Bacharel em Engenharia de Software

Modelos, Métodos e Técnicas de Engenharia de Software

Visão e Análise de Projetos – Integração

Professor: Rubem Koide

Alunos/RA:

Caroline Araldi	172320037
Marcelo Augusto Dos Santos	172313472
Nicole Karolyne Balão	172313357
João Victor	942321478

Curitiba, PR

2023

1. Sistemas de Biblioteca

1.1 Biblioteca Java

A classe Biblioteca implementa alguns serviços típicos de uma biblioteca, como adicionar um livro ao acervo, emprestar um livro a um usuário e receber um livro emprestado.

O método adicionarLivroAcervo() recebe um objeto Livro como parâmetro e o adiciona ao acervo da biblioteca. O método emprestarLivro() também recebe um objeto Livro como parâmetro, mas também recebe um objeto Usuario como parâmetro. O método verifica se o livro já está emprestado e, se não estiver, empresta o livro para o usuário. O método receberLivroEmprestado() recebe um objeto Livro como parâmetro e o recebe de volta da biblioteca. O método livrosEmprestadosUsuario() recebe um objeto Usuario como parâmetro e retorna uma lista de livros emprestados ao usuário.

O teste de unidade BibliotecaTest verifica o comportamento esperado da classe Biblioteca para os seguintes cenários:

- Adicionar um livro ao acervo.

- Emprestar um livro que não está emprestado.

- Tentar emprestar um livro que já está emprestado.

O teste deveAdicionarLivroAcervo() verifica se o método adicionarLivroAcervo() adiciona o livro ao acervo da biblioteca. O teste deveEmprestarLivroSeLivroNaoEstaEmprestado() verifica se o método emprestarLivro() empresta o livro para o usuário, se o livro não estiver emprestado. O teste deveLancarExcecaoSeLivroJaEmprestado() verifica se o método emprestarLivro() lança uma exceção se o livro já estiver emprestado.

Passo a passo do que acontece no teste deveEmprestarLivroSeLivroNaoEstaEmprestado():

1. O teste cria uma nova instância da classe Biblioteca.

2. O teste cria um novo objeto Livro com o ISBN 978-85-92-20000-0 e o título O Pequeno Príncipe.

3. O teste chama o método adicionarLivroAcervo() para adicionar o livro ao acervo da biblioteca.

4. O teste chama o método emprestarLivro() para emprestar o livro para o usuário Fulano.

5. O teste chama o método livroEmprestado() do repositório para verificar se o livro está emprestado.

6. O teste espera que o método emprestarLivro() seja bem-sucedido e que o método livroEmprestado() retorne true. Se o teste passar, significa que o método emprestarLivro() está funcionando corretamente.

RESULTADO DOS TESTES Sistema de Bibliotecas

Tempo: 10 ms

4 testes executados com sucesso.

2. Stack

2.1 Main.java

O código é um exemplo de teste de unidade para a classe Biblioteca. O teste verifica se o método adicionarLivroAcervo() da classe Biblioteca adiciona o livro ao acervo da biblioteca.

O teste começa criando uma nova instância da classe Biblioteca. Em seguida, o teste cria um novo objeto Livro com o ISBN 978-85-92-20000-0 e o título

O Pequeno Príncipe. O teste chama o método adicionarLivroAcervo() para adicionar o livro ao acervo da biblioteca.

Finalmente, o teste chama o método livroExiste() do repositório para verificar se o livro está no acervo. O método livroExiste() retorna true se o livro estiver no acervo.

Se o teste passar, significa que o método adicionarLivroAcervo() está funcionando corretamente.

Passo a passo do que acontece no teste:

1. O teste cria uma nova instância da classe Biblioteca.

2. O teste cria um novo objeto Livro com o ISBN 978-85-92-20000-0 e o título O Pequeno Príncipe.

3. O teste chama o método adicionarLivroAcervo() para adicionar o livro ao acervo da biblioteca.

4. O teste chama o método livroExiste() do repositório para verificar se o livro está no acervo.

5. O teste espera que o método adicionarLivroAcervo() seja bem-sucedido e que o método livroExiste() retorne true. Se o teste passar, significa que o método adicionarLivroAcervo() está funcionando corretamente.

No exemplo, o teste usa um repositório mock para simular o comportamento de um repositório real. Isso permite que o teste se concentre na verificação do comportamento da classe Biblioteca, sem se preocupar com a implementação do repositório.

