Tutorial de Junit

Adaptado por Rosana Martins, Baseado em Daniel Quirino Oliveira http://www.guj.com.br/article.show.logic?id=40

Testes mais fáceis com Junit

JUnit é um framework open-source, feito por Eric Gamma e Kent Beck (dois dos principais nomes por trás da metodologia XP), que lhe dá todo o ambiente necessário para que você possa testar seus códigos antes de promovê-los ao status de "stable release version". A maioria das IDEs (JBuilder, JDeveloper, Netbeans, Eclipse ...) incorporam o JUnit dentro de seu ambiente de desenvolvimento.

Para instalá-lo, simplesmente descompacte o arquivo e adicione o arquivo junit.jar no seu classpath. O JUnit apresenta três diferentes interfaces:

- uma interface texto (junit.textui.TestRunner),
- uma interface AWT (junit.awtui.TestRunner) e
- uma interface Swing (junit.swingui.TestRunner).

Para executar qualquer uma delas, digite no console: java [interface] [classe de testes]. Exemplo: java junit.swingui.TestRunner MyTest.

Criando classes de teste

Para se testar uma classe específica, basta criar uma classe que deve ser **herdeira da classe junit.framework.TestCase**. Para fazer os testes dos métodos de uma certa classe, a classe de teste deve implementar um **testXXX()**, sendo XXX, normalmente (mas não obrigatoriamente), o nome do método da classe que você está testando, mas o JUnit vai rodar só os métodos iniciados com **test**. Nas versões mais novas do Junit não é necessário que o nome do método de teste se inicie com test se antes dele houver a anotação @Test.

Conversor de temperatura

Para o exemplo, vamos implementar um sistema que faça conversão de temperaturas de Celsius para Fahrenheit e vice-versa. As fórmulas para conversão são as seguintes:

Celsius para Fahrenheit: Tf = 1,8 x Tc + 32
Fahrenheit para Celsius: Tc = (5 x Tf - 160)/9

Para nosso sistema de conversão, vamos precisar de uma interface para representar a entidade temperatura (código 1), implementações das escalas Celsius e Fahrenheit (códigos 2 e 3 respectivamente) e nosso conversor universal de temperaturas (código 4).

Depois disso, vamos montar uma classe que, usando o JUnit, vai testar o que pensamos que nossas classes deveriam fazer.

Então, vamos ao código (interface *Temperature*)

```
public interface Temperature{

    public double getValue();

    public void setValue(double value) throws Exception;

    public double getFREEZE(); //retorna a temperatura em que a água congela

    public double getBOIL(); //retorna a temperatura que a água entra em ebulição

    public double getZERO(); //retorna a temperature mais baixa na escala
}
```

Classe que implementa *Temperature* na escala Celsius.

```
public class CelsiusTemperature implements Temperature{

   private double value;

   private final double FREEZE = 0;

   private final double BOIL = 100;

   private final double ZERO = -273;

   public CelsiusTemperature(){ }

   public double getValue(){
      return value;
   }

   public void setValue(double value) throws Exception{
      if(value < ZERO) throw new Exception("Não há temperatura abaixo do zero absoluto");
      else this.value = value;
   }

   public double getFREEZE(){ return FREEZE;}

   public double getBOIL(){ return BOIL;}

   public double getZERO(){ return ZERO;}

   public String toString(){
      return getValue()+" C";
   }

   public boolean equals(Object other){
      if(other instanceof CelsiusTemperature)
         return (other.getValue() == getValue());
      else return false;
   }
}
```

Classe que implementa *Temperature* na escala Fahrenheit.

```
public class FahrenheitTemperature implements Temperature{
    private double value;
    private final double FREEZE = 32;
    private final double BOIL = 212;
    private final double ZERO = -459.4;
    public FahrenheitTemperature(){ }
    public double getValue(){
        return value;
    }
    public void setValue(double value) throws Exception{
        if(value < ZERO) throw new Exception("Não há temperatura abaixo do zero absoluto");
        else this.value = value;
    }
    public double getFREEZE(){ return FREEZE;}
    public double getBOIL(){ return BOIL;}
    public double getZERO(){ return ZERO;}
    public String toString(){
        return getValue()+" F";
    }
    public boolean equals(Object other){
        if(other instanceof FahrenheitTemperature)
            return (other.getValue() == getValue());
        else return false;
    }
}
```

Segue o conversor de temperaturas. Será que ele está correto?

```
public class TemperatureTransformer{

   public TemperatureTransformer(){ }

   public Temperature convert(Temperature temp) throws Exception{
    if(temp instanceof CelsiusTemperature) return convertToFahrenheit(temp);
    else return convertToCelsius(temp);
   }

   private Temperature convertToFahrenheit(Temperature celsius) throws Exception{
    FahrenheitTemperature f = new FahrenheitTemperature();
    double cvalue = celsius.getValue();
    double fvalue = 1.8*cvalue+f.getFREEZE(); // formulinha 1 :)
    f.setValue(fvalue);
    return f;
   }

   private Temperature convertToCelsius(Temperature fahrenheit) throws Exception{
    CelsiusTemperature c = new CelsiusTemperature();
    double fvalue = fahrenheit.getValue();
    double cvalue = (5/9)*fvalue-5*fahrenheit.getFREEZE(); // formulinha 2
    c.setValue(cvalue);
    return c;
   }

}
```

O sistema funciona da seguinte forma: o usuário instancia um bean de uma certa escala termométrica e passa esta instância como parâmetro para o método convert() da classe TemperatureTransformer, que retorna uma instância de uma Temperature.

