

Objetos Dublês



AgilCoop – Cursos de Verão 2010

Mariana Bravo
IME/USP

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

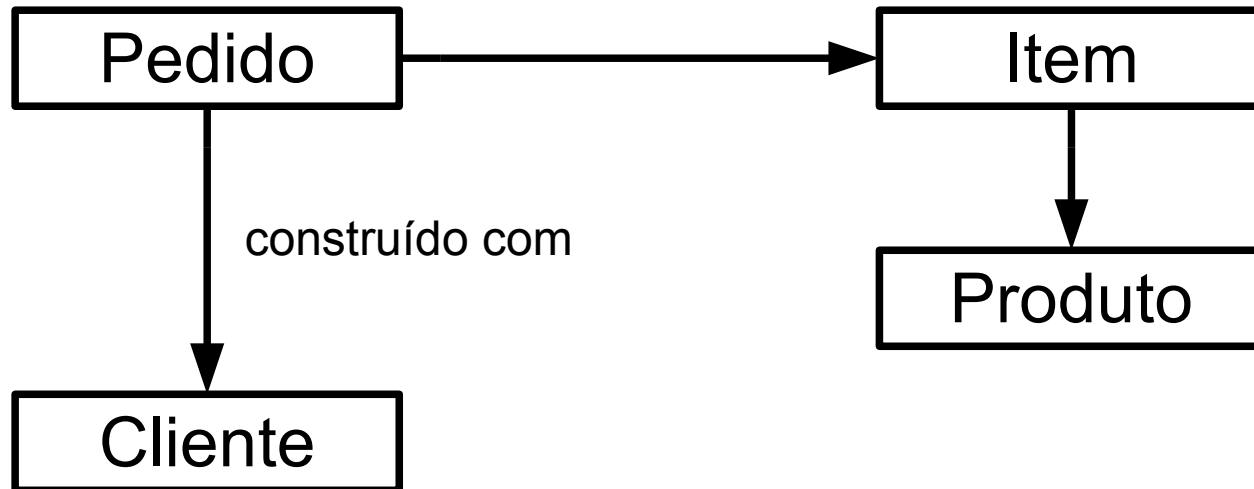
Dependências

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

Exemplo 1



- No teste do Pedido, quero verificar que a adição de produtos funciona
- Para criar o pedido preciso de um Cliente, mas o Cliente não é usado na adição de produtos

Bobo – *Dummy object*

Permite a realização do teste por satisfazer alguma chamada

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

Exemplo 2

- A classe `FormatadorDeTempo` provê a hora 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";  
}
```

Substituto – *Test stub*

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

Exemplo 3 – Substituto sabotador

- Algumas vezes queremos testar fluxos excepcionais do programa.

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

Exemplo 4 – Continuando...

```
public void leInformacao(Reader reader) {  
    try {  
        while (reader.read() != 0) {  
            // Processa informação lida  
            // e armazena de alguma forma  
        }  
    } catch (IOException e) {  
        // Trata exceção  
        logger.log("Erro lendo arquivo", e);  
    }  
}
```

- Como testar que o erro é gravado corretamente?

Espião – *Test spy*

Captura informações
sobre os efeitos colaterais
provodados pela unidade testada,
para que o teste verifique se estão corretos.

Exemplo 5 – Ainda o Espião

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

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

Exemplo 6 – Mesmo problema...

- Outra solução!

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

Imitação – *Mock object*

Verifica diretamente
os efeitos colaterais
causados pela unidade testada.

Peraí – Qual é a diferença?

- **Espião:** verificação de comportamento efetuado
 - A unidade sob teste é chamada
 - O espião captura informações
 - O teste verifica se elas estão corretas
- **Imitação:** especificação do comportamento esperado
 - A imitação é carregada com as chamadas esperadas
 - A unidade sob teste é chamada – se algo der errado, os testes falham

Peraí – Qual é a diferença?

- **Espião:** verificação de comportamento efetuado
 - A unidade sob teste é chamada
 - O espião captura informações
 - O teste verifica se elas estão corretas
- **Imitação:** especificação do comportamento esperado
 - A imitação é carregada com as chamadas esperadas
 - A unidade sob teste é chamada – se algo der errado, os testes falham

Roda até o fim

Falha cedo

Tipos de Imitação

- **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 7 – Agora com EasyMock

- A partir do exemplo anterior!

Exemplo 8

- O `BufferedReader` do Java lê um *stream* linha por linha. Temos um `Reader` especial que lê um parágrafo inteiro.

```
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 testá-lo sem precisar usar um *stream* real?

Falso – *Fake object*

Substitui uma funcionalidade real por uma implementação alternativa.

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

Não é controlado nem observado pelo teste.

Falso – 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

Procura 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
 - **Localizador de serviços:** 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)

Vantagens de usar dublês

Acelera preparação
ou execução dos testes

Vantagens de usar dublês

Permite testar
mesmo que alguma componente
não esteja pronta ou disponível

Vantagens de usar duplês

Evita efeitos colaterais indesejáveis

Perguntas?

Mariana Bravo
marivb@agilcoop.org.br

Referências

- Conteúdo na *web*:
 - <http://www.mockobjects.com/files/endotesting.pdf>
 - <http://www.mockobjects.com/files/mockrolesnotobjects.pdf>
 - <http://xunitpatterns.com/Test%20Double%20Patterns.html>
 - <http://www.martinfowler.com/articles/mocksArentStubs.html>

Referências

- Algumas ferramentas:
 - Java: Mockito - <http://mockito.org>, EasyMock - <http://easymock.org>
 - Ruby: RSpec - <http://rspec.info/documentation>, Mocha - <http://mocha.rubyforge.org/>
 - .NET: NMock - <http://www.nmock.org/>, RhinoMocks - <http://www.ayende.com/projects/rhino-mocks.aspx>
 - C++: Google Mock - <http://code.google.com/p/googlemock/>