

- Nota: La soluzione proposta non usa fixture e quindi ha bisogno di eliminare gli oggetti aggiunti nel database!