Testes usando JUnit

Cristiano Amaral Maffort

cristiano@decom.cefetmg.br

Técnico em Informática

Departamento de Computação

CEFET-MG – Belo Horizonte

Testes de software

- Diretriz do manifesto ágil
  - “responder a mudanças mais que seguir um plano”
    - Para que seja possível, são necessários mecanismos que nos tragam segurança ao fazer alterações
      - evitando o surgimento de efeitos colaterais.
  - “software em funcionamento mais que documentação abrangente”
    - Escrever teste com boa qualidade é importante pois:
      - cria uma fonte de documentação;
      - facilita a manutenção e a evolução da aplicação;
    - Cuidado! Outro artefato de código e, como tal,
      - Necessita da mesma atenção que o código-fonte.

Automação de Testes

- Um teste automatizado é um trecho de código que testa uma parte da aplicação que estamos desenvolvendo.
- Basicamente, é uma prática que utiliza software para automatizar o teste de outro software
- Tornou-se mais comum com a popularização de metodologias ágeis
- Existem centenas de ferramentas para automação de testes

Automação de Testes

- Cada teste corresponde a uma unidade específica da aplicação
  - Para abrangê-la completamente, necessitamos de um conjunto de casos de teste
    - Esse conjunto é chamado de suíte de testes
- A implementação de um teste requer 3 etapas:
  - criação do cenário de teste;
  - execução/realização do cenário;
  - verificação de aderência com o resultado esperado.

JUnit

- Ferramenta open source padrão para automação de testes Java
- Projetado para suportar o sucesso ou a falha de uma operação sem a necessidade de interpretar resultados

JUnit

- @Test: indica que esse método é um teste
- @BeforeClass: análogo ao construtor da classe
- @AfterClass: análogo ao destrutor da classe
- @Before: executa método antes de cada teste
- @After: executa método após cada teste
- Assert: verifica aderência entre resultado obtido e esperado
  - Ex. de métodos: assertEquals(), assertNotEquals(), assertTrue(), assertFalse(), assertNull, assertNotNull(), etc.

Junit - Exemplo

```
public class MyTest {
   protected static EJBContainer container;
    protected static Logger logger;

    @BeforeClass
    public static void setUpClass() throws Exception {
        container = EJBContainer.createEJBContainer();
        logger = Logger.getLogger(BaseTest.class.getName());
    }

     @AfterClass
    public static void tearDownClass() throws Exception {
        container.close();
    }
```

Junit - Exemplo

```
@Before
 public void setUp() {
     try {
    UserTransaction transaction = (UserTransaction)
        container.getContext().lookup("java:comp/UserTransaction");
         transaction.begin();



    } catch (Exception ex) {
        logger.log(Level.SEVERE, null, ex);
        throw new RuntimeException(ex);
    }
}
```

Junit - Exemplo

```
@After
public void tearDown() {
    try {
        UserTransaction transaction = (UserTransaction)
            container.getContext().lookup("java:comp/UserTransaction");
       transaction.rollback();
    } catch (Exception ex) {
       logger..log(Level.SEVERE, null, ex);
       throw new RuntimeException(ex);
    }
}
@Test
public void testCadatroObra() throws Exception {

    CadastroFacade facade = (CadastroFacade) c
      container.getContext().lookup("java:global/classes/CadastroFacade");
    ....
}
```

Casos de Teste – Boas Práticas

- Conheça bem os requisitos que se quer testar
- Escreva casos de teste para interfaces
- Maximize a cobertura do teste
- Não confie na ordem dos casos de teste
  - Escreva testes autosuficientes, ou seja, que são executados independentemente de qualquer outro teste.
- Escreva casos que teste que favoreçam a identificação e a legibilidade dos requisitos