Além disso, se o usuário tentar atribuir um valor que seja abaixo do zero absoluto, é jogada uma exceção avisando que tal valor não pode ser atribuido.

Aparentemente, nada poderia dar errado aqui, certo? Errado. Veja que **há um erro na implementação** da fórmula para converter a temperatura para a escala Celsius. Ao invés de dividir toda a expressão por 9 só divide o primeiro termo, podendo causar algumas dores de cabeça.

Ainda bem que o sistema converte apenas temperaturas, mas imagine se convertesse moedas! Um desastre. Então, para evitar dores de cabeça e desastres, vamos testar nosso sistema antes de promovê-lo a versão 1.0. Para tanto, temos que criar a classe de teste. O código está abaixo:

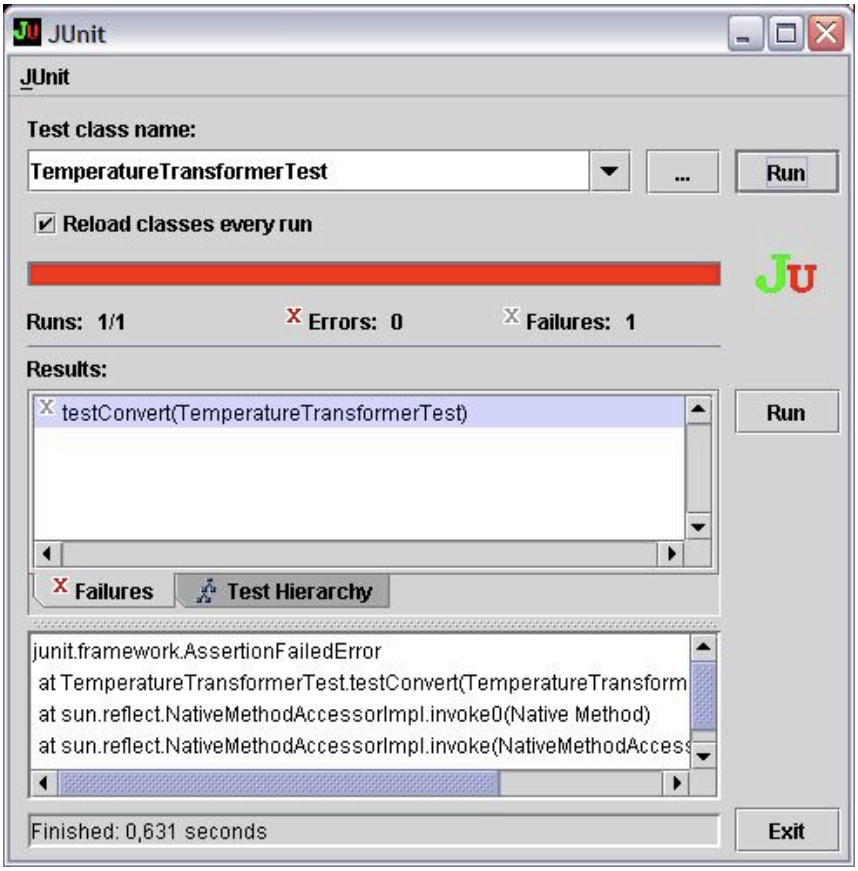
```
import junit.framework.TestCase;

public class TemperatureTransformerTest extends TestCase{
   public void testConvert() throws Exception{
      Temperature t = new FahrenheitTemperature();
      t.setValue(32);
      TemperatureTransformer tc = new TemperatureTransformer();
      Temperature f = tc.convert(t);
      assertTrue(f.getValue() == 0);
  }
}
```

12 linhas de código para criar um teste. Acho que isso não deve tomar muito tempo, certo?

Uma convenção normalmente usada é nomear a classe de testes com **o nome da classe** que vamos testar **seguido da palavra Test**. Ou seja, como vamos testar a classe TemperatureTransformer, então daremos o nome de TemperatureTransformerTest para a classe de testes. Mas, lembre-se: isso é apenas uma convenção. Nada lhe impede de dar os nomes que você quiser para sua classe. No entanto, **o JUnit só executa testes sobre aqueles métodos que começarem com a palavra "test"** ou vierem precedidos pela anotação **@Test**. E isso não é convenção, é obrigatório mesmo.

A parte mais importante desta classe está na linha 10. O método **assertTrue(boolean)** checa se determinada condição é verdadeira. E vai ser a partir da resposta deste método que o JUnit vai fazer a detecção de erros do seu código. Na nossa classe de testes, pediu-se para verificar se a conversão de 32 graus Fahrenheit para Celsius é igual a 0. Porém, como vimos, a fórmula de conversão foi implementada de maneira errada.

