



MANUAL DE USO

EasyAccept: Uma ferramenta de testes automatizados



Programação II/Laboratório de Programação II - 2015.2

Monitores: Gleyser Guimarães e Paulo Vinícius Soares

Defeitos estão por toda parte!

Até no nosso projeto de P2/LP2! :o



E o que foi entregue, condiz com o que o cliente queria?

É importante que os nossos projetos **atendam aos desejos do cliente!**



Como o cliente explicou...



Como o líder de projeto entendeu...



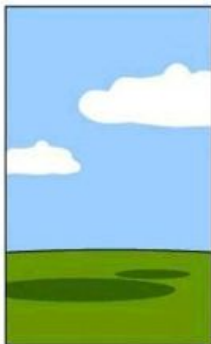
Como o analista projetou...



Como o programador construiu...



Como o Consultor de Negócios descreveu...



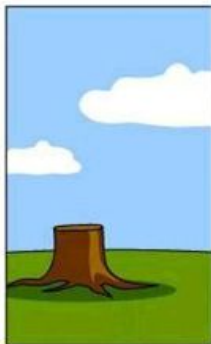
Como o projeto foi documentado...



Que funcionalidades foram instaladas...



Como o cliente foi cobrado...



Como foi mantido...



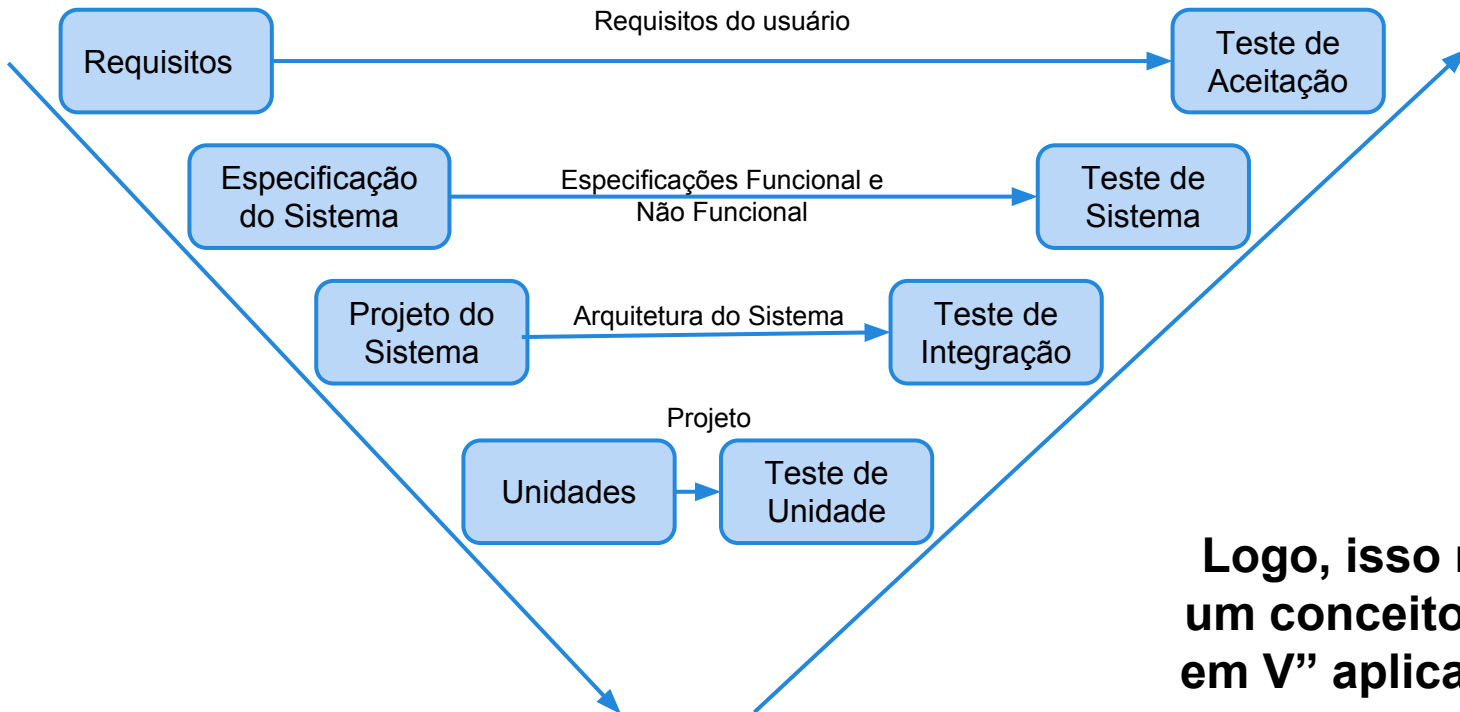
O que o cliente realmente queria...

Solução!? Ter um bom processo de testes!

