JUnit

Alexandre Menezes Silva
alexandre_crvg@hotmail.com

Eduardo Manuel de Freitas Jorge
emjorge1974@gmail.com

Sumário

O que é?

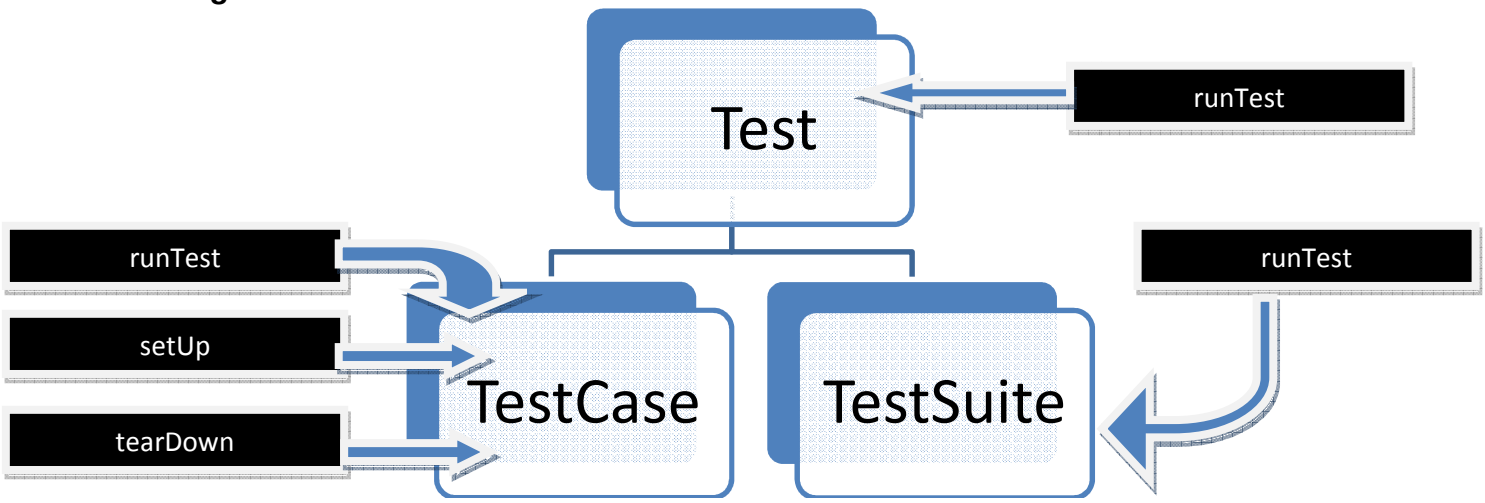
JUnit é um Framework open-source utilizado para facilitar o desenvolvimento de códigos em Java verificando se os resultados gerados pelos métodos são os esperados. Caso não sejam, o JUnit exibe os possíveis erros que estão ocorrendo nos métodos. Essa verificação é chamada de teste unitário ou teste de unidade.

Para que serve?

Atualmente, buscando cada vez mais melhorias nos softwares, os desenvolvedores fazem uma bateria de testes nos seus códigos. Um desses testes é o teste de unidade que testa a menor parte do código garantindo uma maior qualidade do produto no processo de desenvolvimento. No caso da linguagem Java esse teste é feito através do JUnit em cada método separadamente.

Arquitetura

Figura 1



A API do JUnit é organizada da seguinte forma:

- Existe uma classe Test que contem um método runTest responsável por fazer testes particulares.
- Duas classes que herdam da classe Test. A classe TestCase, que testa os resultados de um método, e a classe TestSuite que faz um único teste em mais de um método registrando os resultados.
- A classe TestCase possui também os métodos setUp e TearDown.

Método de comparação – assertEquals()

O método assertEquals (que será exemplificado neste tutorial) é um método que pode ser implementado de várias formas diferentes. Ele recebe como parâmetro o resultado do método que está sendo testado e o resultado esperado pelo desenvolvedor caso o método testado esteja correto. O tipo

desses valores passados como parâmetro pode ser vários (int, double, String, etc...).

