7

J820

Mock objects
Testes de código com dependências

Helder da Rocha (helder@acm.org)

argonavis.com.br

Como lidar com testes difíceis

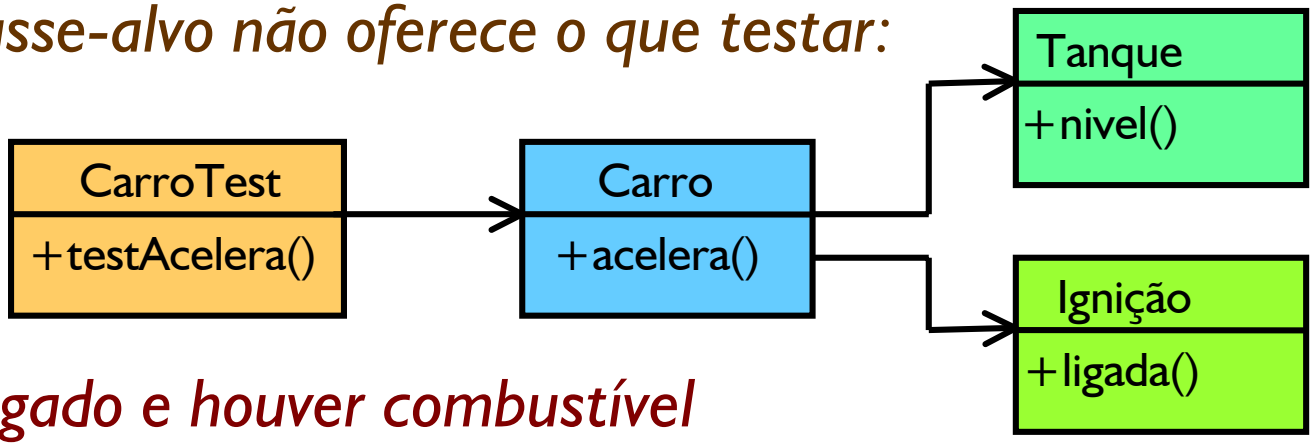
- Testes devem ser simples e suficientes
  - Comece com testes mais importantes
  - Sempre pode-se escrever novos testes, quando necessário
- Não complique
  - Não teste o que é responsabilidade de outra classe/método
  - Assuma que outras classes e métodos funcionam
- Testes difíceis (ou que parecem difíceis)
  - Aplicações gráficas: eventos, layouts, threads
  - Objetos inacessíveis, métodos privativos, Singletons
  - Objetos que dependem de outros objetos
  - Objetos cujo estado varia devido a fatores imprevisíveis
- Soluções
  - Alterar o design da aplicação para facilitar os testes
  - Simular dependências usando proxies e stubs

Como testar GUIs

- O que testar?
  - Assumir que GUI (Swing, AWT, etc.) funciona
  - Concentrar-se na lógica de negócio e não na UI
- Como testar?
  - "Emagrecer" o código para reduzir a chance de falha
  - Usar MVC: isolar lógica de apresentação e controle
  - Separar código confiável do código que pode falhar
  - Usar mediadores (Mediator pattern) para intermediar interações entre componentes
- Exemplo: event handlers
  - Devem ter 0% de lógica de negócio: "A Stupid GUI is an Unbreakable GUI" (Robert Koss, Object Mentor)
  - Responsabilidades delegadas a mediadores testáveis

Dependência de código-fonte

- *Problema*
  - *Como testar componente que depende do código de outros componentes?*
  - *Classe-alvo não oferece o que testar:*



