TDD

Testes Unitários:

Um teste de unidade testa uma única unidade do nosso sistema. Geralmente, em sistemas orientados a objetos, essa unidade é a classe. Em nosso sistema de exemplo, muito provavelmente existem classes como “CarrinhoDeCompras”, “Pedido”, e assim por diante. A ideia é termos baterias de testes de unidade separadas para cada uma dessas classes; cada bateria preocupada apenas com a sua classe.

```
public class MaiorEMenor {

    private Produto menor;
    private Produto maior;

    public void encontra(CarrinhoDeCompras carrinho) {
        for(Produto produto : carrinho.getProdutos()) {
            if(menor == null || produto.getValor() < menor.getValor()) {
                menor = produto;
            }
            else if (maior == null ||
                    produto.getValor() > maior.getValor()) {
                maior = produto;
            }
        }
    }

    public Produto getMenor() {
        return menor;
    }

    public Produto getMaior() {
        return maior;
    }

}
```

```
public class TestaMaiorEMenor {
    public static void main(String[] args) {
        CarrinhoDeCompras carrinho = new CarrinhoDeCompras();
        carrinho.adiciona(new Produto("Liquidificador", 250.0));
        carrinho.adiciona(new Produto("Geladeira", 450.0));
        carrinho.adiciona(new Produto("Jogo de pratos", 70.0));

        MaiorEMenor algoritmo = new MaiorEMenor();
        algoritmo.encontra(carrinho);

        System.out.println("O menor produto: " +
            algoritmo.getMenor().getNome());
        System.out.println("O maior produto: " +
            algoritmo.getMaior().getNome());
    }
}
```

Veja bem essa é uma implementação simples de um teste unitário, a partir dessa classe de Teste podemos mapear as funcionalidades e verificar erros de forma mais rápida e eficiente.

Testes Automatizados:

Veja bem se analisarmos a classe de teste acima percebemos que mesmo que ela automatize algumas tarefas e teste a unidade ainda é necessário que a intervenção de um ser Humano para analisar as saídas e registrar os erros, nesse sentido é importante também adicionar automatização nesse registro, isso pode ser feito através de códigos que façam isso ou podemos utilizar de **frameworks** de automatização de testes, no Java o mais conhecido é o jUnit.

O JUnit possui um plugin para Eclipse, que mostra uma lista de todos os testes executados, pintando-os de verde em caso de sucesso, ou vermelho em caso de falha. Ao clicar em um teste vermelho, a ferramenta ainda apresenta a linha que falhou, o resultado que era esperado e o resultado devolvido pelo método.

Para converter uma classe em um teste jUnit devemos declarar ela como uma classe de teste com a anotação **@Test** e devemos declarar quais são os resultados esperados através do método **Assert.assertEquals();**

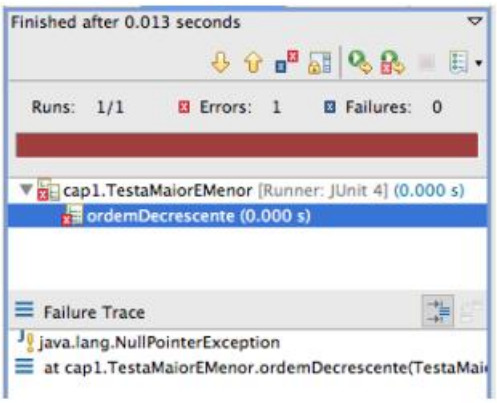
```java
import org.junit.Assert;
import org.junit.Test;

public class TestaMaiorEMenor {

    @Test
    public void ordemDecrescente() {
        CarrinhoDeCompras carrinho = new CarrinhoDeCompras();
        carrinho.adiciona(new Produto("Geladeira", 450.0));
        carrinho.adiciona(new Produto("Liquidificador", 250.0));
        carrinho.adiciona(new Produto("Jogo de pratos", 70.0));

        MaiorEMenor algoritmo = new MaiorEMenor();
        algoritmo.encontra(carrinho);

        Assert.assertEquals("Jogo de pratos",
            algoritmo.getMenor().getNome());
        Assert.assertEquals("Geladeira",
            algoritmo.getMaior().getNome());
    }
}
```



Test-Driven Development:

Quando falamos de TDD estamos falando em testes feitos anteriormente a criação da própria classe, de forma simplificada pensamos primeiro no resultado esperado e montamos a classe de teste, em seguida montamos a classe a ser testado com o mínimo de recursos possíveis para o teste ter sucesso:

```java
public class ConversorDeNumeroRomanoTest {

    @Test
    public void deveEntenderOSimboloI() {
        ConversorDeNumeroRomano romano = new ConversorDeNumeroRomano();
        int numero = romano.converte("I");
        assertEquals(1, numero);
    }

}

public class ConversorDeNumeroRomano {

    public int converte(String numeroEmRomano) {
        return 0;




    }

}
```

Perceba um Detalhe aqui, utilizamos diretamente o asserEquals graças ao import estático da classe: import static org.junit.Assert.*

Após montarmos a estrutura básica, cabe ao desenvolvedor desenvolver as duas classes de forma simultânea.

De maneira mais abstrata, o ciclo que foi repetido ao longo do processo de desenvolvimento da classe acima foi:

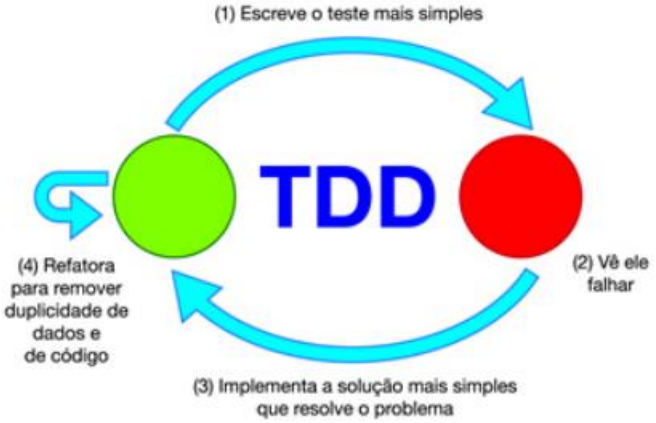
• Escrevemos um teste de unidade para uma nova funcionalidade;

• Vimos o teste falhar;

• Implementamos o código mais simples para resolver o problema;

• Vimos o teste passar;

• Melhoramos (refatoramos) nosso código quando necessário


(1) Escreve o teste mais simples
TDD
(2) Vê ele falhar
(3) Implementa a solução mais simples que resolve o problema
(4) Refatora para remover duplicidade de dados e de código