E, por causa disso, o JUnit vai acusar uma falha (junit.framework.AssertionFailedError).

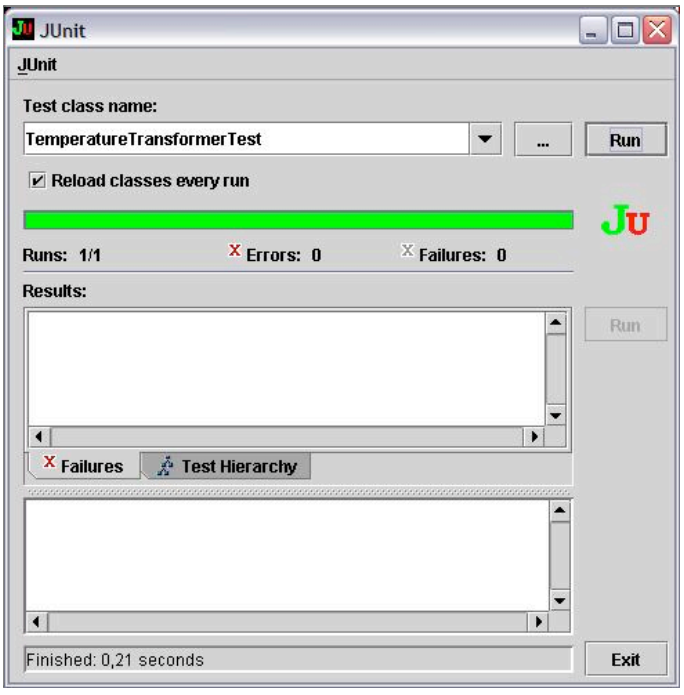

JUnit
JUnit
Test class name:
TemperatureTransformerTest
...
Run
Reload classes every run
Runs: 1/1
Errors: 0
Failures: 1
Results:
testConvert(TemperatureTransformerTest)
Run
Failures
Test Hierarchy
junit.framework.AssertionFailedError
at TemperatureTransformerTest.testConvert(TemperatureTransform
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAcces
Finished: 0,631 seconds
Exit

Detectada a falha, podemos consertar nosso erro:

```
public class TemperatureTransformer{
 public TemperatureTransformer(){ }

 public Temperature convert(Temperature temp) throws Exception{
  if(temp instanceof CelsiusTemperature) return convertToFahrenheit(temp);
  else return convertToCelsius(temp);
  }

  private Temperature convertToFahrenheit(Temperature celsius) throws Exception{
  FahrenheitTemperature f = new FahrenheitTemperature();
  double cvalue = celsius.getValue();
   double fvalue = 1.8*cvalue+f.getFREEZE(); // formulinha 1 :)
 f.setValue(fvalue);
   return f;
  }

  private Temperature convertToCelsius(Temperature fahrenheit) throws Exception{
   CelsiusTemperature c = new CelsiusTemperature();
   double fvalue = fahrenheit.getValue();
   double cvalue = (5*fvalue-5*fahrenheit.getFREEZE())/9;// formulinha 2 :)
 c.setValue(cvalue);
   return c;
  }

}
```

Se fizermos novamente os testes, não veremos mais as mensagens de erro.

E é isso. Como podemos ver, fazer testes não é tão doloroso nem tão demorado assim. Uma boa dica de como fazer testes é ensinada pela metodologia XP, que tem como um de seus pilares testes contínuos. E lembre-se: não acredite puramente nos seus dotes de programador; testes são as únicas formas de garantir que seu software faz exatamente aquilo que ele deveria fazer.



Outros métodos são assertFalse(boolean),

assertEquals(par1,par2), assertNull(object), assertNotNull(object), assertSame(par1,par2), assertNotSame(par1,par2). E antes de cada parâmetro podemos inserir uma frase, para melhor identificação:

assertTrue(**"Erro ao tentar converter 32F para 0C"**, f.getValue() == 0);

Exercícios:

1- O teste que foi realizado garante cobertura de commandos para alguma unidade do sistema? Qual unidade?
2- Faça um método de teste que garante cobertura de comandos para outro componente do sistema. (procure experimentar com outros métodos asserts) http://junit.sourceforge.net/javadoc_40/org/junit/Assert.html
3- Faça agora uma suite de testes incorporando os dois métodos na suite. Execute a suite de testes.
4- Crie testes com cobertura de desvios para as duas classes de temperatura. Neste caso você vai precisar da Anotação @Test passando a exceção esperada como parâmentro. Consulte aqui mais informações sobre as anotações: http://junit.sourceforge.net/javadoc_40/org/junit/package-tree.html

Observação:
Se você não encontrar o JUnit na máquina acesse o arquivo .jar do JUnit na pasta do drive U: -> SI -> 6 periodo -> labs-> Junit

Com o botão da direita, clicar no nome do projeto java e selecionar buid path e depois configure build path. Em java buid path selecionar a aba libraries e ai Add External jars