Objetos Dublês

Mariana Bravo
marivb@agilcoop.org.br

AgilCoop – Cursos de Verão 2009

AgilCoop

Motivação

- Testes de unidade focam em **uma unidade**
- Mas as unidades do sistema têm dependências entre si

- Dublês oferecem uma maneira de isolar as dependências

Dependências

- **Entrada indireta** – dados que a unidade de teste obtém de algum objeto do qual ela depende (atributo de instância, parâmetro, etc).

- **Saída indireta** – resultados esperados da unidade de teste que não podem ser conferidos apenas pelo valor de retorno. Efeitos colaterais.

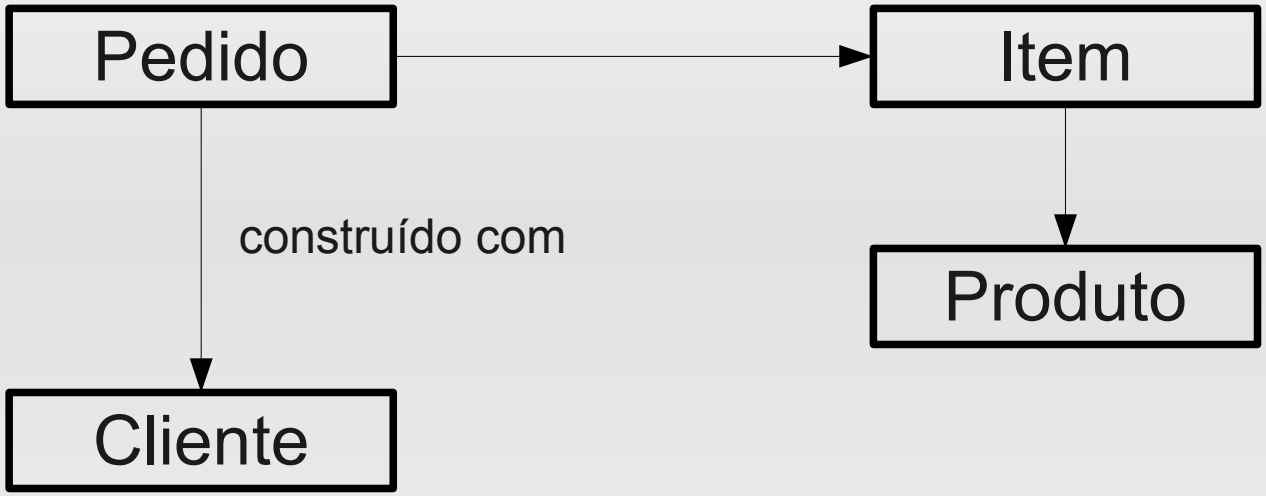
Definição

- Um **dublê** de teste é uma especialização de alguma **componente** da qual a **unidade** testada depende

Tipos de dublês

- *Dummy object*: permite a criação/execução do teste
- *Test stub*: provê informações para a unidade sob teste
- *Test spy*: captura e armazena chamadas indiretas
- *Mock object*: verifica chamadas indiretas e provê dados
- *Fake object*: contém uma implementação falsa de um componente real

Exemplo – Dummy Object



- No teste do Pedido, quero verificar que a adição de um produto funciona:
  - ```
    pedido.adicionaQuantidade(produto, 1);
    assertEquals(1, pedido.getItens().size());
    assertEquals(itemEsperado, itens.get(0));
    ```
- Para criar o Pedido preciso de um Cliente, mas o Cliente não é usado para adição de Produto
    ```

Dummy Object

Permite a realização do teste por satisfazer alguma chamada

Mas **não é usado** em nenhum momento pela unidade sob teste

Exemplo – Test Stub

- A classe FormatadorDeTempo provê a hora atual em diversos formatos.