Métodos setUp() e tearDown()

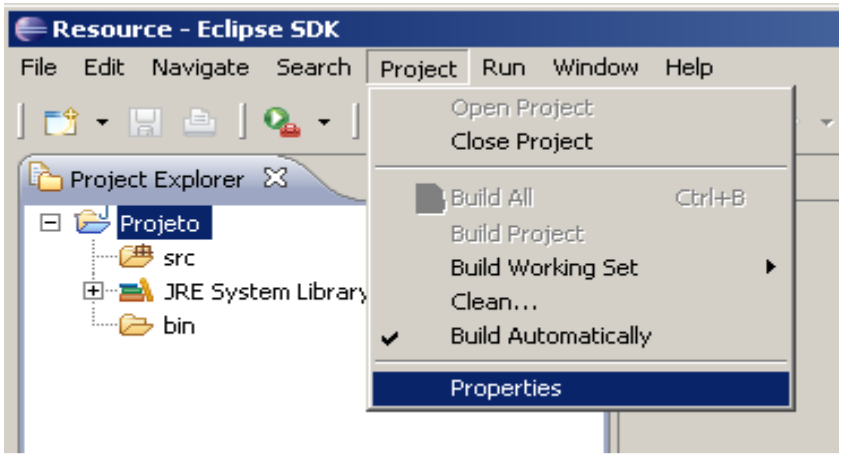
O método setUp() é utilizado para sinalizar o início do processo de teste. Vem antes do método de teste. O método tearDown() sinaliza o final do processo, desfazendo o que o setUp() fez. Vem depois do método de Teste.

Configuração do eclipse

Antes de iniciar o exemplo com o JUnit é necessário inserir no seu Eclipse a biblioteca do Framework. Pode ser encontrado no site http://www.junit.org na seção de downloads o arquivo junit-4.4.jar.

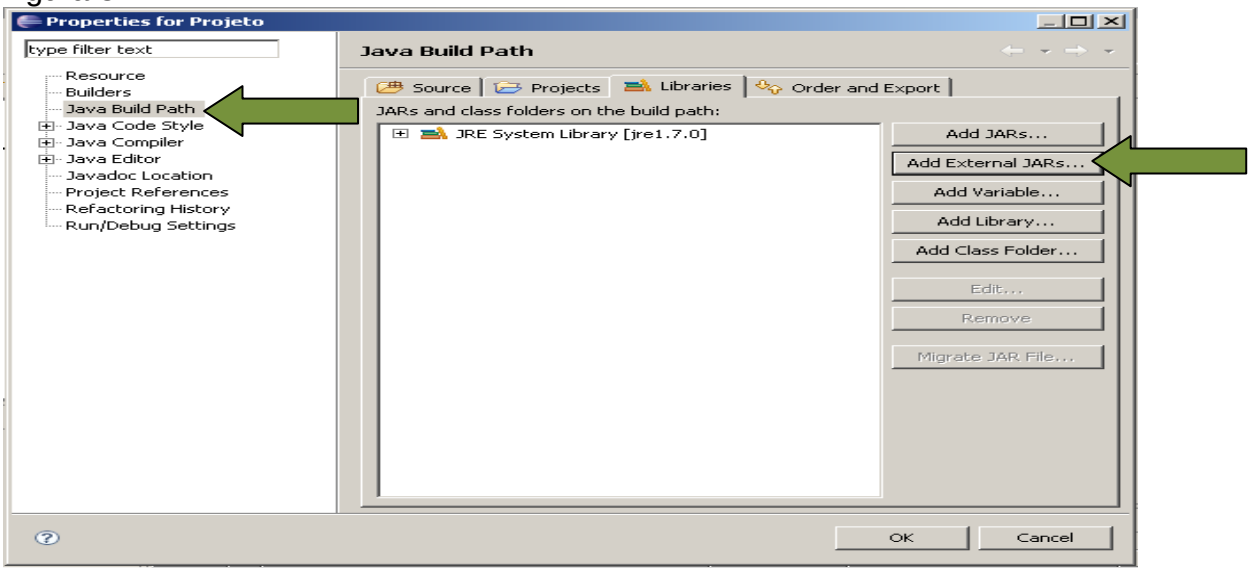
Crie um Projeto com um nome qualquer. Depois vá em Project/Properties, como mostra a figura 2.

Figura 2



Dentro de Properties selecione a opção Java Build Path. Na aba libraries clique em Add External JARs, como mostra a figura 3.

Figura 3



Agora o seu Eclipse está pronto para ser usado com a biblioteca do JUnit.

