TDD-Test Driven Design

Prof. Pasteur Ottoni de Miranda Junior
Disponível em http://www.tesestec.com.br/pasteurjr/TDD.pdf

O TDD-Test Driven Design foi criado por Kent Beck (Extreme Programming) e calca-se em ciclos de repetições, onde se cria preliminarmente um teste para cada funcionalidade do sistema. Como a funcionalidade não foi ainda implementada, esse teste falha. Desta feita, implementamos a funcionalidade que faça com que esse teste passe. O ciclo completo é o seguinte (BECK, 2002):

1. Adicionar um teste
2. Executar o teste e verificar que ele falha
3. Efetuar uma pequena mudança no código de forma a fazer o programa funcionar rapidamente.
4. Refatorar , reestruturar
5. Volte a 1

Tutorial

Procedimentos iniciais:

1-Crie um projeto Java chamado TDD
2-Crie uma classe test case do JUnit chamada TestTDD

Passos do ciclo

Como exemplo, vamos criar um programa para obtenção de resultados finais com base em nota, frequência e inadimplência do aluno.

As entradas são a nota, a frequência e uma variável booleana que indica se o aluno é inadimplente ou não. A saída do programa são os resultados possíveis para cada situação:

"Aprovado" : se nota >= 60, frequência >= 75 e não está inadimplente.

"Recuperação":
- se nota >= 60, frequência >= 75 e está inadimplente.
- se nota < 60 e >= 50, frequência >=75 e não está inadimplente.

"Reprovado":
- se nota < 50, frequência qualquer, qualquer situação de inadimplente.
- se nota < 60 e >= 50, frequência >=75 e está inadimplente.

Casos de teste (gerados por grafo causa-efeito, por exemplo):

Nota	Freq	Inadimp	Saída
80	90	False	"Aprovado"
80	90	True	"Recuperação"
55	90	False	"Recuperação"
40	90	False	"Reprovado"
55	90	True	"Reprovado"

Vamos executar, agora, um ciclo de TDD para cada caso de teste acima

Primeiro ciclo

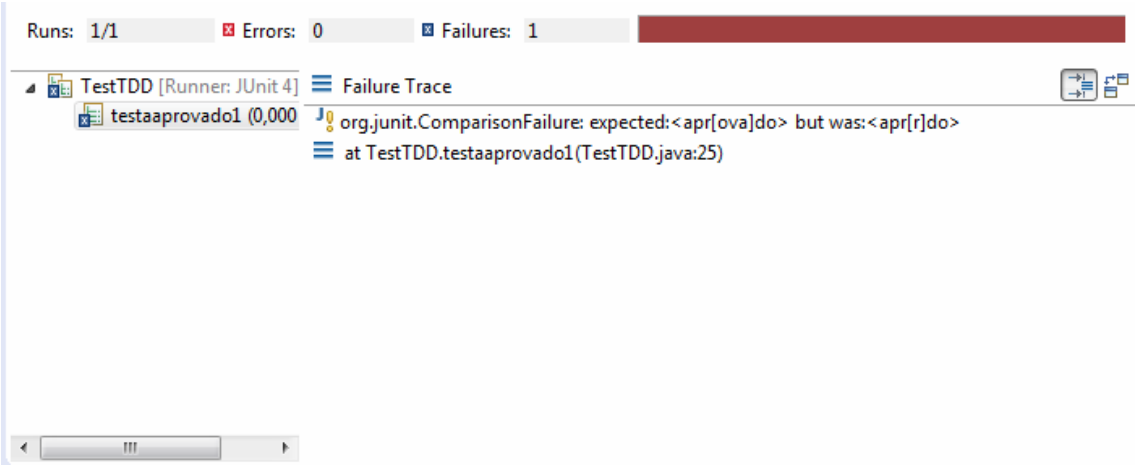
Adicionar um teste

1-Modifique o método test() gerado pelo Junit renomeando-o para testaaprovado1() e acrescente o código Java abaixo, que implementa o primeiro caso de teste:

```
@Test
public void testaaprovado1() {
	int nota = 80;
	int freq = 90;
	boolean inadimp = false;
	String resultado = "aprrdo";
	assertEquals("aprovado",resultado);
}
```