```
public String horaAtualPorEscrito() {
  Calendar agora = Calendar.getInstance();
  if (agora.get(Calendar.HOUR_OF_DAY) == 0)
    return "Meia-noite";
  else if (agora.get(Calendar.HOUR_OF_DAY) == 12)
    return "Meio-dia";
  else
    return agora.get(Calendar.HOUR_OF_DAY) + "
horas";
}
```

- Como testar esse método?

Test Stub

Permite controlar a entrada indireta da unidade testada, **fornecendo dados** que poderiam ser difíceis de obter com o componente real

Exemplo – Test Stub Sabotador

- Algumas vezes queremos testar fluxos excepcionais do programa.
- Consideremos o código:

```
public void leInformacao(Reader reader) {
  try {
    while (reader.read() != 0) {
      // Processa informação lida
      // e armazena de alguma forma
    }
  } catch (IOException e) {
    // Trata exceção
  }
}
```

- Como testar se o tratamento da exceção está correto?

Exemplo – Test Spy

- Continuando o exemplo anterior:

```
catch (IOException e) {
  // Trata exceção
  logger.log("Erro lendo arquivo", e);
}
```

- Como fazer para testar que o erro é logado corretamente?

Test Spy

Captura informações sobre os efeitos colaterais provocados pela unidade sob teste, para que o teste verifique se estão corretos

Outro Exemplo – Test Spy

- Tenho uma classe Desenho que deve notificar observadores quando ela é modificada

```
public void adicionaFigura(Figura figura) {
   figuras.add(figura);
   setChanged();
   notifyObservers(figura);
}
```

- Como testar?

Exemplo – Mock Object

- Mesmo exemplo anterior! Mas outra maneira de testar.

Mock Object

Verifica diretamente os efeitos colaterais causados pela unidade sob teste

Test Spy vs. Mock Object

- Test Spy: *verificação de comportamento efetuado (tudo roda)*
  - A unidade sob teste é chamada
  - O spy captura informações
  - O teste verifica se elas estão corretas
- Mock Object: *especificação do comportamento esperado (falha cedo)*
  - O mock é carregado com as chamadas esperadas
  - A unidade sob teste é chamada – se algo der errado os testes falham

Tipos de Mock Object

- Estrito:
  Espera as mesmas chamadas exatamente na mesma ordem que especificado

- Tolerante:
  Aceita qualquer ordem das chamadas, inclusive com chamadas a mais ou a menos

Exemplo com EasyMock

- A partir do exemplo anterior!

Exemplo – Fake Object

- BufferedReader do Java lê linha por linha. Queremos um Reader especial que leia um parágrafo inteiro (sequência de linhas não vazias).

```
public class ReaderEspecial extends BufferedReader {

  public MeuReaderEspecial(Reader in) { super(in); }
  public String readLine() throws IOException;
  public String readParagraph() throws IOException;
  public boolean isOver() throws IOException;
}
```

- Como testar sem precisar ler um arquivo?

Fake Object

Substitui funcionalidade real com uma implementação alternativa.

Emula o comportamento do componente real, com características amigáveis ao teste.

Não é controlado nem observado pelo teste.

Fake Object - Motivos

- Componente real ainda não existe
- Componente real não está disponível no ambiente de teste
- Componente real é muito demorado
- Componente real causa efeitos colaterais indesejáveis

Instalando o dublê

- Uma vez que o dublê foi instanciado e configurado, precisamos fazer a unidade sob teste usá-lo.

- Existem diversas maneiras de fazer isso, a escolha pode depender de gosto ou de necessidade.

Injeção de dependência

- O cliente (ou teste) fornece à unidade sob teste o componente do qual ela depende.
  - Por um setter
  - Por um parâmetro
  - Pelo construtor

Lookup de dependência

- A unidade sob teste pede a algum componente que crie ou encontre o objeto do qual ela precisa.
  - Fábrica de objetos: cria o componente real, é informada pelo teste se deve devolver o componente de teste
  - Service locator: obtém de alguma forma o objeto a ser utilizado, pode ser configurado pelos testes para devolver o dublê desejado

Vantagens de usar dublês

- Isola testes de unidade (*bug* em uma unidade não afeta testes da outra)
- Acelera preparação ou execução dos testes
- Permite testar mesmo que alguma componente não esteja pronta ou disponível
- Evita efeitos colaterais indesejáveis

Fim

Perguntas?