Exemplo Prático

O exemplo prático consiste na implementação de três classes que são MemoriaS (classe pai), HD (herda de MemoriaS) e CD (também herda de MemoriaS) e a classe de teste do JUnit (que vai herdar de TestCase). Essa última será chamada de MemoriaTeste. As três primeiras classes desse exemplo prático foram geradas a partir de um exercício que pode ser encontrado no site do Professor Eduardo Manuel de Freitas Jorge, através do link http://www.novatec.org/emjorge/tp/lista1.htm.

Implementação das classes

MemoriaS:

```
public abstract class MemoriaS { //Inicio da classe MemoriaS
	public static final int BYTE=1; /*constante que define a unidade
como BYTE*/
	public static final int KB=2; /*constante que define a unidade
como KB*/
	public static final int MB=3; /*constante que define a unidade
como MB*/
	public static final int GB=4; /*constante que define a unidade
como GB*/
	protected double total;
	protected double utilizadoKB;
	protected int unidade;
	/*Método que busca o valor gravado*/
	public double getUtilizadoKB(){
		return this.utilizadoKB;
	}
	/*Método construtor. Recebe como parâmetro quanto deve ser
gravado e a unidade em que deve ser gravado*/
	public MemoriaS(int newTotal,int newUnidade){
		this.total=newTotal;
		this.unidade=newUnidade;
		this.utilizadoKB=0;
	}
	/*Método construtor que recebe como parâmetro quanto deve ser
gravado e define a unidade como KB e chama o outro método construtor*/
	public MemoriaS(int newTotal){
		this(newTotal, KB);
	}
	/*Método abstrato que retornará a perda na gravação*/
	public abstract double getPerda();
	/*Método abstrato que retornará o espaço disponínel real em KB*/
	public abstract double getEspacoDisponivelRealKB();
	/*Método que retorna o espaço disponível em KB*/
	public double getEspacoDisponivelKB(){
		return this.getConverteKB(this.total);
	}
	/*Método utilizado na gravação*/
	public boolean GravaKB(int newTamanho){
```

```
            if(this.getConverteKB(this.total)-
this.utilizadoKB>=newTamanho){
                  this.utilizadoKB=this.utilizadoKB+newTamanho;
                  return true;
            }
            return false;
      }

/*Método que converte a unidade para KB*/
      public double getConverteKB(double valor){
            if(this.unidade==BYTE){
                  return valor/1024;
            }
            else if(this.unidade==MB){
                  return valor*1024;
            }
            else if(this.unidade==GB){
                  return valor*1024*1024;
            }
            else return valor;

      }
      /*Método que busca a unidade*/
      public String getUnidade(){
            if(this.unidade==BYTE){
                  return "BYTE";
            }
            else if(this.unidade==KB){
                  return "KB";
            }
            else if(this.unidade==MB){
                  return "MB";
            }
            else return "GB";
      }

/*Método que busca o percentual disponível*/
public double getPercentualDisponivel(){
            return (this.getConverteKB(this.total)-
this.utilizadoKB)/this.getConverteKB(this.total)*100;
      }

      public String toString(){
            return "Percentual Disponivel:
"+getPercentualDisponivel()+"%\nEspaço Total:
"+getEspacoDisponivelKB()+"KB\nEspaço Disponivel Real:
"+getEspacoDisponivelRealKB()+"KB\nPerda: "+getPerda()+"KB";
      }

      public static void main(String args[]){
            MemoriaS hd = new HD("46327", 10, MemoriaS.MB);
            MemoriaS cd = new CD(650, MemoriaS.MB);
            hd.GravaKB(10242);
            cd.GravaKB(665602);
            System.out.println(hd);
            System.out.println(cd);
      }

}
```

HD:

```
public class HD extends MemoriaS{/*Início da classe HD que estende
MemoriaS*/
	protected String numeroSerie;
	/*Método constructor. Recebe como parâmetro o numero da série, o
tamanho do HD e a unidade que representa o tamanho*/
	public HD(String newNumeroSerie, int newTotal, int newUnidade){
		super(newTotal, newUnidade);
		this.numeroSerie=newNumeroSerie;
	}
	/*Método que reescreve o método abstrato da classe pai. Retorna
o espaço real disponível em KB*/
	public double getEspacoDisponivelRealKB() {
		return this.getEspacoDisponivelKB()-super.getUtilizadoKB();
	}
	/*Método que reescreve o método abstrato da classe pai. Retorna
a perda na gravação*/
	public double getPerda() {
		return this.getEspacoDisponivelKB()/1024/100;
	}
	/*Método que busca o número de série do HD*/
	public String getNumeroSerie(){
		return this.numeroSerie;
	}

	public String toString(){
		return "HD Numero de Serie:
"+this.getNumeroSerie()+"\n"+super.toString();
	}

}
```