Fase do processo de software

Processo de testes

Níveis de teste de software



Logo, isso não pode faltar! É um conceito de “programação em V” aplicado em seu projeto

Teste de Unidade e Teste de Aceitação

Fase do processo de software

Processo de testes

Níveis de teste de software

Requisitos do usuário

Requisitos

Teste de Aceitação

Especificação do Sistema

Especificações Funcional e Não Funcional

Teste de Sistema

Projeto do Sistema

Arquitetura do Sistema

Teste de Integração

Unidades

Projeto

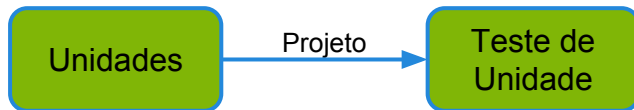
Teste de Unidade

Os testes devem, preferencialmente, ocorrer em **diferentes níveis** e em **paralelo** ao **desenvolvimento do software**.

Nesse sentido, atuaremos em dois níveis do processo de teste de software: **Teste de Unidade** e **Teste de Aceitação**.

Teste de Unidade

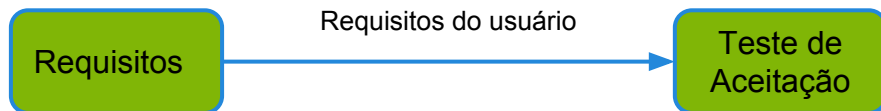
Teste de Unidade tem por objetivo **explorar a menor unidade do projeto**, procurando provocar falhas ocasionadas por **defeitos de lógica e de implementação** em cada módulo, separadamente. O universo alvo desse tipo de teste são os **métodos dos objetos** ou mesmo **pequenos trechos de código**. (ROCHA *et al.*, 2001)



Teste de Aceitação

Teste de Aceitação são realizados geralmente por um restrito grupo de usuários finais do sistema que **simulam operações de rotina do sistema** de modo a **verificar se seu comportamento está de acordo com o solicitado**. (ROCHA *et al.*, 2001)

Note que, os testes de aceitação são criados **a partir da especificação dos requisitos** ou da **especificação do projeto**.



Teste de Aceitação Automatizados

Os **testes de aceitação** podem ser realizados **de forma manual ou automática**. Utilizaremos testes automatizados por meio da ferramenta [EasyAccept](#) que é um framework desenvolvido em Java pela equipe de pesquisa do professor Dr. Jacques Philippe Sauvé (DSC/CEEI/UFCG).



EasyAccept

Arquitetura da Ferramenta

[EasyAccept,2011]



Instalando a ferramenta no Eclipse

Passo 1:

1. Abrir o Eclipse
2. Criar um novo projeto Java
3. Denominar “EasyAcceptProject”
4. Clicar em Finish

Instalando a ferramenta no Eclipse

Passo 2:

1. No projeto, criar a pasta “teste_aceitacao”. É nessa pasta que colocaremos os scripts de teste das estórias do usuário. Para iniciarmos, [adicione esse arquivo](#).
2. No projeto, criar um package denominado “monopoly”.
3. No projeto, criar uma Façade dentro do package monopoly, denominada “GameFacade”.

Instalando a ferramenta no Eclipse

Passo 3:

1. Na classe GameFacade.java, adicionar o método “main” abaixo, para executar o teste do EasyAccept.

```
public static void main(String[] args) {  
    args = new String[] { "monopoly.GameFacade",  
                           "teste_aceitacao/us1.txt" };  
    EasyAccept.main(args);  
}
```

Instalando a ferramenta no Eclipse

Passo 4:

1. Abra o arquivo “us1.txt” e verifique que todas as linhas estão comentadas.
2. Execute a classe, GameFacade. Caso a classe esteja com erro de compilação, adicionar:
`import easyaccept.EasyAccept;`

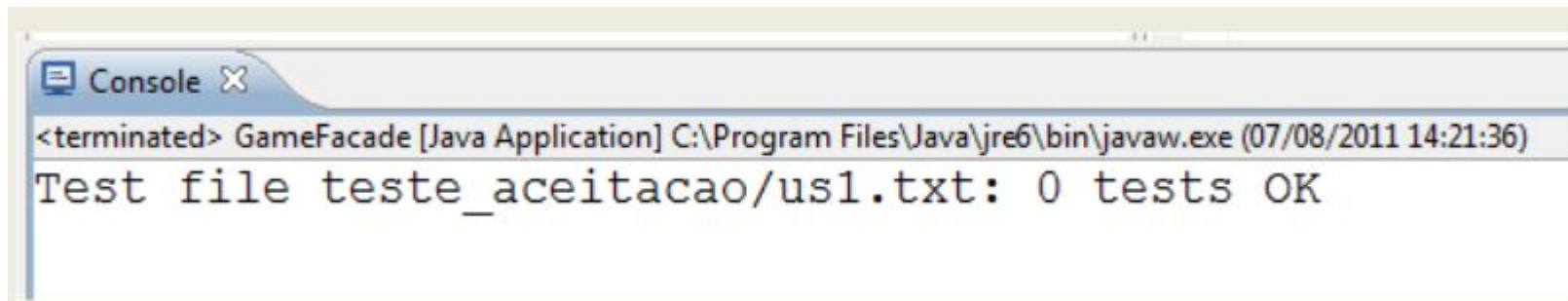
Instalando a ferramenta no Eclipse

Passo 5:

1. Após executar, o sistema abrirá o console com o texto:

Test file teste_aceitacao/us1.txt: 0 tests OK.

