PET UFSC COMPUTAÇÃO

Universidade Federal de Santa Catarina
Programa Especial de Treinamento
Ciências da Computação

JUnit
Um framework para testes

"Nunca no campo do desenvolvimento de software se deveu tanto e por tantos para tão poucas linhas de código." *Martin Fowler

PET/CCO/UFSC

Daniel Borges Ribeiro

danielbr@inf.ufsc.br

* Martin Fowler é um astuto e pragmático defensor dos padrões da programação XP (e também autor do livro sobre modelos de objetos denominado Analysis Patterns).

O que é JUnit?

- *Framework* para facilitar o desenvolvimento e a execução de testes de unidade em programas Java.
- Unidade em Java são classes.

Por que testar o código?

Apesar de demorar mais no momento da implementação, você irá ganhar tempo na localização de erros a medida que o programa começa ficar muito grande.

"As vezes é preciso dar um passo pra trás agora, pra dar dez passos pra frente mais tarde."

Por que testar o código?

- Melhora a integração das camadas de um software(interface, lógica, persistência), já que permite que cada um que desenvolveu cada camada garanta que a sua está funcionando perfeitamente.

Porque usar JUnit?

- Permite separar o código fonte dos testes.
- Possui regras já definidas para apresentação de erros.
- Permite criar os testes antes mesmo de implementar o código. O que permite que você faça uma implementação dirigida por contrato.

API para construir testes

- junit.framework.*

Ex.:

- Assert
- TestCase - fornece ferramentas para rodar testes, gerar relatórios e definir conjuntos de testes.
- AssertionFailedError – ocorre quando os resultados do teste estiverem incorretos.

API Junit: http://junit.sourceforge.net/javadoc/

Principais asserções

- `assertEquals(objEsperado,objRecebido);`
- `assertTrue(expBooleana);`
- `assertNull(obj);`
- `assertNotNull(obj)`
- `assertSame(obj1, obj2);`
- `fail(mensagem);`

Um exemplo simples

```
public class Coisa{

    int n;

    public Coisa(int n){this.n = n;}
    public int add(int x){return n + x;}
    public int getValue(){return this.n;}
}
```

Classe Teste

```
public class TesteCoisa extends TestCase {

    public void testAdd() {

        Coisa s = new Coisa(500);
        assertEquals("Resultado da adio = ", 600, s.add(100));
        assertEquals("Valor Retornado = ", 600, s.getValue());
    }
    public void testGetValue() {
        Coisa s = new Coisa(100);
        assertEquals(100, s.getValue());
    }
}
```

Resultado

O teste encontra um erro em Valor Retornado:

Valor Retornado = expected:<600> but was:<500>

Onde está o erro?

O método: public int add(int x){return n + x;}
deveria ser: public int add(int x){return this.n = n + x;}

Ele esqueceu de atualizar a variável n. Perceba um erro bobo e quase imperceptivel, que seria difícil de perceber a medida que o código começar a ficar mais complexo.

Exemplo

```
public interface Quadrado {

    public int getArea();
    public int getPerimetro();
    public int getLado();
    public int getAltura();

}
```