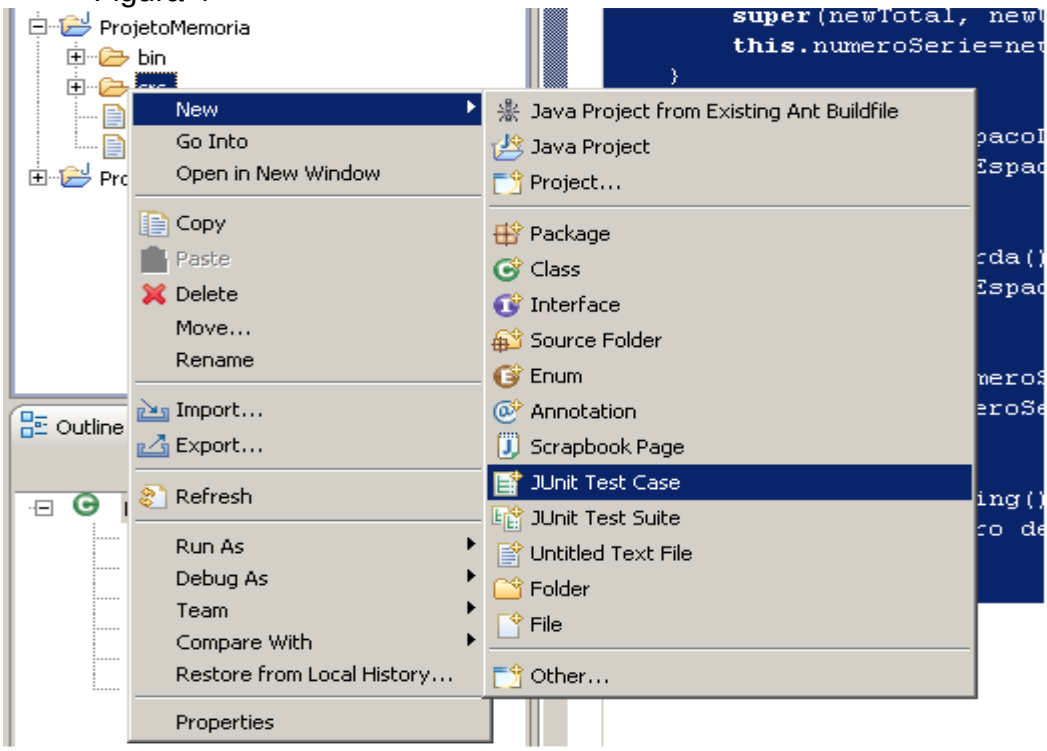
CD:

```
public class CD extends MemoriaS {/*Início da classe CD que estende
MemóriaS*/
	public static final int ABERTO =1;/*constante que define o
estado como aberto*/
	public static final int FECHADO=2; /*constante que define o
estado como fechado*/
	protected int estado;
	/*Método construtor. Recebe como parâmetro o total que deve ser
gravado e a unidade.*/
	public CD(int newTotal, int newUnidade){
		super(newTotal, newUnidade);
		this.estado=ABERTO;
	}
	/*Método que reescreve o método abstrato da classe pai. Retorna
o espaço real disponível em KB*/
	public double getEspacoDisponivelRealKB() {

		return this.getEspacoDisponivelKB()-super.getUtilizadoKB();
	}
	/*Método que reescreve o método abstrato da classe pai. Retorna
a perda na gravação*/
	public double getPerda() {

		return this.getEspacoDisponivelKB()*0.02;
	}
```

```
	/*Método que reescreve o método da classe pai. Utilizado para
gravar recebendo como parâmetro o tamanho.*/
	public boolean GravaKB(int newTamanho){
		if(this.estado==ABERTO){
			if(super.GravaKB(newTamanho)){
				this.estado=2;
				return true;
			}
			else return false;
		}
		else return false;
	}
	/*Método que busca o estado (aberto ou fechado)*/
	public String getEstado(){
		if(this.estado==ABERTO)
			return "ABERTO";
		else return "FECHADO";
	}

	public String toString(){
		return "CD Estado:
"+this.getEstado()+"\n"+super.toString();
	}
}
```

Para criar a classe MemóriaTeste clique com o botão direito na pasta src do projeto, New/ JUnit Test Case, como mostra a figura 4.