Isso indica que nenhum teste foi executado.



Instalando a ferramenta no Eclipse

Passo 6:

1. Abra o arquivo “us1.txt” e tirar o comentário do primeiro comando.

```
# USER STORY 1 - START A NEW GAME

#####
# Creating a game with two players #
#####

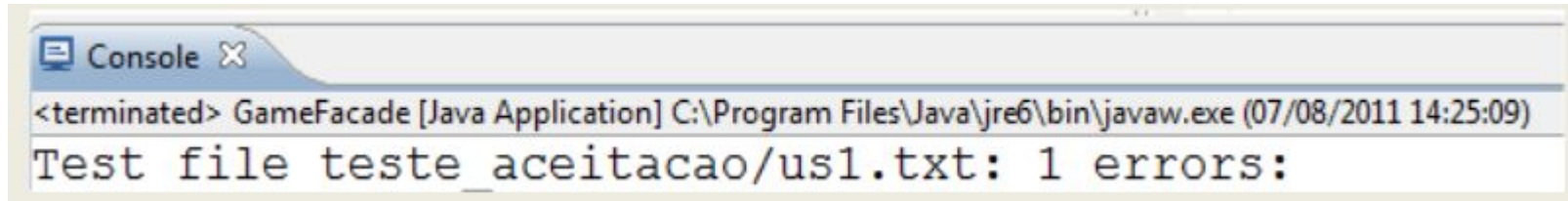
createGame numPlayers=2 playerNames={player1,player2} tokenColors={black,white}

#expect 2 getNumberOfPlayers
#expect "black" getPlayerToken playerName="player1"
#expect "white" getPlayerToken playerName="player2"
#expect 1500 getPlayerMoney playerName="player1"
#expect 1500 getPlayerMoney playerName="player2"
#expect 40 getPlayerPosition playerName="player1"
#expect 40 getPlayerPosition playerName="player2"
```

Instalando a ferramenta no Eclipse

Passo 7:

1. Executar novamente a classe GameFacade.java. Desta vez o teste apresentará 1 erro.

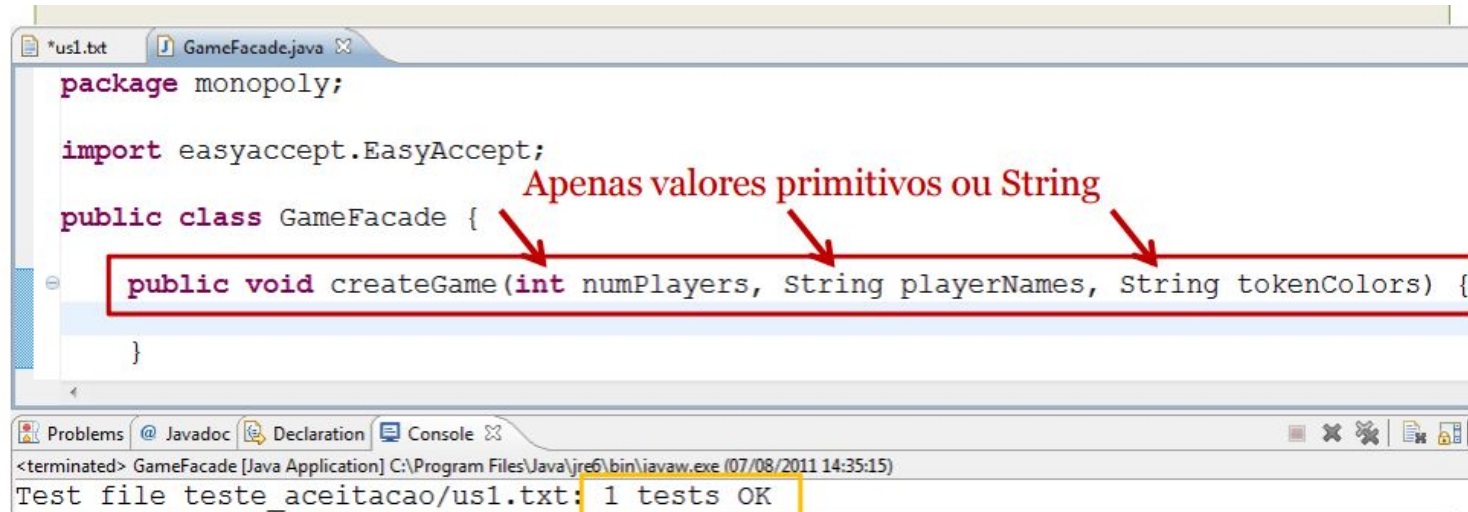
A screenshot of the Eclipse IDE's Console window. The window has a title bar with a document icon, the word 'Console', and a close button. The text inside the console shows the termination of a Java application and a test result. The text is: '<terminated> GameFacade [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (07/08/2011 14:25:09)' followed by 'Test file teste_aceitacao/us1.txt: 1 errors:'.

```
<terminated> GameFacade [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (07/08/2011 14:25:09)
Test file teste_aceitacao/us1.txt: 1 errors:
```


Instalando a ferramenta no Eclipse

Passo 7:

1. Para que o primeiro comando passe no teste basta implementar a assinatura do método para o primeiro comando.



```
package monopoly;

import easyaccept.EasyAccept;

public class GameFacade {

    public void createGame(int numPlayers, String playerNames, String tokenColors) {

    }

}
```

Apenas valores primitivos ou String

Test file teste_aceitacao/us1.txt: 1 tests OK

Pronto! Eclipse configurado!