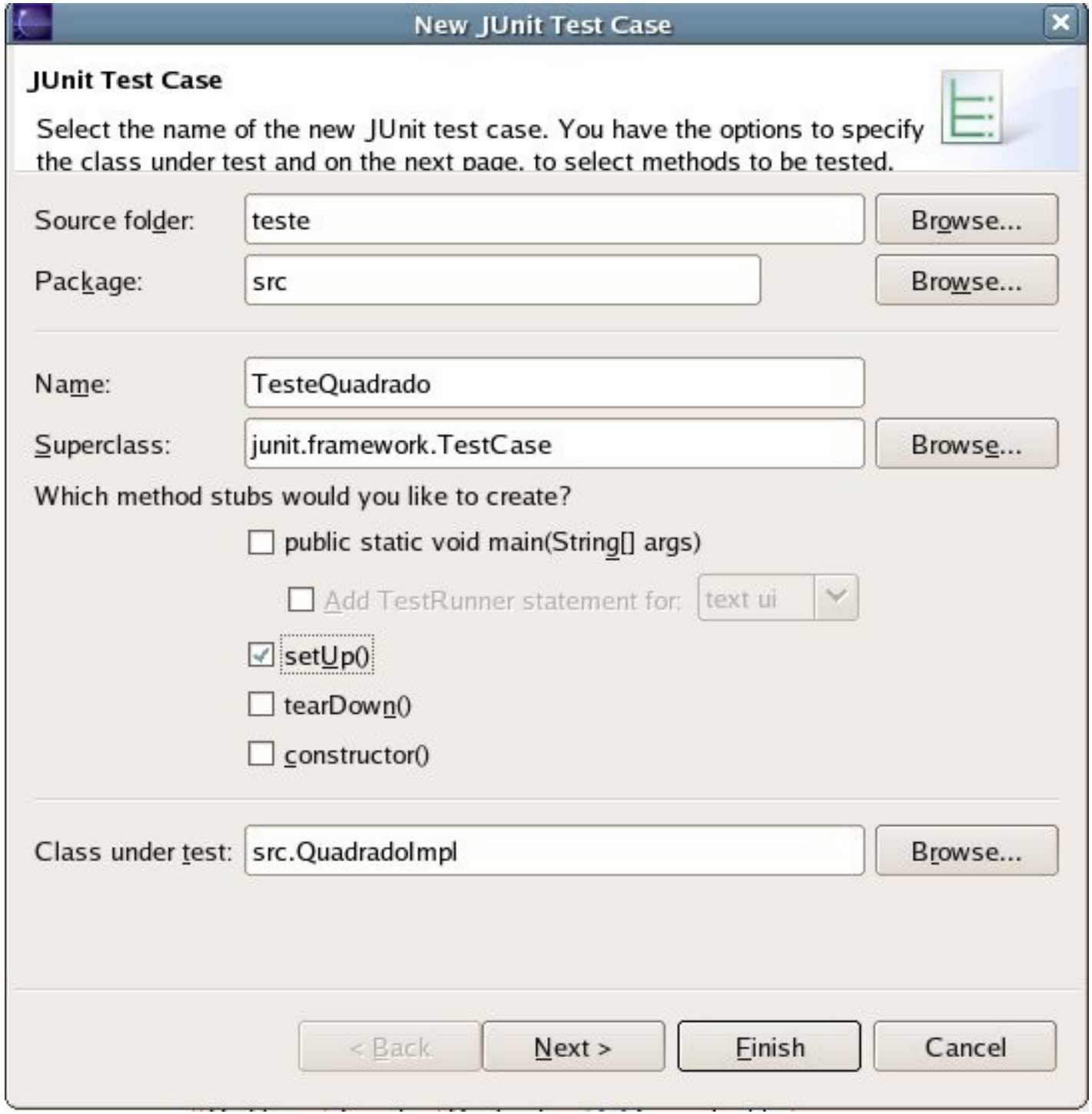
```
public class QuadradoImpl implements Quadrado {

    int l;
    int h;

    public QuadradoImpl(int l, int h){
        this.l = l;
        this.h = h;
    }
    public int getArea() {return l*h;}

    public int getPerimetro() {return 2*l + 2*h;}

    public int getLado() {return this.l;}

    public int getAltura() {return this.h;}

}
```