Executar o teste e verificar que ele falha

Importante:Antes de executar qualquer teste, se modificar o código, salve antes!
2-Execute o teste clicando com o botão direito sobre a classe TestTDD e depois Run as-> Junit Test. O resultado deve ser uma "barra vermelha" indicando que o teste falhou, conforme exibido abaixo:



Efetuar uma pequena mudança no código de forma a fazer o programa funcionar rapidamente:

3- Modifique o médodo testaaprovado1() iniciando a implementação:

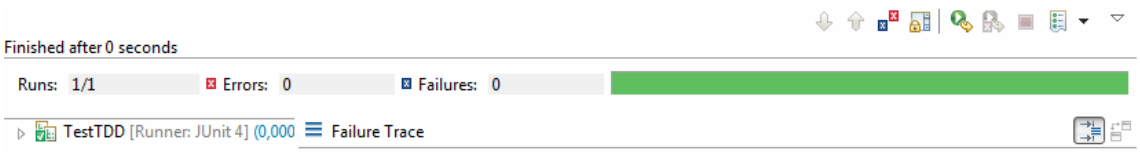
```
public void testaaprovado1() {
	int nota = 80;
	int freq = 90;
	boolean inadimp = false;
	String resultado = calcularresultado(nota,freq,inadimp);
	assertEquals("aprovado",resultado);
}
```

Esse código não compila, portanto vamos implementar o método calcularresultado(). De forma a implementar rapidamente, crie-o dentro da classe TestTDD;

```
public String calcularresultado(int nota, int freq, boolean inadimp)
{
   if ((nota >= 60) && (freq >=75 )  && (inadimp== false)){

         return "aprovado";
   }
   else return "null";
}
```

Salve o programa e execute esse teste. O resultado será a "barra verde" de teste bem sucedido:



Refatorar , reestruturar

Vamos refatorar o programa modificando-o para reestruturá-lo dentro dos princípios da orientação a objetos. Para tal, vamos criar uma classe Resultado que vai conter o método calcula(), que contém a mesma implementação do calcularresultado(). Além disso essa classe terá os atributos nota, freq e inadimp que serão iniciados via construtor.

4-Crie a classe Resultado dentro do mesmo arquivo da classe TestTDD, conforme abaixo (crie-a logo abaixo do import no cabeçalho da classe TesTDD):

```
class Resultado{

	int nota;
	int freq;
	boolean inadimp;
	Resultado(int notai, int freqi, boolean inadimpi){
		nota = notai;
		freq=freqi;
		inadimp = inadimpi;

	}

	String calcular(){
		if ((nota >= 60) && (freq >=75 )  && (inadimp== false)){

				return "aprovado";
		   }
		   else return "null";
		}
}
```

5-Modifique o método testaaprovado1():

```
	public void testaaprovado1() {
		int nota = 80;
		int freq = 90;
		boolean inadimp = false;
		Resultado res = new Resultado(nota,freq,inadimp);
		String resultado = res.calcular();
		assertEquals("aprovado",resultado);
```

```
}
```

6-Execute o teste e a "barra verde" será exibida. Isso conclui o primeiro ciclo de TDD.