O Eclipse já está **configurado** para trabalhar em conjunto com o EasyAccept.

Os próximos passos:

- Entender a **sintaxe** dos scripts de teste
- Entender os **scripts** iniciais do projeto
- Criar os seus **próprios testes**.

Introdução ao ambiente de testes

O ambiente de testes é composto por uma **façade** aliada à **scripts**, também chamados de **user stories**, que representam as interações do usuário com o sistema.

Mas, o que é uma Façade?



Introdução ao ambiente de testes:

Façade

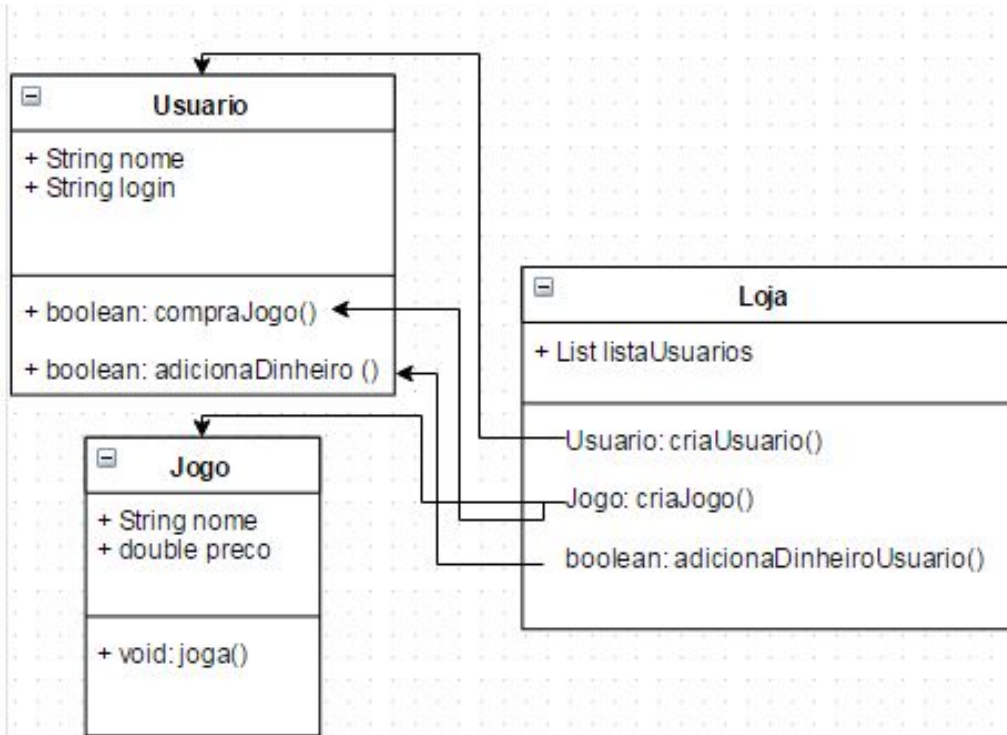
É uma fachada que simplifica uma interface, **ocultando detalhes** de implementação e **abstraindo** os métodos a um nível mais alto. Dessa forma, fornece um **ambiente unificado** para o conjunto de interfaces em um subsistema, facilitando a utilização deste.

Introdução ao ambiente de testes:

Façade

A classe “Loja” do LAB07 atuava como uma Façade intermediando ações entre usuários e jogos. **Métodos** como `criaUsuario()`, `criaJogo()` eram compostos por outros diversos métodos das classes `Usuario` e `Jogo`, porém isso era **transparente** à pessoa que fosse utilizar a classe “Loja”.

Introdução ao ambiente de testes: Façade



Representação extremamente simplificada da classe Loja, Usuario e Jogo para melhor compreensão

Introdução ao ambiente de testes:

Façade

- Perceba que dentro do método `criaJogo()` podem existir **diversas implementações** na forma de criar o jogo, mas todas o fazem.
- Por exemplo, poderia ser utilizado o padrão *Factory* dentro desse método, ou a implementação direta dos construtores das classes. Mas isso é invisível para quem está usando a loja. Isso é **ocultação de detalhes** e **abstração**.

Mas, qual a importância da façade para meu sistema?