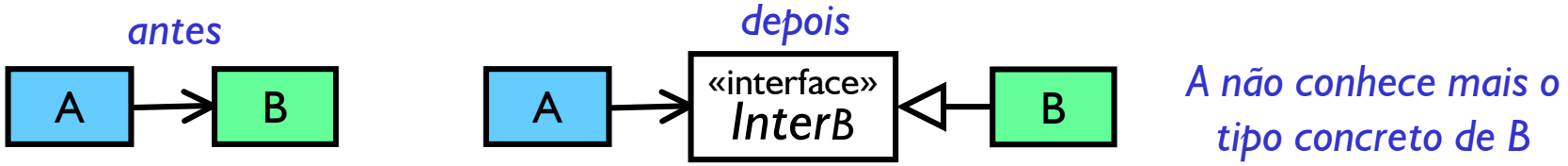
- *Se ligado e houver combustível método* **`void acelera()`** *deve funcionar*
- *Como saber?*

```
public void testAcelera() {
    Carro carro =
                new Carro();
    carro.acelera();
    assert???(???);
}
```

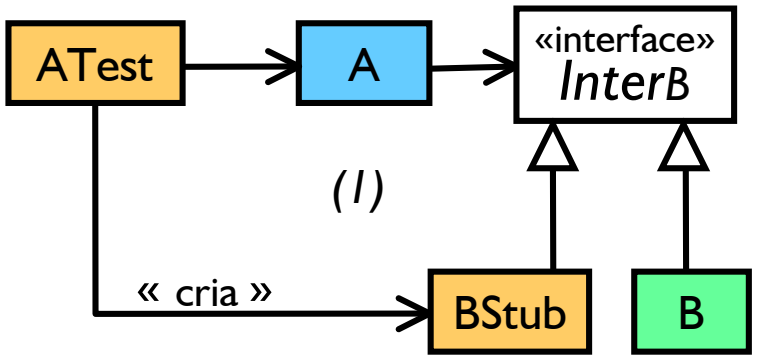
Fonte: www.objectmentor.com, 2002

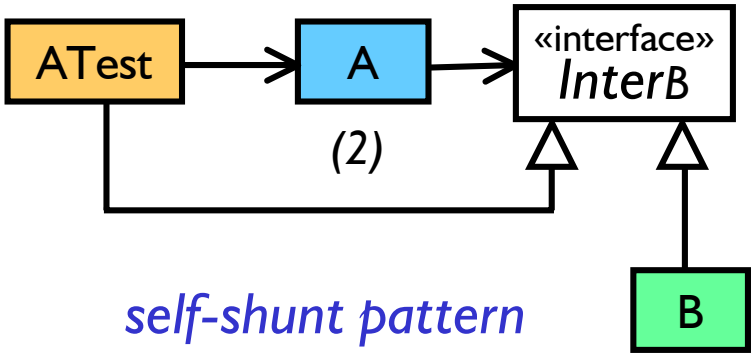
Stubs: objetos "impostores"

- *É possível remover dependências de código-fonte refatorando o código para usar interfaces*