Segundo ciclo

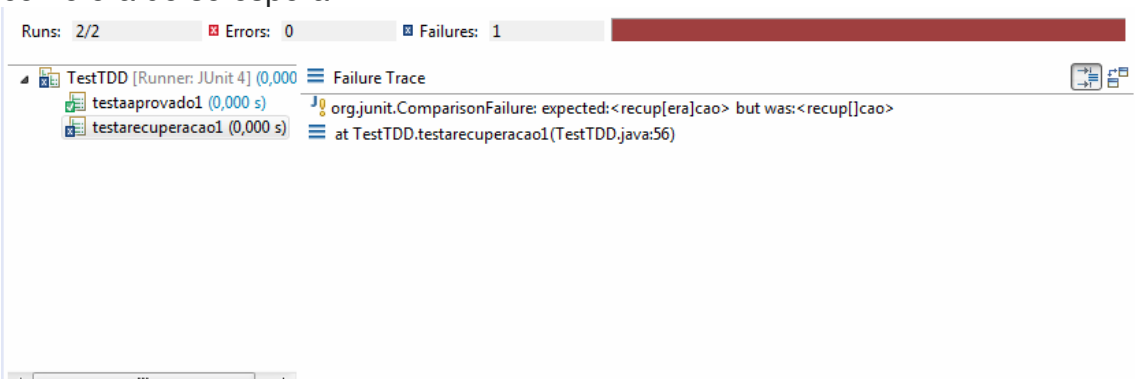
Adicionar um teste

1-Criar dentro da classe testTDD o método testarecuperacao1() abaixo, que implementa o segundo caso de teste (não se esqueça da tag @Test):

```
@Test
  public void testarecuperacao1() {
        int nota = 80;
        int freq = 90;
        boolean inadimp = true;
        String resultado = "recupcao";
        assertEquals("recuperacao",resultado);
  }
```

Executar o teste e verificar que ele falha

2-Execute o teste clicando com o botão direito sobre a classe TestTDD e depois Run as-> Junit Test. O resultado deve ser a "barra vermelha" indicando que esse teste falhou.O "Failure Trace" indica que o teste testarecuperacao1() falhou, mas o testaprovado1(), já realizado anteriormente, foi bem sucedido, como era de se esperar:



Efetuar uma pequena mudança no código de forma a fazer o programa funcionar rapidamente:

2-Modifique o método testarecuperacao1() para incluir agora a chamada da classe Resultado e do método calcular:

```
@Test
public void testarecuperacao1() {
        int nota = 80;
        int freq = 90;
        boolean inadimp = true;
        Resultado res = new Resultado(nota,freq,inadimp);
```

```
            String resultado = res.calcular();
            assertEquals("recuperacao",resultado);
      }
```

3- Modificar o método calcular() para que o teste acima funcione:

```
String calcular(){
            if ((nota >= 60) && (freq >=75 )  && (inadimp== false)){

                  return "aprovado";
               }


          else

           if ((nota >= 60) && (freq >=75 )  && (inadimp== true)){

                  return "recuperacao";
               }

               else return "null";
}
```

4-Execute o teste e a "barra verde" deve ser exibida.

Refatorar , reestruturar

5-O código do método calcular() contém uma duplicação desnecessária das condições (nota >= 60) && (freq >= 75), que serão removidas, conforme abaixo:

```
         String calcular(){
            if ((nota >= 60) && (freq >=75 )  && (inadimp== false)){

                  return "aprovado";
               }


           else

           if  (inadimp== true){

                  return "recuperacao";
               }

           else return "null";
        }
```

6-Execute novamente o teste e será exibida, novamente, a "barra verde", finalizando o segundo ciclo.

Terceiro ciclo

Adicionar um teste

1-Criar dentro da classe testTDD o método testarecuperacao2() abaixo, que implementa o terceiro caso de teste (não se esqueça da tag @Test):

```
@Test
  public void testarecuperacao2() {
        int nota = 55;
        int freq = 90;
        boolean inadimp = true;
        String resultado = "recupcao";
        assertEquals("recuperacao",resultado);
  }
```

Executar o teste e verificar que ele falha

2-Execute o teste clicando com o botão direito sobre a classe TestTDD e depois Run as-> Junit Test. O resultado deve ser a "barra vermelha" indicando que esse teste falhou.

Efetuar uma pequena mudança no código de forma a fazer o programa funcionar rapidamente:

2-Modifique o método testarecuperacao2() para incluir agora a chamada da classe Resultado e do método calcular:

```
@Test
public void testarecuperacao2() {
        int nota = 55;
        int freq = 90;
        boolean inadimp = false;
        Resultado res = new Resultado(nota,freq,inadimp);
        String resultado = res.calcular();
        assertEquals("recuperacao",resultado);
}
```

3- Modificar o método calcular() para que o teste acima funcione:

```
        String calcular(){
        if ((nota >= 60) && (freq >=75 )  && (inadimp== false)){

                return "aprovado";
           }


         else

          if  (inadimp== true){

                return "recuperacao";
           }

            else
                if ((nota < 60) && (nota >= 50) && (freq >=75 )  &&
(inadimp== false)){

                    return "recuperacao";
             }
             else return "null";


          }
```

4-Execute o teste e a "barra verde" deve ser exibida.

