**QA – JUnit**

JUnit é um Framework de teste de unidade para a linguagem de programação Java, proveniente de uma família de frameworks de Teste de Unidade denominada xUnit.

Suporta diversas linguagens de programação, como C e C++, Java, Phyton, PHP, entre outras;

Utilizaremos o JDK 1.5 ou superior, o Framework de Teste JUnit e a IDE Eclipse, ambos disponíveis gratuitamente para download.

**JUnit Framework – Características:**

- Open Source;

- Utiliza Annotations para identificar métodos de teste;

- Fornece Assertions para avaliar valores esperados;

- Possibilita a execução de testes automatizados, com feedback imediato ao testador.

**Principais Tipos de Annotations:**

|  |  |
| --- | --- |
| **Annotation** | **Descrição** |
| **@Test** | Informa ao JUInit que um método ***public void*** que está associado pode ser executado como um caso de teste |
| **@Before** | Diversos testes necessitam de objetos similares, criados antes da execução. Essa anotação indica que o método anotado deve ser executado antes da execução de cada método de teste |
| **@After** | Se um determinado recurso é alocado em um método com anotação **@Before**, o mesmo deverá ser liberado. Anotar um método com **@After** faz bom que sua execução ocorra após a execução de cada método de test |
| **@BeforeClass** | Informa que um método deve ser executado uma única vez, antes de cada caso de teste da classe |
| **@AfterClass** | Informa que um método deve ser executado, uma única vez, após todos os testes terem sido realizados |
| **@Ignore** | Utilizado para ignorar um teste que não deve ser executado |

**API JUnit – Principais Classes do *junit.framework*:**

|  |  |
| --- | --- |
| **Classes** | **Funcionalidades** |
| **Assert** | Conjunto de métodos de ***Assertions*** |
| **TestCase** | Define características de um caso de teste |
| **TestResult** | Coleta os resultados da execução de um caso de teste |
| **TestSuite** | Composição de um conj. de testes |

**Teste Funcional – Passos para sua aplicação:**

- Analisar as especificações de requisitos;

- Escolher Entradas Válidas para determinar se o produto em teste se comporta corretamente;

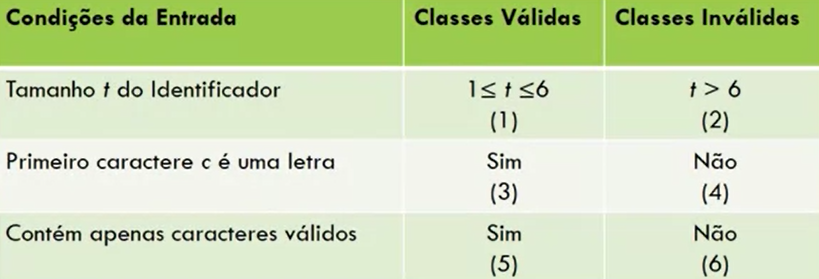
- Escolher Entradas Inválidas para verificar se estas estão detectadas e manipuladas adequadamente;

- Construir os casos de testes – Saídas são determinadas para cada entrada;

- Executar o conjunto de teste e obter as saídas, comparando com os resultados esperados;

- Gerar um relatório para avaliação dos resultados dos testes.

**Exemplo de caso – Software *Identifier:***

******

**Especificações para o software Identifier:**

- Nome – *Identifier*: Determina se um identificador é válido ou não.

- Uso – *IdentifierMain <string>*

- Descrição: O programa Identifier determina se um identificador é válido ou não. Um identificador válido deve começar com uma letra e conter, apenas, letras ou números. Além disso, deve conter, ao mínino, um caractere e, ao máximo, seis caracteres de comprimento.

**Exemplos:**

- Verificação de um Identificador válido: Respeitar as restrições impostas na descrição;

> ***IdentifierMain “string”* – Output: *Valido***;

- Verificação de um Identificador inválido: Extrapolar o número máximo de caracteres aceitos:

> ***IdentifierMain “stringmuitogrande”* – Output: Inválido**;

**Obs:** Adicionamos as Bibliotecas JRE e JUnit ao projeto do Eclipse para efetuarmos os testes. Verificar no Eclipse o projeto ***Identifier***.