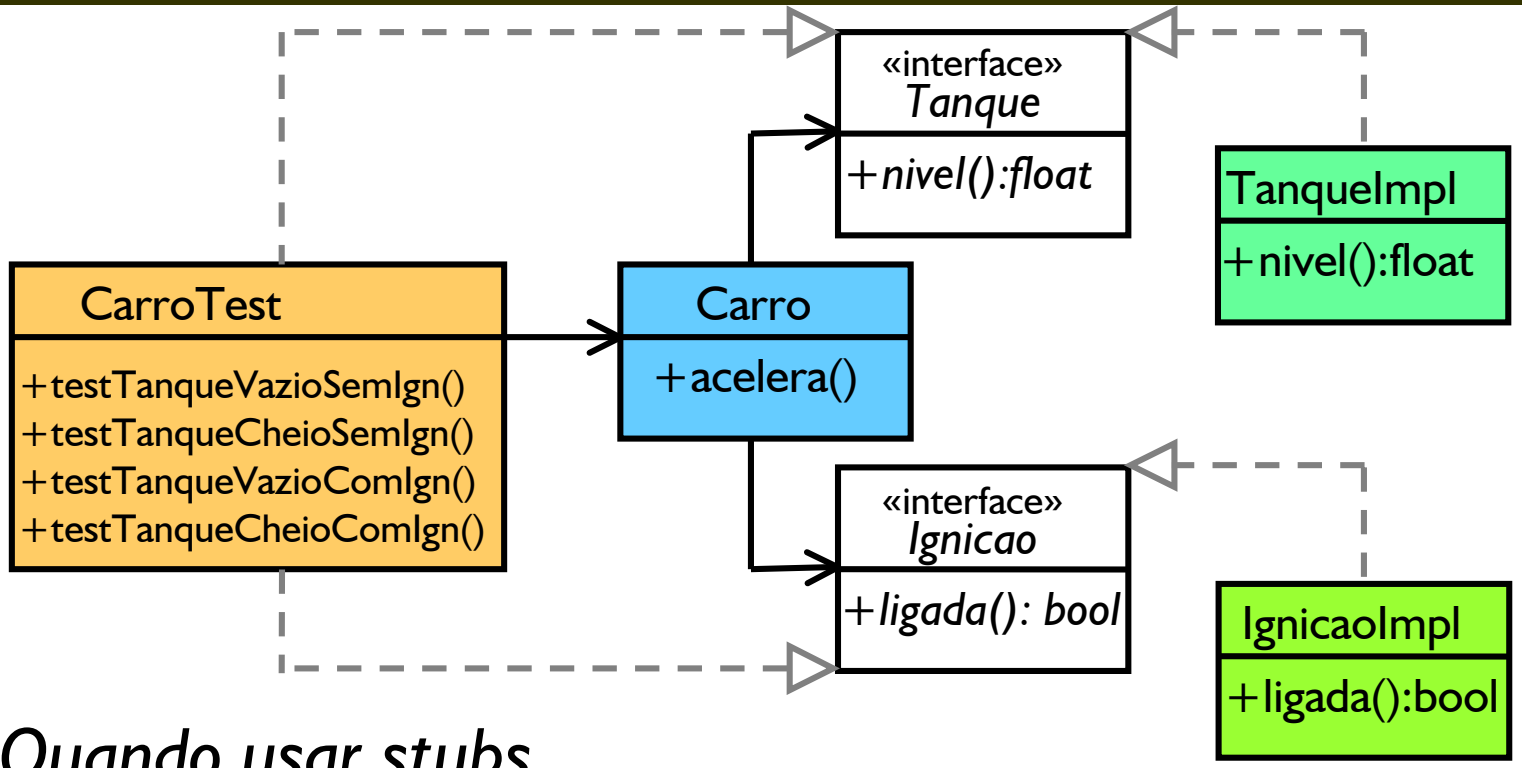
- *Agora B pode ser substituída por um stub*
  - *BStub está sob controle total de ATest (1)*
  - *Em alguns casos, ATest pode implementar InterB (2)*





Fonte: www.objectmentor.com, 2002

Dependência: solução usando stubs



- *Quando usar stubs*
  - *Dependência não existe ainda (não está pronta)*
  - *Dependências tem estado mutante, imprevisível ou estão indisponíveis durante o desenvolvimento*
    - *BDs, servidores de aplicação, servidores Web, hardware*

Fonte: www.objectmentor.com, 2002

Dependências de servidores

- Use **stubs** para simular serviços e dados
  - É preciso implementar classes que devolvam as respostas esperadas para diversas situações
  - Complexidade muito grande da dependência pode não compensar investimento (mas não deixe de fazer testes por causa disto!)
  - Vários tipos de stubs: mock objects, self-shunts.
- Use **proxies** (mediadores) para serviços reais
  - Oferecem interface para simular comunicação e testa a integração real do componente com seu ambiente
  - Não é teste unitário: teste pode falhar quando código está correto (se os fatores externos falharem)
  - Exemplo: Cactus, HttpUnit