Refatorar , reestruturar

5-O código do método calcular() contém uma duplicação desnecessária da condição (inadimp==true) no terceiro "if", que será removida, conforme abaixo:

```
String calcular(){
if ((nota >= 60) && (freq >=75 )  && (inadimp== false)){

        return "aprovado";
   }


 else

  if  (inadimp== true){

        return "recuperacao";
   }

    else
        if ((nota < 60) && (nota >= 50) && (freq >=75 )){

              return "recuperacao";
          }
          else return "null";


   }
```

6-Execute novamente o teste e será exibida, novamente, a "barra verde", finalizando o terceiro ciclo.

OBS: Repare que, a cada ciclo de teste, os testes anteriormente realizados devem continuar funcionando após as alterações no método e refatorações.Se a "barra vermelha" for exibida e algum teste falhar, obviamente, o programa deve ser corrigido até que todos os testes funcionem e a "barra verde" seja exibida.

Quarto ciclo

Adicionar um teste

1-Criar dentro da classe testTDD o método testareprovado1() abaixo, que implementa o terceiro caso de teste (não se esqueça da tag @Test):

```
@Test
  public void testareprovado1() {
        int nota = 45;
        int freq = 90;
        boolean inadimp = false;
        String resultado = "reprovad";
```

```
        assertEquals("reprovado",resultado);
    }
```

Executar o teste e verificar que ele falha

2-Execute o teste clicando com o botão direito sobre a classe TestTDD e depois Run as-> Junit Test. O resultado deve ser a "barra vermelha" indicando que esse teste falhou.

Efetuar uma pequena mudança no código de forma a fazer o programa funcionar rapidamente:

2-Modifique o método testarecuperacao2() para incluir agora a chamada da classe Resultado e do método calcular:

```
    @Test
    public void testareprovado1() {
        int nota = 45;
        int freq = 90;
        boolean inadimp = false;
        Resultado res = new Resultado(nota,freq,inadimp);
        String resultado = res.calcular();
        assertEquals("reprovado",resultado);
    }
```

3- Modificar o método calcular() para que o teste acima funcione:

```
        String calcular(){
        if ((nota >= 60) && (freq >=75 )  && (inadimp== false)){

                return "aprovado";
           }


       else

        if  (inadimp== true){

                return "recuperacao";
           }

           else
                if ((nota < 60) && (nota >= 50) && (freq >=75 )){

                    return "recuperacao";
                }
                else if (nota < 50) {
                        return "reprovado";

                } else return "null";


}
```

4-Execute o teste e a "barra verde" deve ser exibida.

Refatorar , reestruturar

5- A condição (nota < 50) no último "if" pode ser melhor posicionada no topo do código, pois o resultado é "reprovado" sempre que nota < 50, independentemente dos outros parâmetros. Dessa forma, nessa situação, o código é executado de maneira mais eficiente pois evita-se que os outros "ifs" sejam percorridos quando nota < 50, como ocorria na implementação pré-refatoração exibida no item 4. Para tal, modifique o código do método calcular() conforme abaixo:

```
            String calcular(){
            if (nota < 50){
                  return "reprovado";

            }
            else
            if ((nota >= 60) && (freq >=75 )  && (inadimp== false)){

                  return "aprovado";
               }


         else

          if  (inadimp== true){

                  return "recuperacao";
            }

               else
                     if ((nota < 60) && (nota >= 50) && (freq >=75 )){

                           return "recuperacao";
                     }
                     else  return "null";


          }
```

6-Execute novamente o teste e será exibida, novamente, a "barra verde", finalizando o quarto ciclo.

Exercício:

Implemente o último caso de teste em um ciclo análogo aos 4 outros executados anteriormente. Atente para os passos de refatoração e reestruturação e lembre-se que, ao executar todos os 5 testes, a "barra verde" deve ser exibida. **Entregue somente o arquivo TestTDD.java final.**

Referências

BECK, Kent. *Test-Driven Development By Example* - Three Rivers Institute, 2002