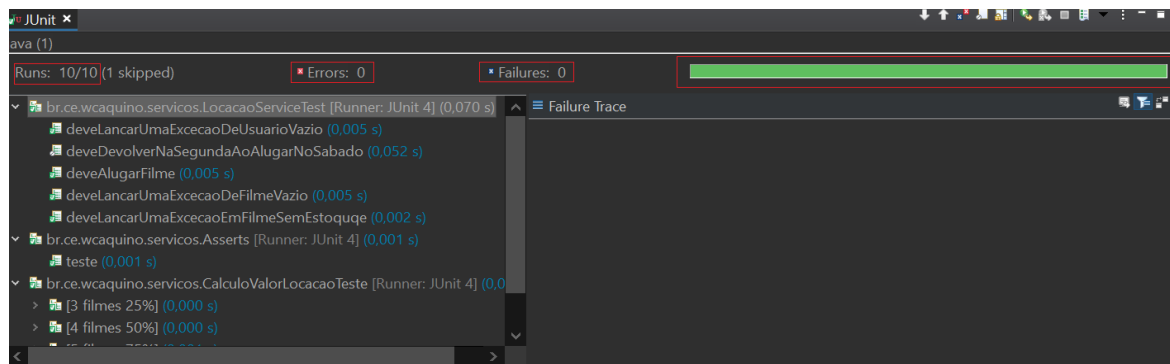


## Análise de cobertura de testes:

A partir da execução de testes unitários podemos extrair algumas métricas, as duas métricas mais comuns são: **Percentual de aceitação dos testes** e **percentual de cobertura de código**.

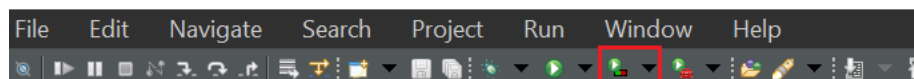
### Percentual de aceitação:

O percentual de aceitação é bem simples, ele é apenas a **quantidade de testes executados com sucesso** dividido pela **quantidade de testes no geral**. Se deixarmos sempre a barra de execução de testes verde, então não precisaremos perder tempo com essa métrica:



### Percentual de cobertura de código:

Para essa segunda métrica precisamos de uma ferramenta externa, basta ir ao marketplace do eclipse e instalar o plugin "**eclemma**", depois de instalado, aparecerá um novo botão que serve para executar os **testes e gerar um relatório**:



No fim da execução as linhas de código ficaram coloridas em **verde, amarelo ou vermelho**:

### Linha verde:

A linha verde indica que essa linha foi executada completamente, ou seja, essa lógica foi coberta por um teste unitário. Então se verificarmos o relatório da classe de **LocacaoService**, percebemos que as linhas em verde dos **"if"** foram executadas completamente, em outras palavras, **existiam testes para elas**:

```

LocacaoService.java x
21 public Locacao alugarFilme(Usuario usuario, List<Filme> filmes) throws
22
23
24     if(filmes == null) {
25         throw new LocadoraException("Lista de filmes está vazia");
26     }
27
28     for(Filme filme : filmes) {
29         if(filme.getEstoque() == 0) {
30             throw new FilmeSemEstoqueException();
31         }
32     }
33
34     if(usuario == null) {
35         throw new LocadoraException("Usuario está vazio");
36     }
37
38     double valorTotal= 0;
39
40     Locacao locacao = new Locacao();

```

```

@Test(expected = FilmeSemEstoqueException.class)
public void deveLancarUmaExcecaoEmFilmeSemEstoque() throws Exception {

```

```

@Test
public void deveLancarUmaExcecaoDeUsuarioVazio() throws FilmeSemEstoqueException {

```

```

@Test
public void deveLancarUmaExcecaoDeFilmeVazio() throws FilmeSemEstoqueException, LocadoraException{

```

## Linha amarela:

indica que a linha foi executada parcialmente, isso só acontece quando o trecho possui alguma lógica, ou seja, aquele trecho de código pode levar a caminhos diferentes. Esses caminhos são chamados de **branches** e basicamente quer dizer que só **um/alguns** caminho(s) do método **foram cobertos por teste**, mas **não todos**.

Um bom exemplo disso é a **checagem da data de devolução do filme**, temos que **devolver o filme no dia seguinte**, menos se for alugado em um **sábado**, filmes **alugados ao sábado devem ser devolvidos só na segunda**:

Como o método que faz **esse teste só é executado aos sábados(através dos Assume)** ele não rodou no dia **que executamos a bateria de teste(hoje é terça-feira)**, portanto esse caminho não foi coberto e ele nem chegou a executar a linha que **adiciona mais um dia a data de entrega**(que por si só já setava um dia a mais):

```

//Entrega no dia seguinte
Date dataEntrega = new Date();
dataEntrega = adicionarDias(dataEntrega, 1);
if(DataUtils.verificarDiaSemana(dataEntrega, Calendar.SATURDAY)) {
    dataEntrega = adicionarDias(dataEntrega, 1);
}
locacao.setDataRetorno(dataEntrega);

```

66 1 of 2 branches missed

Teste que **checa se a data de devolução esta correta**, no **Assume** dizemos que ele será executado quando não for sábado, ou seja, adiciona um dia a mais referente data de locação que será a data de devolução:

```

@Test
public void deveAlugarFilme() throws Exception {
    Assume.assumeFalse(DataUtils.verificarDiaSemana(new Date(), Calendar.SATURDAY));
}

```

Esse é o teste que **checa se a data de devolução será segunda caso o filme seja alugado em um sábado**, como hoje(dia que escrevo isso) é terça e o **Assume** esta satado para true quando for sábado, ele não executa:

```

@Test
public void deveDevolverNaSegundaAoAlugarNoSabado() throws FilmeSemEstoqueException, L
    Assume.assumeTrue(DataUtils.verificarDiaSemana(new Date(), Calendar.SATURDAY));
//GIVEN/DADO QUE
Usuario caraQueAlugou = UsuarioBuilder.umUsuario().agora();
List<Filme> listaDeFilmes = Arrays.asList(
    FilmeBuilder.umFilme().agora(),
    FilmeBuilder.umFilme().agora());

```

E foi exatamente por isso que a linha que verifica essa condição no método de alugar um filme no serviço de locação ficou amarela, somente uma das duas branches foram cobertas por teste.

## linha vermelha:

Significa que a linha não foi executada, tirando exceções como o Assume, no geral ela é um bom indicativo para dizer que a determinada linha ou trecho de lógica não tem um teste de cobertura.

## Relatório de cobertura:

Também é gerado um relatório de cobertura:

JUnit Coverage					
Element		Coverage	ed Instructions	ed Instructions	tal Instructions
✓ TestesUnitarios		72,7 %	824	310	1.134
> src/test/java		73,8 %	522	185	707
> src/main/java		70,7 %	302	125	427

Desse relatório não importam muito a porcentagem do package de testes nem do projeto em sí, mas sim das classes que estamos testando:

Element		Coverage	ed Instructions	ed Instructions	tal Instructions
✓ TestesUnitarios		72,7 %	824	310	1.134
> src/test/java		73,8 %	522	185	707
▼ src/main/java		70,7 %	302	125	427
> br.ce.wcaquino.entidades		51,8 %	102	95	197
> br.ce.wcaquino.utils		70,8 %	63	26	89
▼ br.ce.wcaquino.servicos		97,0 %	130	4	134
> LocacaoService.java		97,0 %	130	4	134
> br.ce.wcaquino.exceptions		100,0 %	7	0	7

O motivo desse percentual tão alto é estar evoluindo o código usando TDD.