Mock Objects

- **Mock objects** (MO) é uma estratégia similar ao uso de stubs mas que não implementa nenhuma lógica
  - Um mock object não é exatamente um stub, pois não simula o funcionamento do objeto em qualquer situação
- Comportamento é controlado pela classe de teste que
  - Define comportamento esperado (valores retornados, etc.)
  - Passa MO configurado para objeto a ser testado
  - Chama métodos do objeto (que usam o MO)
- Implementações open-source que facilitam uso de MOs
  - EasyMock (tammofreese.de/easymock/) e MockMaker (www.xpdeveloper.com) geram MOs a partir de interfaces
  - Projeto MO (mockobjects.sourceforge.net) coleção de mock objects e utilitários para usá-los

Exemplo de Mock Object

Interface

```
public interface Dependencia {
   public String metodo();
}
```

Implementação "mock" da interface gerada pelo mock maker

```
import mockmaker.ReturnValues;
import com.mockobjects.*;

public class MockDependencia implements Dependencia{
   private ExpectationCounter metodoCalls = new ... ;
   private ReturnValues metodoReturnValues = new ... ;
   public void setExpectedMetodoCalls(int calls){
      metodoCalls.setExpected(calls);
   }
   public void setupMetodo(boolean arg){
      metodoReturnValues.add(new Boolean(arg));
   }
   public void verify(){
      metodoCalls.verify();
   }
   public boolean metodo(){
      metodoCalls.inc();
      Object nextReturnValue = metodoReturnValues.getNext();
      return ((String) nextReturnValue).toString();
   }
}
```

métodos para configuração do mock object e resultados esperados

implementação

Usando um Mock Object

Objeto que usa Dependência. Em tempo de desenvolvimento, ela será implementada por um Mock object.

```
public class DepClient {
    private Dependencia dep;
    public DepClient(Dependencia dep) {
        this.dep = dep;
    }
    public String operacao(String texto) {
        return dep.metodo(texto);
    }
}
```

```
import junit.framework.*;

public class DepClientTest extends TestCase {
    private DepClient client;
    private MockDependencia mockObject; // implements Dependencia!

    public void setUp() {
        mockObject = new MockDependencia();
        client = new DepClient(mockObject);
        mockObject.setExpectedMetodoCalls(1);
        mockObject.setupMetodo("DADOS"); // define comportamento
    }

    public void testMetodo() throws java.io.IOException {
        String result = client.operacao("dados");
                assertEquals("DADOS", result);
        mockObject.verify();
    }
}
```

Test Case que usa um Mock Object para simular uma dependência real

Como testar bancos de dados

- Refatore seu código para que não seja necessário usar mock objects para JDBC
  - Forneça um DAO (Data-Access Object) que encapsule os métodos de acesso ao banco que sua aplicação irá utilizar
  - Escreva testes para a Interface do DAO
  - Escreva um stub para a interface do DAO ou gere um mock object para ela
- Outra alternativa é criar implementação de DAO que faça acesso a banco pequeno, facilmente configurável
  - Muitas vezes é mais fácil testar no próprio sistema que escrever os stubs e mock objects

Exercício

- 1. Gere um MockObject para testar a aplicação fornecida que depende de um DAO
  - Use o MockMaker
  - Escreva um test-case que utilize o mock object criado pelo mock maker.
  - Escreva um segundo teste (no mesmo test case) que use o DAO implementado (antes de executar, rode ant create-table para criar a tabela)

Fontes

[1] *Documentação XDoclet*. xdoclet.sourceforge.net

[2] Erik Hatcher. *Java Development with Ant*. Manning, 2003

[3] Eric Burke & Brian Coyner. *Java eXtreme Programming Cookbook*. O'Reilly, 2003

Curso J820
Produtividade e Qualidade em Java: Ferramentas e Metodologias

Revisão 1.1

(helder@acm.org)

argonavis.com.br