Introdução ao ambiente de testes:

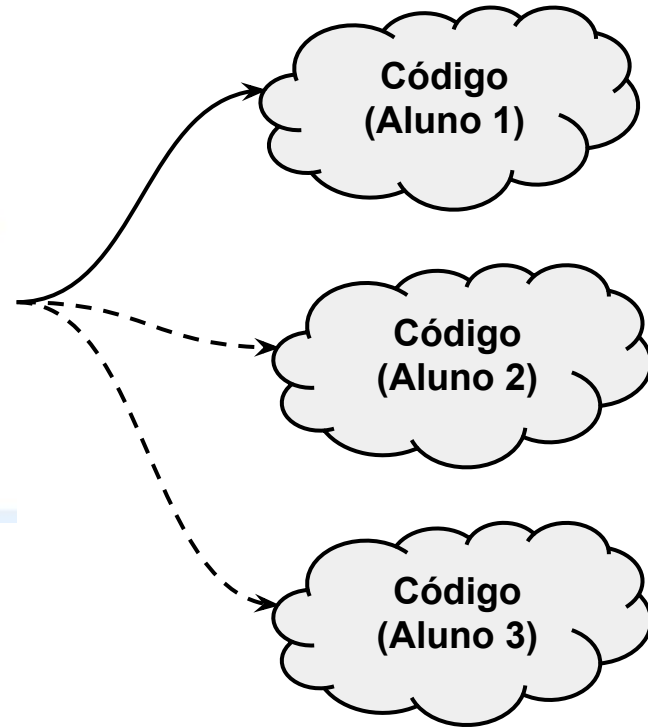
Façade

- As vantagens da Façade são:
 - Mudanças nas classes não afetam diretamente o cliente, ou **modificações** no cliente não afetam diretamente as classes. Isso é consequência da **redução do acoplamento**, gerando uma excelente solução para **manutenção do código**.
 - Aumento da **legibilidade** do código.
 - Redução na complexidade do sistema.

Introdução ao ambiente de testes:

Façade

```
public class Façade {  
    private Loja loja;  
  
    public void criaJogo(String nomeJogo, String tipoJogo, double preco){  
        try {  
            loja.criaJogo(nomeJogo, tipoJogo, preco);  
        } catch (Exception e) {  
            System.out.println(e.getMessage());  
        }  
    }  
}
```



Introdução ao ambiente de testes:

Façade

```
public class Façade {  
    private Loja loja;  
  
    public void criaJogo(String nomeJogo, String tipoJogo, double preco){  
        try {  
            loja.criaJogo(nomeJogo, tipoJogo, preco);  
        } catch (Exception e) {  
            System.out.println(e.getMessage());  
        }  
    }  
}
```

Exemplo:
Código de Vinícius

```
public Jogo criaJogo(String nomeJogo, String tipoJogo, double preco) throws Exception{  
    Jogo novoJogo;  
    if(tipoJogo.equals("RPG")){  
        novoJogo = new RPG(nomeJogo, preco);  
    }else if(tipoJogo.equals("Plataforma")){  
        novoJogo = new Plataforma(nomeJogo, preco);  
    }else if(tipoJogo.equals("Luta")){  
        novoJogo = new Luta(nomeJogo, preco);  
    }else{  
        throw new Exception("Tipo de jogo inválido.");  
    }  
    return novoJogo;  
}
```

The diagram illustrates the Facade pattern. On the left, the `public void criaJogo` method in the `Façade` class is shown. A curved arrow points from the `loja.criaJogo` call inside the `try` block to the `public Jogo criaJogo` method in the implementation class on the right. Below the `Exemplo: Código de Vinícius` text, a straight arrow points from the text to the implementation class code.

