ATIVIDADE:: Demais Dublês de Teste

Dummy Object , Test Spy e Fake Object como tipos de Dublês de Teste, mas não fornece suas definições, pois a tarefa é uma atividade de pesquisa.

Conforme solicitado pela atividade, realizei a pesquisa. Abaixo estão os resultados:

1. Definições Breves

- Dummy Object (Objeto Fictício): É o tipo mais simples de dublê. Ele é passado como parâmetro para um método, mas nunca é realmente utilizado. Dummies são usados apenas para satisfazer assinaturas de métodos (listas de parâmetros) para que o código possa ser compilado e executado, mas eles não têm nenhuma implementação funcional e não afetam o teste.
- Fake Object (Objeto Falso): É um objeto que substitui o componente real, mas possui uma implementação funcional simplificada. Diferente de Stubs ou Mocks, um Fake não é focado em retornar valores pré-configurados para um teste específico, mas sim em fornecer uma alternativa funcional mais leve. Um exemplo clássico é um banco de dados em memória (como o H2) substituindo um banco de dados de produção (como Oracle ou MySQL) durante os testes.
- Test Spy (Espião de Teste): É um dublê que age como o componente real (muitas vezes envolvendo a própria classe real), mas secretamente registra informações sobre como ele foi utilizado. Após o SUT (Sistema Sob Teste) ser exercitado, o teste pode interrogar o Spy para verificar se os métodos corretos foram chamados, quantas vezes foram chamados ou quais argumentos foram passados. Ele é uma mistura de Stub e Mock: pode fornecer dados (como um Stub) e verificar o comportamento (como um Mock).

ATIVIDADE:: Demais Dublês de Teste

2. Principais Diferenças

A principal diferença entre os tipos de dublês reside em seu propósito e complexidade:

Tipo de Dublê	Propósito Principal	Implementação	Verifica Estado ou Comportamento?
Dummy	Apenas preencher parâmetros	Objeto vazio, sem implementação (ex: null ou new Object())	Nenhum
Fake	Substituir um componente complexo por uma versão mais simples e funcional	Implementação real, porém leve (ex: banco em memória)	Estado (o teste verifica os resultados no Fake)
Stub	Fornecer respostas pré-definidas (entradas indiretas) para o SUT	Implementação simples que retorna valores fixos	Estado (o teste verifica o SUT)
Spy	Registrar como o SUT interagiu com ele (saídas indiretas)	Envolve o objeto real ou o simula, adicionando lógica de registro	Comportamento (o teste verifica o Spy)
Mock	Verificar se o SUT o utilizou da maneira esperada (saídas indiretas)	Objeto gerado (geralmente por um framework) que contém expectativas pré-definidas	Comportamento (o Mock verifica a si mesmo)

Diferenças Chave:

- Dummy vs. Outros: Dummies são apenas "espaços reservados" e não têm lógica.
- Fake vs. Stub/Mock: Fakes têm lógica de negócios real (embora simplificada), enquanto Stubs e Mocks são específicos para um cenário de teste e apenas simulam respostas ou comportamentos.
- Spy vs. Mock: Ambos verificam o comportamento (saídas indiretas). A diferença é sutil:
    - Um Mock é configurado *antes* da ação (fase de *Setup*) com as expectativas do que *deve* acontecer. Se o SUT não se comportar como esperado, o próprio Mock falha no teste (fase de *Verify*).

- Um Spy *apenas registra* o que aconteceu. O *próprio caso de teste* é responsável por verificar (usando *asserts*) se o Spy registrou as interações corretas *depois* da ação (fase de *Verify*).

3. Exemplos Práticos

Dummy Object: Suponha um método que precise de um objeto Logger, mas o teste atual não verifica o log, apenas o cálculo de retorno.

```
// SUT
public class Calculadora {
  // O logger é obrigatório, mas não usado neste método específico
  public int somar(int a, int b, Logger logger) {
    return a + b;
  }
}

// Teste
@Test
public void testSomar() {
  Calculadora sut = new Calculadora();
  Logger dummyLogger = null; // Um 'null' pode agir como um Dummy
  // ou: Logger dummyLogger = new DummyLogger();

  int resultado = sut.somar(2, 3, dummyLogger);

  assertEquals(5, resultado);
  // Ninguém verifica o 'dummyLogger'
}
```

Fake Object: Para testar um serviço (
AccountService) que depende de um repositório (AccountManager), em vez de usar o AccountManager real que acessa um banco de dados, usamos um FakeAccountManager que armazena os dados em um HashMap.

```
// O documento chama isso de "MockAccountManager",
// mas pela definição de "Fake", este exemplo se encaixa melhor:
public class FakeAccountManager implements AccountManager {
```

```
    private Map<String, Account> accounts = new HashMap<>();

    public void addAccount(String userId, Account account) {
        this.accounts.put(userId, account);
    }

    public Account findAccountForUser(String userId) {
        return this.accounts.get(userId);
    }

    public void updateAccount(Account account) {
        // Lógica real de atualização no HashMap
        this.accounts.put(account.getAccountId(), account);
    }
}

// Teste
@Test
public void testTransferenciaComFake() {
    Account sender = new Account("1", 200);
    Account beneficiary = new Account("2", 100);

    // O Fake tem uma lógica interna real (usando um Map)
    FakeAccountManager fakeManager = new FakeAccountManager();
    fakeManager.addAccount("1", sender);
    fakeManager.addAccount("2", beneficiary);

    AccountService sut = new AccountService();
    sut.setAccountManager(fakeManager);

    sut.transfer("1", "2", 50);

    // Verificamos o estado final dentro do Fake
    assertEquals(150, fakeManager.findAccountForUser("1").getBalance());
}
```

ATIVIDADE:: Demais Dublês de Teste

Test Spy: Queremos testar se um serviço de e-mail (NotificationService) chama corretamente o método send de um cliente de e-mail (EmailClient), e queremos verificar *exatamente* com qual e-mail e mensagem ele foi chamado.

```
// Spy (Espião)
public class EmailClientSpy implements EmailClient {
  public int chamadasAoSend = 0;
  public String ultimoEmailChamado;
  public String ultimaMensagemChamada;

  @Override
  public void send(String email, String message) {
    // O Spy registra as interações
    this.chamadasAoSend++;
    this.ultimoEmailChamado = email;
    this.ultimaMensagemChamada = message;

    // Ele também pode (opcionalmente) chamar a classe real
    // super.send(email, message);
  }
}

// Teste
@Test
public void testNotificacao() {
  EmailClientSpy spy = new EmailClientSpy();
  NotificationService sut = new NotificationService(spy);

  // Exercise SUT
  sut.notificarUsuario("usuario@teste.com", "Bem-vindo!");

  // Verify (Verificação manual feita pelo teste no Spy)
  assertEquals(1, spy.chamadasAoSend);
  assertEquals("usuario@teste.com", spy.ultimoEmailChamado);
  assertEquals("Bem-vindo!", spy.ultimaMensagemChamada);
}
```