

Code Listing

JUnit Test Generator

I.T. Carlow
Software Engineering
Bachelor Degree (With Honours)

Sergio Alcocer Vázquez
C00132732

Introduction

The JUnit Test Generator with Human Oracle (from now on, “the application”) is a tool that helps developers to create JUnit Tests, providing a graphical interface that displays all the methods' information, allowing to associate instructions to test each method, create objects to use in the tests following the constructors. It is written in Java and delivered as a jar file and some other files (images and another Jar file that provides the randomization)

CFG File Structure

In order to store the information of the Tests, the instructions, etc. a configuration file is stored, using the following structure:

```
OBJECT_COUNT = {number of objects, amount of objects that are going to be declared}
# for each {OB} in {all objects}
  O_{OB number, from 0 to number of objects - 1}_OBJECT_NAME = {object name}
  O_{OB number}_PARAM_COUNT = {number of parameters that OB has}
  # for each {PARAM} in {OB.parameters}
    O_{object number}_{PARAM number} = {PARAM value}
  # end for
# end for
METHOD_COUNT = {number of methods that has information}
# for each {method} in {all methods}
  {method_number}_INSTRUCTIONS = {method.number_of_instructions}
  {method_number}_METHOD = {result of calling toString() on an object of type Method}
  # for each {instruction} in {method.instructions}
    {method_number}_{instruction_number}_TYPE = {ASSERT|CUSTOM}
    # if instruction type = ASSERT
      {method_number}_{instruction_number}_ASSERT_TYPE = {TRUE|FALSE|EQUALS}
      {method_number}_{instruction_number}_METHOD_ASSOCIATED = {bare name of the method, like getName }
      # if instruction assert type == TRUE OR FALSE
        {method_number}_{instruction_number}_OBJECT_INVOLVED = {object that calls the function}
        {method_number}_{instruction_number}_PARAM_COUNT = {number of parameters to use the method}
        # for each {param} in {instruction.parameters}
          {method_number}_{instruction_number}_{param_number} = {string of the parameter}
        # end for
        {method_number}_{instruction_number}_REST_CODE = {instruction.restCode (is a private attribute of ast.Assert)}
      # else
        {method_number}_{instruction_number}_OBJ1 = {instruction's first parameter of the assert equals}
        {method_number}_{instruction_number}_OBJ2 = {instruction's second parameter of the assert equals}
      # end if
    # else if instruction type = CUSTOM
      {method_number}_{instruction_number}_CODE = {instruction.code (code associated to that custom instruction)}
    # end if
  # end for
# end for
```

Note: whatever is between {} is pseudo-code to be evaluated. Consider # as a way to start lines with for, if, etc.

Package: analyzer

Filename: FileLoader.java

Package: analyzer

Filename: Lexer.java

Package: analyzer

Filename: LexicalError.java

Package: analyzer

Filename: Parser.java

Package: analyzer

Filename: Token.java

Package: analyzer

Filename: TokenConstants.java

Package: ast

Filename: Attribute.java

Package: ast

Filename: Class.java

Package: ast

Filename: Elem.java

Package: ast

Filename: File.java

Package: ast

Filename: Method.java

Package: filters

Filename: ConfigFilter.java

Package: filters

Filename: Filter.java

Package: filters

Filename: JavaFilter.java

Package: generator

Filename: Assert.java

Package: generator

Filename: Custom.java

Package: generator

Filename: Instruction.java

Package: analyzer

Filename: ObjectCreated.java

Package: generator

Filename: TestInfo.java

Package: gui

Filename: CreateInstructionUI.java

Package: gui

Filename: CreateMethodConstructorUI.java

Package: gui

Filename: CreateObjectUI.java

Package: gui

Filename: MainFrame.java

Package: Skeleton

Filename: SkeletonManager.java