Introdução ao ambiente de testes: Façade

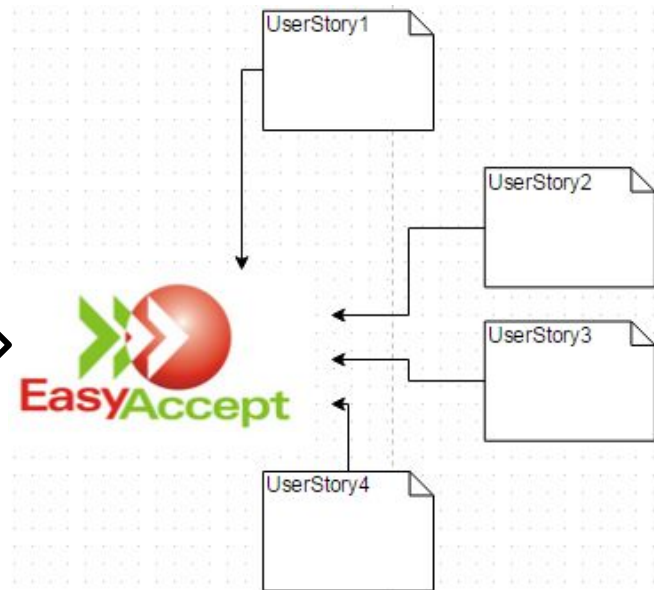
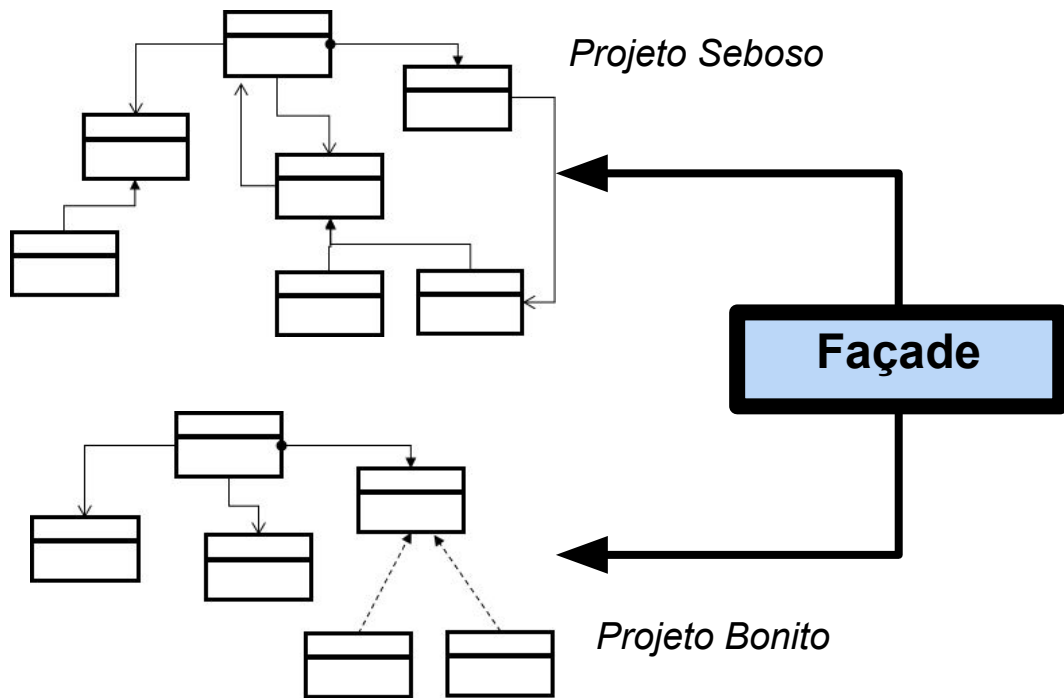
```
public class Façade {  
    private Loja loja;  
  
    public void criaJogo(String nomeJogo, String tipoJogo, double preco){  
        try {  
            loja.criaJogo(nomeJogo, tipoJogo, preco);  
        } catch (Exception e) {  
            System.out.println(e.getMessage());  
        }  
    }  
}
```

Exemplo:
Código de Gleyser

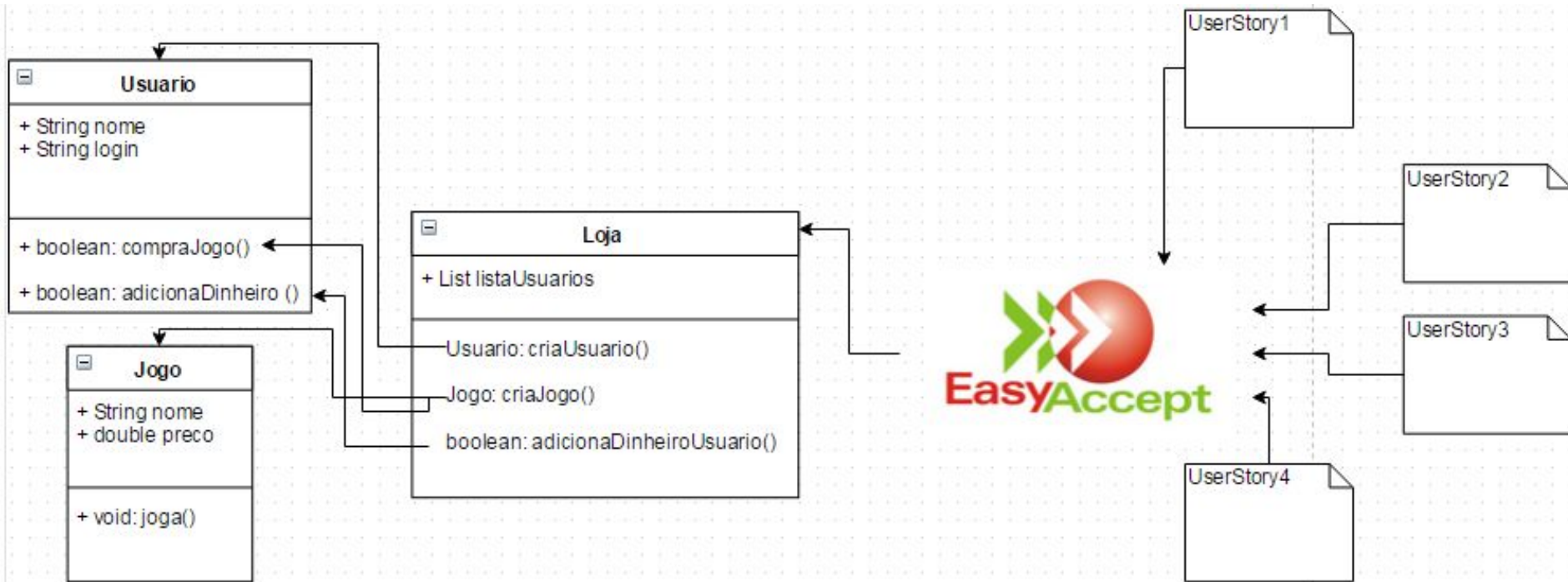
```
private JogoFactory fabricaJogos;  
  
public Jogo criaJogo(String nomeJogo, String tipoJogo, double preco) throws Exception{  
    //Uso do Factory.  
    return fabricaJogos.criaJogo(nomeJogo, tipoJogo, preco);  
}
```

Note que:
Façade não mudou,
apenas o código do Aluno.