Figura 4



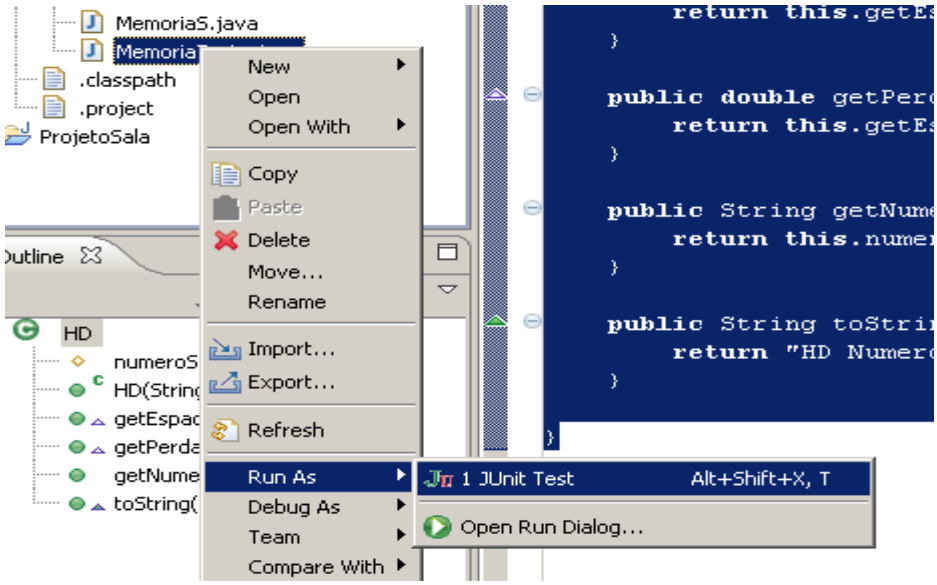
MemoriaTeste:

```
import junit.framework.Test;
import junit.framework.TestCase;
/*Início da classe de teste*/
```

```
public class MemoriaTeste extends TestCase implements Test{
	/*Método construtor que utiliza o construtor da classe pai
TestCase*/
	public MemoriaTeste() {
		super();
		}
	/*Método utilizado para sinalizar o início da classe de teste*/
	protected void setUp() {

		System.out.println("Iniciando...");
		}
	/*Método que vai testar o método GravaKB das classes CD e HD.
Repare na assinatura do método. Não retorna nada e o nome do método
começa sempre com test. */
	public void testGravaKB(){
		CD cd = new CD(10000, 2);
	/*Método assertEquals. Nesse caso recebe como parâmetro dois
atributos booleanos. Um é o retorno do método GravaKB e o outro é o
retorno esperado caso a implementação do método esteja certa.*/
		assertEquals(cd.GravaKB(10000), true);
		assertEquals(cd.GravaKB(10001), false);

		HD hd = new HD("01", 1000, 2);
		assertEquals(hd.GravaKB(1000), true);
		assertEquals(hd.GravaKB(1001), false);

	}


	/*Método que sinaliza o final do processo de teste desfazendo o
trabalho do método steUp().*/
	protected void tearDown(){
		System.out.println("Finalizando...");
		}

}
```

Seguindo todos os passos corretamente, agora teste o seu JUnit. Clique com o botão direito na classe MemoriaTeste Run As/ JUnit Test.

Figura 5



Conclusão

O teste de unidade é importante na construção de métodos, pois permite ao programador testá-los durante a construção do sistema garantindo com isso a implementação de métodos livres de erros lógicos, que ocorrem com muita freqüência. Por possibilitar os testes antes da conclusão do sistema, o programador pode testar seus métodos separadamente assim que eles estejam prontos. Isso evita que o programador fique percorrendo o código inteiro para descobrir os erros lógicos que apareciam quando o programa já estava pronto.

Referências

http://www.javafree.org/content/view.jf?idContent=213%A0

http://www.junit.org/

http://www.novatec.org/emjorge