UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
INSTITUTO METRÓPOLE DIGITAL
BACHARELADO EM TECNOLOGIA DA INFORMAÇÃO
DIM0507 - TESTE DE SOFTWARE I

BOAS PRÁTICAS - JUNIT

Componentes - Grupo 02:
Clarissa Alves Soares

Professora:
Roberta de Souza Coelho

Natal/2018

1. **Não use o construtor do testcase para configurar o caso de teste.**

Deve-se usar o método *set up* para configurar, pois se usar o construtor da classe para fazer configuração, caso seja lançada uma exceção, será enviada uma exceção dizendo que a classe não foi instanciada, o que não dá informações diretas do problema. Caso a configuração seja feita no método *set up*, qualquer exceção lançada é informada qual é e de onde, o que torna muito mais fácil explicar falhas de configuração.

2. **Não presuma a ordem em que os testes dentro de um caso de teste são executados**

Não existe uma ordem pré-definida para execução dos testes e isso é bom para que os testes sejam mais robustos, pois evita o acoplamento temporal. Em casos que faz sentido que os testes sejam feitos em certa ordem, pode-se usar o método estático suite(), apesar de que a API do JUnit não dá garantias de que os testes realmente serão executados na ordem, mas pode-se esperar que os testes declarados com esse método serão executados na ordem em que foram adicionados ao conjunto de testes.

3. **Evite escrever casos de teste com efeitos colaterais**

Os casos de teste que apresentam efeitos colaterais apresentam dois problemas:

- Eles podem afetar dados em que outros casos de teste dependem, pois podem ocasionar falhas em outros testes da mesma suíte de testes;
- Não poder repetir testes sem intervenção manual, ou seja, um caso de teste pode ter atualizado algum estado do sistema para que ele não possa ser executado novamente sem intervenção manual, o que pode consistir em excluir dados de teste do banco de dados (por exemplo).

4. **Chamar os metodos setUp() e tearDown() da superclasse quando subclasses**

Nesse caso, os métodos *setUp()* e *tearDown()* devem chamar os métodos da superclasse para que seja garantido o ambiente definido em outras classes de casos de testes que são superclasse da que está sendo usada, com exceção nos casos em que se projetar a classe base para trabalhar com dados de teste arbitrários, não haverá um problema.

5. **Não carregue um dado de locais hard coded em um filesystem**

Deve-se tomar cuidado com a criação de dados em disco porque pode gerar problemas com sistemas operacionais diferentes e colocar o local por extenso também pode remeter a uma pasta que não existe.

6. Mantenha a fonte de testes no mesmo local da fonte do código

Se a fonte de teste for mantida no mesmo local que as classes testadas, o teste ea classe serão compilados durante uma compilação. Isso obriga você a manter os testes e as classes sincronizados durante o desenvolvimento. Na verdade, testes unitários não considerados parte da construção normal tornam-se rapidamente datados e inúteis.

7. Nomes dos testes apropriados

Nomes apropriados ajudam na legibilidade de cada teste proposto. Assim, o nome da classe deverá remeter a classe testado, por exemplo se a classe é Pessoa, a classe que implementa os testes dessa classe deve chamar PessoaTeste, e os métodos devem remeter ao que eles fazem.

8. Assegure que os testes são atemporais

Sempre que possível, evite usar dados que possam expirar; esses dados devem ser revistos manual ou programaticamente. Muitas vezes, é mais fácil instrumentar a classe sob teste, com um mecanismo para mudar a noção de hoje. O teste pode então funcionar de forma independente do tempo sem ter que atualizar os dados.

9. Considere a localização ao escrever testes

Um teste que usa datas deve levar em consideração as diferentes localizações para que o código seja mais resiliente a mudanças.

10. Utilize os métodos de assert/fail do JUnit e o tratamento de exceções para o código de teste limpo

Recomenda-se usar os metodos fail/assert do JUnit, pois são mais apropriados para deixar o código mais legível. Também é importante tratar as exceções para evitar falhas quando o caso de teste for executado.

11. Documento de teste em javadoc

Se possível, deve-se incluir os planos de teste no javadoc dos testes, garantindo que todos os dados do plano de teste concentrem-se em um só lugar.

12. Evite a inspeção visual

Testar servlets, interfaces de usuário e outros sistemas que produzem saída complexa geralmente são deixados para inspeção visual. Inspeção visual - um ser humano que inspeciona

dados de saída para erros - requer paciência, capacidade de processar grandes quantidades de informações e grande atenção aos detalhes: atributos que muitas vezes não são encontrados no ser humano médio.