Introdução ao ambiente de testes: Façade



Introdução ao ambiente de testes: Façade



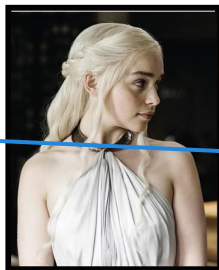
* A loja está atuando como façade

Mas, o que eu preciso testar no meu projeto?

Daenarys Targaryen → Nome de usuário

Lista de amigos

Publicações



A mother does not flee without
her children.

#dragons #motherhood #epicwin

14/06 16:00

Curtir (5) Rejeitar (3)

Exibição do Ranking

Ranking

- (1) Alan Turing
- (2) Barack Obama
- (3) Sheldon Cooper

- (3) Cersei Lannister
- (2) Hitler
- (1) Nazare Tedesco

Trending Topics

Barack Obama

Good morning!
New day, new ideas.

#newlife

14/06 07:00

Curtir (67) Rejeitar (5)

Your voice can change
the world

#change #voice

13/06 15:38

Curtir (96) Rejeitar (2)

Amigos



Barack Obama



Tyrion Lannister



Walter White



Fátima Bernardes



Alan Turing



Nazaré Tedesco



Introdução ao ambiente de testes:

User Stories

- Vamos pensar na user story que levou o programa a exibir a tela exemplo do slide anterior. Além disso, vamos pensar agora usando a lógica de **testes de aceitação**, diferenciando da lógica de **testes de unidade**.
- Temos uma **usuária** chamada “Daena Targaryen”, com uma **lista de amigos** contendo Barack Obama, Fátima Bernardes, entre outros. Além disso temos uma **lista de postagens** exibida no centro da tela.

Introdução ao ambiente de testes:

User Stories

- Então, para que isso seja necessário, precisamos de uma **sequência de passos**:
 - Cadastro do usuário “Daena Targaryen”;
 - Adição dos amigos;
 - Postagem no mural;
- Cada item caracteriza uma **User Story**! É uma interação do usuário com o sistema a qual eu quero testar e cujo resultado desta é conhecido.

Introdução ao ambiente de testes:

User Stories

- Uma possível **caso de teste** seria:
 - Criação do usuário;
 - Efetuar o login;
 - Efetuar o logout;

Com isso, há de se testar se o programa retorna o resultado esperado.

Introdução ao ambiente de testes:

User Stories

Criar um usuário:

```
id1=createUser name="Amanda" email="amanda.n@email.com" password="senha" birthday="06/02/2000" image="resources/amanda.jpg"
```

Observe que não deve ser escrito um espaço a mais ou a menos no script, caso isso ocorra culminará em um erro de execução. O que está sendo dito ao programa é basicamente que um objeto foi criado com os seguintes parâmetros informados e sob identificação de “id1”. *CreateUser* é o método da fachada a ser testado e o restante são os parâmetros solicitados pelo mesmo como *name*, *email*, *password*, *birthday* e *image*.

Introdução ao ambiente de testes: User Stories

Criar um usuário:

Devemos ter em mente ainda que precisamos nos certificar da construção correta do usuário, dentro da lógica de sistema.

expect “Amanda” getName usuario=\${id1}

ATENÇÃO: Como o EasyAccept só trabalha com dados primitivos, o que será passado como parâmetro para o método pela chamada de *\${id1}* será o `toString()` do objeto. Portanto, **atenção** na escolha do `toString`.

Introdução ao ambiente de testes: Sintaxe

expect: Verifica se um método retorna o dado esperado.

The diagram illustrates the syntax of the `expect` command with five examples. Annotations with arrows point to specific parts of the syntax:

- método da façade**: Points to `getAtributoEmpregado` in the first example.
- id do objeto criado**: Points to `emp=${id1}` in the first example.
- parâmetro requerido pelo método**: Points to `atributo=nome` in the first example.
- retorno esperado**: Points to the values `"João da Silva"` and `horista` in the first and third examples, respectively.
- comando**: Points to the `expect` keyword in the first example.

```
expect "João da Silva" getAtributoEmpregado emp=${id1} atributo=nome
expect "Rua dos Joões, 333 - Campina Grande" getAtributoEmpregado emp=${id1} atributo=endereco
expect horista getAtributoEmpregado emp=${id1} atributo=tipo
expect 23,00 getAtributoEmpregado emp=${id1} atributo=salario
expect false getAtributoEmpregado emp=${id1} atributo=sindicalizado
```

LEMBRE-SE: A ferramenta só trabalha com tipos primitivos de dados. É importante ressaltar que há a possibilidade de trabalhar com strings simples sem a necessidade de aspas, porém, por questões de boas práticas, estas deverão ser utilizadas.