É importante escrever testes de unidade para todas as classes e métodos de um sistema. Os testes de unidade ajudam a garantir a qualidade do código e a identificar erros precocemente no processo de desenvolvimento.

RESULTADO DOS TESTES STACK

Tempo: 9 ms

5 testes executados com sucesso.

3. Google Guava

3.1 Main.java

O código é um exemplo de como usar a biblioteca Google Guava para escrever testes de unidade. A biblioteca Guava fornece uma variedade de ferramentas e recursos que podem ser usados para escrever testes de unidade eficientes e eficazes.

No exemplo, a classe Main é a classe principal da aplicação. Ela é responsável por configurar o Runner do JUnit e chamar os testes da aplicação.

A classe TestRunner é responsável por chamar todos os testes anotados com a anotação @Test. A anotação @Test é usada para indicar que um método é um teste de unidade.

A classe TesteGoogleGuava contém alguns testes de unidade da biblioteca Guava. Os testes são copiados e adaptados do código da biblioteca Guava.

- O primeiro teste, testToList(), verifica se o método toList() da classe Arrays converte um array para uma lista. O teste passa.

- O teste testToList() começa criando um array de inteiros. Em seguida, o teste chama o método toList() para converter o array em uma lista. O teste então verifica se a lista contém os mesmos valores que o array original.

- O teste passa porque a lista contém os mesmos valores que o array original.

- O segundo teste, testAsList(), verifica se o método asList() da classe Arrays cria uma lista a partir de um array. O teste passa.

- O teste testAsList() começa criando um array de inteiros. Em seguida, o teste chama o método asList() para criar uma lista a partir do array. O teste então verifica se a lista contém os mesmos valores que o array original.

- O teste passa porque a lista contém os mesmos valores que o array original.

- O terceiro teste, testIterable(), verifica se o método Iterable() da classe Arrays retorna um iterador para um array. O teste passa.

- O teste testIterable() começa criando um array de inteiros. Em seguida, o teste chama o método Iterable() para retornar um iterador para o array. O teste então verifica se o iterador pode ser usado para iterar sobre o array.

- O teste passa porque o iterador pode ser usado para iterar sobre o array.

- O quarto teste, testCollection(), verifica se o método Collection() da classe Arrays retorna uma coleção para um array. O teste passa.

- O teste testCollection() começa criando um array de inteiros. Em seguida, o teste chama o método Collection() para retornar uma coleção para o array. O teste então verifica se a coleção contém os mesmos valores que o array original.

- O teste passa porque a coleção contém os mesmos valores que o array original.

RESULTADO DA EXECUÇÃO DOS TESTES Google Guava

Tempo: 17 ms

4 testes executados com sucesso.

4. Mocks

4.1 Book.java

O código é um exemplo de como usar mocks para escrever testes de unidade. Um mock é um objeto simulado que pode ser usado para substituir um objeto real em um teste.

No exemplo, a classe BookService depende de um objeto BookRepository para obter informações sobre livros. Para escrever um teste para a classe BookService, precisamos de um objeto BookRepository que podemos controlar. Podemos usar um mock para criar um objeto BookRepository que podemos controlar.

O teste verifica se o método getTituloLivro() da classe BookService retorna o título correto do livro. O teste começa criando um mock para a classe BookRepository. Em seguida, o teste cria uma instância da classe BookService e passa o mock para o construtor.

O teste então configura o mock para retornar um objeto Book com o título "O Pequeno Príncipe" quando o método getBookByTitulo() for chamado com o argumento "O Pequeno Príncipe".

Finalmente, o teste chama o método getTituloLivro() com o argumento "O Pequeno Príncipe" e verifica se o retorno é igual a "O Pequeno Príncipe".

Passo a passo do que acontece no teste:

1. O teste começa criando um mock para a classe BookRepository.

2. O teste cria uma instância da classe BookService e passa o mock para o construtor.

3. O teste configura o mock para retornar um objeto Book com o título "O Pequeno Príncipe" quando o método getBookByTitulo() for chamado com o argumento "O Pequeno Príncipe".

4. O teste chama o método getTituloLivro() com o argumento "O Pequeno Príncipe".

5. O teste verifica se o retorno é igual a "O Pequeno Príncipe".

6. O teste passa porque o método getTituloLivro() retorna o título correto do livro.

Os mocks são uma ferramenta poderosa que pode ser usada para escrever testes de unidade eficientes e eficazes. Eles podem ser usados para substituir objetos reais que são difíceis ou impossíveis de testar diretamente.