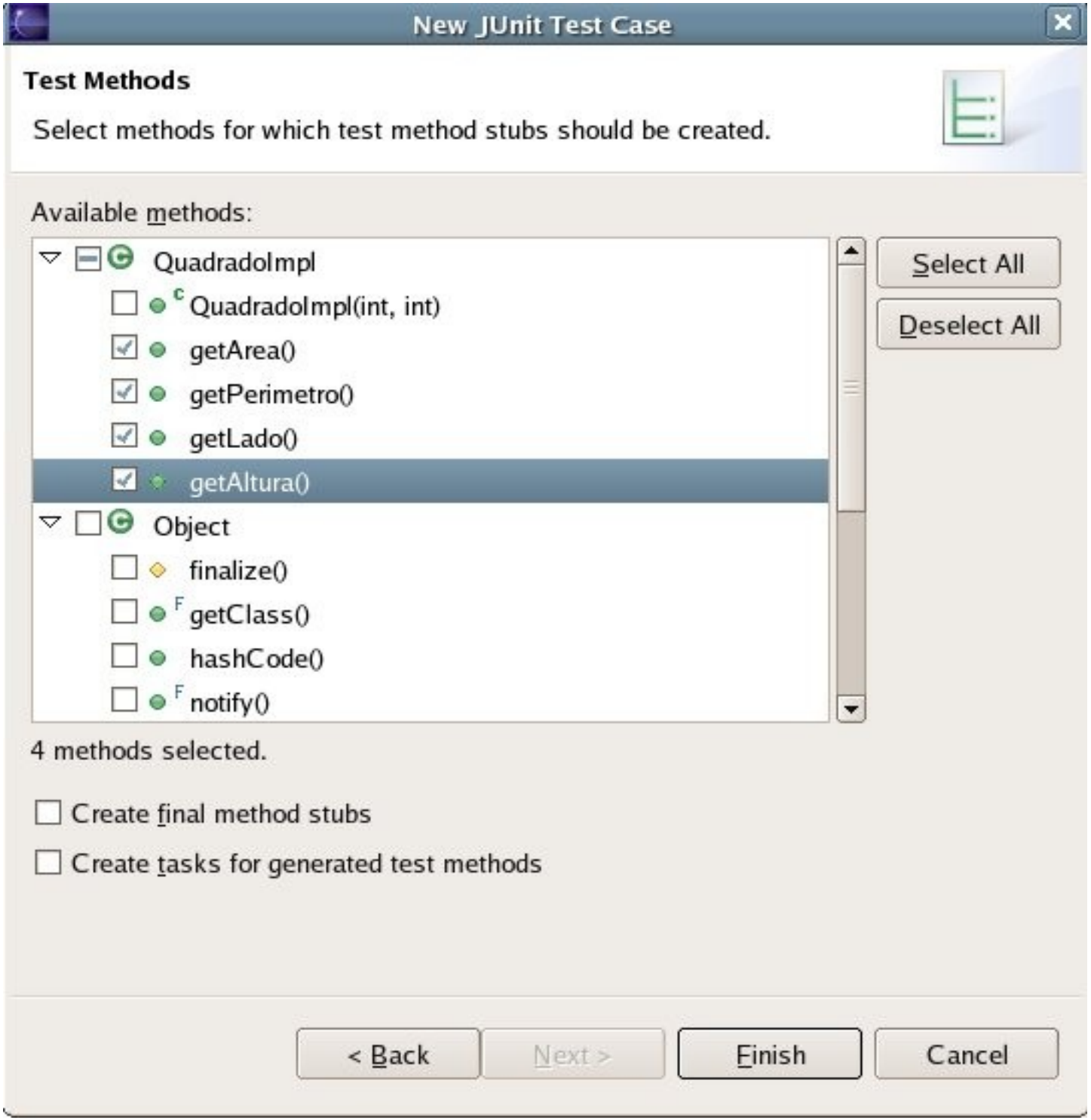
Criando teste com Junit no Eclipse

1. File>New>JUnit Test Case

2. Escolha um nome para o teste e selecione a classe a ser testada. Selecione a opção setUp() e clique em Next.

3. Escolha os métodos a serem testados.


New JUnit Test Case
JUnit Test Case
Select the name of the new JUnit test case. You have the options to specify the class under test and on the next page, to select methods to be tested.
Source folder:
teste
Browse...
Package:
src
Browse...
Name:
TesteQuadrado
Superclass:
junit.framework.TestCase
Browse...
Which method stubs would you like to create?
public static void main(String[] args)
Add TestRunner statement for:
text ui
setUp()
tearDown()
constructor()
Class under test:
src.QuadradoImpl
Browse...
< Back
Next >
Finish
Cancel


New JUnit Test Case
Test Methods
Select methods for which test method stubs should be created.
Available methods:
QuadradoImpl
QuadradoImpl(int, int)
getArea()
getPerimetro()
getLado()
getAltura()
Object
finalize()
getClass()
hashCode()
notify()
Select All
Deselect All
4 methods selected.
Create final method stubs
Create tasks for generated test methods
< Back
Next >
Finish
Cancel

Implementando o teste

```
package teste;

import junit.framework.TestCase;
import src.QuadradoImpl;

public class TesteQuadrado extends TestCase {

    QuadradoImpl q;

    //exemplo de uso do setup.
    protected void setUp() throws Exception {
        super.setUp();
        q = new QuadradoImpl(20, 10);
    }
```

```
public void testGetArea() {
        assertEquals("Area = ", 200, q.getArea());
}
```

* assertEquals(String, valorEsperado, valorEncontrado);

O String geralmente é utilizado para ajudar identificar onde ocorreu o erro ao se testar vários métodos ao mesmo tempo.

A idéia do teste é que se funcionou para um valor conhecido, o método está correto e irá funcionar para todos os outros valores. É claro que o programador deve ter o bom senso de saber se os valores que o usuário vai usar pode extrapolar o alcance do tipo da variável como float, double, int, etc. Então ele deve fazer o teste com o máximo e mínimo valor possível que o usuário irá utilizar, então se o teste funcionar para esses valores ele poderá garantir que funcionará para qualquer valor colocado naquele intervalo.

```
public void testGetArea() {
        assertEquals("Area = ", 200, q.getArea());
    }

public void testGetPerimetro() {
        assertEquals("Perimetro = ", 60, q.getPerimetro());
    }
```

```
public void testGetArea() {
        assertEquals("Area = ", 200, q.getArea());
    }

public void testGetPerimetro() {
        assertEquals("Perimetro = ", 60, q.getPerimetro());
    }

public void testGetLado() {
        assertEquals("Lado l = ", 20, q.getLado());
    }
```

```
public void testGetArea() {
        assertEquals("Area = ", 200, q.getArea());
    }

public void testGetPerimetro() {
        assertEquals("Perimetro = ", 60, q.getPerimetro());
    }

public void testGetLado() {
        assertEquals("Lado l = ", 20, q.getLado());
    }

public void testGetAltura() {
        assertEquals("Altura h = ", 10, q.getAltura());
    }
}
```

Dica

- Para testar arrays utilizar:

Arrays.Equals(array1, array2);

Pois, assertEquals(array1, array2), retornará o endereço da memória à qual a variável faz referência.

Código fonte

Baixar apresentação:

download Apresentação de JUnit

Código fonte do exemplo:

download código fonte do exemplo

FIM