Introdução ao ambiente de testes: Sintaxe

echo: concatena dois parâmetros dentro de outro comando, geralmente vem aliado ao **expectDifferent** que verifica se um método produz um resultado diferente de um dado passado.

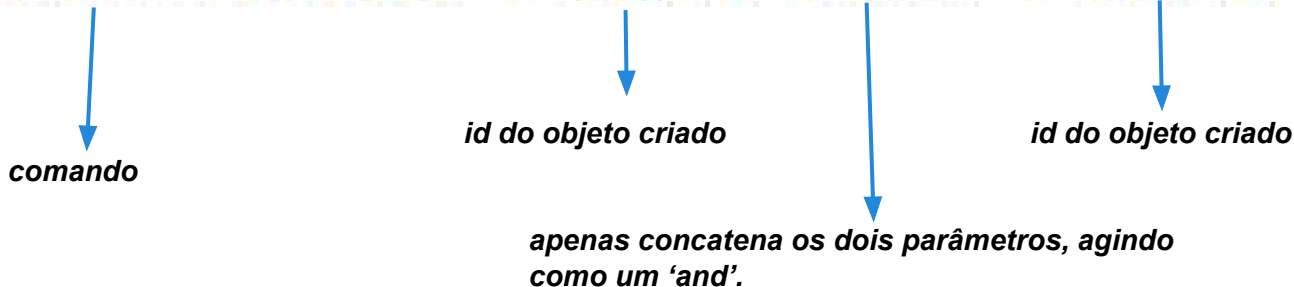
```
expectDifferent ${id1} echo ${id2}
```



comando



id do objeto criado



apenas concatena os dois parâmetros, agindo como um 'and'.



id do objeto criado

Introdução ao ambiente de testes:

Sintaxe

expectError: é usado para checar erros na lógica de negócios, garantindo que tal ação retorne uma mensagem reportando tal.

```
removerEmpregado emp=${id1}  
expectError "Empregado nao existe." removerEmpregado emp=${id1}  
expectError "Empregado nao existe." getAtributoEmpregado emp=${id1} atributo=nome  
expect "Maria" getAtributoEmpregado emp=${id2} atributo=nome
```

comando

retorno esperado

método da façade

id do objeto criado

*parâmetro
do método*

Introdução ao ambiente de testes: Sintaxe

expectWithin: é utilizado para comparar uma saída de ponto flutuante dado uma precisão específica.

```
createUser key=key1 name="John Doe" birthdate=1957/02/04
```

```
expect "John Doe" getUsername key=key1
```

```
expectWithin .01 2345.67 getSalary key=key1
```

comando

precisão

retorno esperado

método da façade

*parâmetro
do método*

Introdução ao ambiente de testes:

Sintaxe

equalFiles: compara dois arquivos a fim de verificar se a gravação de texto em um arquivo está como o esperado.

```
expect "John Doe" getUsername key=${key1}  
produceReport key=${key1} outputFile=rep.txt  
equalFiles file1=expected-report.txt file2=rep.txt
```

*Método que salva
informações em um .txt*

comando

arquivo modelo

*arquivo produzido
pelo método*

Introdução ao ambiente de testes:

Sintaxe

stackTrace: utilizado para mostrar um “stack trace”, ao debugar. É bastante útil ao se deparar com exceções inesperadas.

```
# the following command produces an exception  
stackTrace someCommand param=someValue
```



comando

método da façade

*parâmetro
do método*

Introdução ao ambiente de testes: Sintaxe

quit: utilizado ao fim de cada script para encerrar a ferramenta.

```
expectError "Atributo nao existe." getAtributoEmpregado emp=${id1} atributo=abc
```

```
encerrarSistema → método da façade
```

```
quit
```



comando

Referências

Façade

- FAÇADE Pattern. Disponível em: <https://en.wikipedia.org/wiki/Facade_pattern>. Acesso em: 16 jul. 2015.
- PADRÃO de projeto Façade em Java. Disponível em:<<http://www.devmedia.com.br/padrao-de-projeto-facade-em-java/26476>>. Acesso em: 16 jul. 2015.
- DESIGN patterns: Padrão Façade e Strategy. Disponível em:<<http://www.dainf.ct.utfpr.edu.br/~tacla/DesignPatterns/0050-JavaDP-FacadeSingletonStrategy.pdf>>. Acesso em: 16 jul. 2015.

EasyAccept

- EASYACCEPT – A Tool to Create Acceptance Tests in Java - User Manual. Disponível em: <<http://www.dsc.ufcg.edu.br/~jacques/projetos/common/easyaccept/usermanual.html>>. Acesso em 17 jul. 2015.
- EASYACCEPT Tutorial. Disponível em:<<http://easyaccept.sourceforge.net/tutorial.html>>. Acesso em 17 jul. 2015.
- TESTES de software - Ferramentas de automação de testes. Disponível em: <<http://www.nti.ufpb.br/~caroline/curso/Aula03-Curso%20de%20Testes%20de%20Software%20-%20NTI.pdf>>. Acesso em 17 jul. 2015.



Laboratório de Programação II - Programação Orientada à Objetos

Professores responsáveis:



Lívia Sampaio

Raquel Lopes

Francisco Neto