ICMC USP SÃO CARLOS

Teste Automatizado

Prof. Marcio Delamaro

Colaborador: Stevão Andrade

Teste

- Ato de executar um programa e verificar se os resultados produzidos estão corretos, a fim de se encontrar problemas.
  - **Manual**: realizado passando-se os parâmetros e “olhando” se o resultado está certo
  - **Automatizado**: escrevendo um programa que executa seu programa e verifica os resultados

- A vantagem do teste automatizado é que ele pode ser executado muitas vezes, com um mínimo de esforço

- Teste de regressão

Classe Cal

- Vamos testar a classe Cal manualmente
- Executamos o Cal
- Em cada nova execução, verificar se o Calendário foi gerado corretamente

Classe Cal

- Vamos testar a classe Cal manualmente
- Executamos o Cal
- Em cada nova execução, verificar se o Calendário foi gerado corretamente
- E se...

```
 Maio 2019

Do Se Te Qa Qi Se Sa
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
```

Classe Cal

- Vamos testar a classe Cal manualmente
- Executamos o Cal
- Em cada nova execução, verificar se o Calendário foi gerado corretamente
- E se...

```
 Maio 2019

Do Se Te Qa Qi Se Sa

          10  2  3  4

 5  6  7  8  9 10 11

12 13 14 15 16 17 18

19 20 21 22 23 24 25

26 27 28 29 30 31
```

Automatizar

- Antes de usar uma classe, vamos tentar garantir que ela funciona

- Ao alterarmos a implementação de uma classe também

Automatizar

- Antes de usar uma classe, vamos tentar garantir que ela funciona

- Ao alterarmos a implementação de uma classe também

```
/**
 * Não tem função real dentro da classe. Foi usada
 * apenas para testar os métodos implementados
 * @param args -- Sem uso.
 */
static public void main(String[] args)
```

Iniciando a automatização

```
System.out.println("Teste para maio: ");
Cal.main(new String[] {"5", "2019"});

System.out.println("Teste para dezembro: ");
Cal.main(new String[] {"12", "2019"});
```

Resultado

```
Teste para maio:
   Maio 2019
Do Se Te Qa Qi Se Sa
            1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
```

```
Teste para dezembro:
   Dezembro 2019
Do Se Te Qa Qi Se Sa
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31
```

Resultado

Continuamos testando manualmente. Está correto o resultado apresentado?

```
Teste para maio:
   Maio 2019
Do Se Te Qa Qi Se Sa
            1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
```

```
Teste para dezembro:
   Dezembro 2019
Do Se Te Qa Qi Se Sa
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31
```

Resultado

Continuamos testando manualmente. Está correto o resultado apresentado?

```
Teste para maio:
   Maio 2019
Do Se Te Qa Qi Se Sa
            1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
```

```
Teste para dezembro:
   Dezembro 2019
Do Se Te Qa Qi Se Sa
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31
```

Como podemos melhorar essa automatização?

Melhorando

```
ByteArrayOutputStream baOut = new ByteArrayOutputStream();

System.out.println("Teste para maio: ");

Cal.main(new String[] {"5", "2019"});


//primeiro teste

String execucao = baOut.toString();

String resultado_esperado = "   Maio 2019\nDo Se Te Qa Qi Se Sa\n          1  2  3  4\n 5  6
7  8  " +

   "9 10 11\n12 13 14 15 16 17 18\n19 20 21 22 23 24 25\n26 27 28 29 30 31 \n";


//verificação

if (execucao.equals(resultado_esperado)){

     System.out.println("O programa esta funcionando OK!");

}else {

     System.out.println("Opa! Aconteceu um erro!");

}
```

Problema

- O problema é que a quantidade de código que é necessária para implementar os casos de teste é enorme

- Existem meios mais adequados para se fazer esse tipo de automatização

Junit

- Framework para automatização
- Teste de unidade
- Permite definir casos de teste
  - Entradas para um método
  - Saídas esperadas para aquelas entradas
- São definidos usando a própria linguagem Java

Criar classe de teste

- IDEs como Eclipse possuem suporte ao Junit

- Adicionar Jnuti ao projeto: Build Path >> Adicionar biblioteca >> JUnit (JUnit4)

- Criar a classe de teste: Selecionar classe a testar >> File New >> JUnit >> selecionar métodos a testar

Resultado

```
public class NewCalTest {

@Test
public void testMain() {
fail("Not yet implemented");
}

@Test
public void testFirstOfMonth() {
fail("Not yet implemented");
}
.
.
.
```

Resultado

```
public class NewCalTest {

@Test
public void testMain() {
fail("Not yet implemented");
}

@Test
public void testFirstOfMonth() {
fail("Not yet implemented");
}
.
.
.
```

Indica que cada um desses métodos é um caso de teste

Resultado

```
public class NewCalTest {

@Test
public void testMain() {
fail("Not yet implemented");
}

@Test
public void testFirstOfMonth() {
fail("Not yet implemented");
}
.
.
.
```

Indica que cada um desses métodos é um caso de teste

Pode-se usar qualquer nome mas aconselha-se usar um nome que indique quem está sendo testado

Executando

- Ao executar esse arquivo de teste o JUnit identifica @Test e executa cada um

- Em cada caso de teste devemos chamar um método passando parâmetros que desejarmos

- Devemos também verificar se o resultado da chamada é aquele esperado

Verificando o calendário

```
@Test

public void testMain01(){

        Cal.main(new String[] {});

        String s = baOut.toString();

        assertEquals("   Maio 2019\nDo Se Te Qa Qi Se Sa\n
1  2  3  4\n 5  6  7  8 9 10 11\n12 13 14 15 16 17 18\n19 20 21 22
23 24 25\n26 27 28 29 30 31 \n", s);

}
```

Instanciação e chamada normal de método

Verificando o calendário

```
@Test
public void testMain01(){
        Cal.main(new String[] {});
        String s = baOut.toString();
        assertEquals("   Maio 2019\nDo Se Te Qa Qi Se Sa\n
1  2  3  4\n 5  6  7  8 9 10 11\n12 13 14 15 16 17 18\n19 20 21 22
23 24 25\n26 27 28 29 30 31 \n", s);
}
```

Método que compara saída produzida com a esperada

Assertions

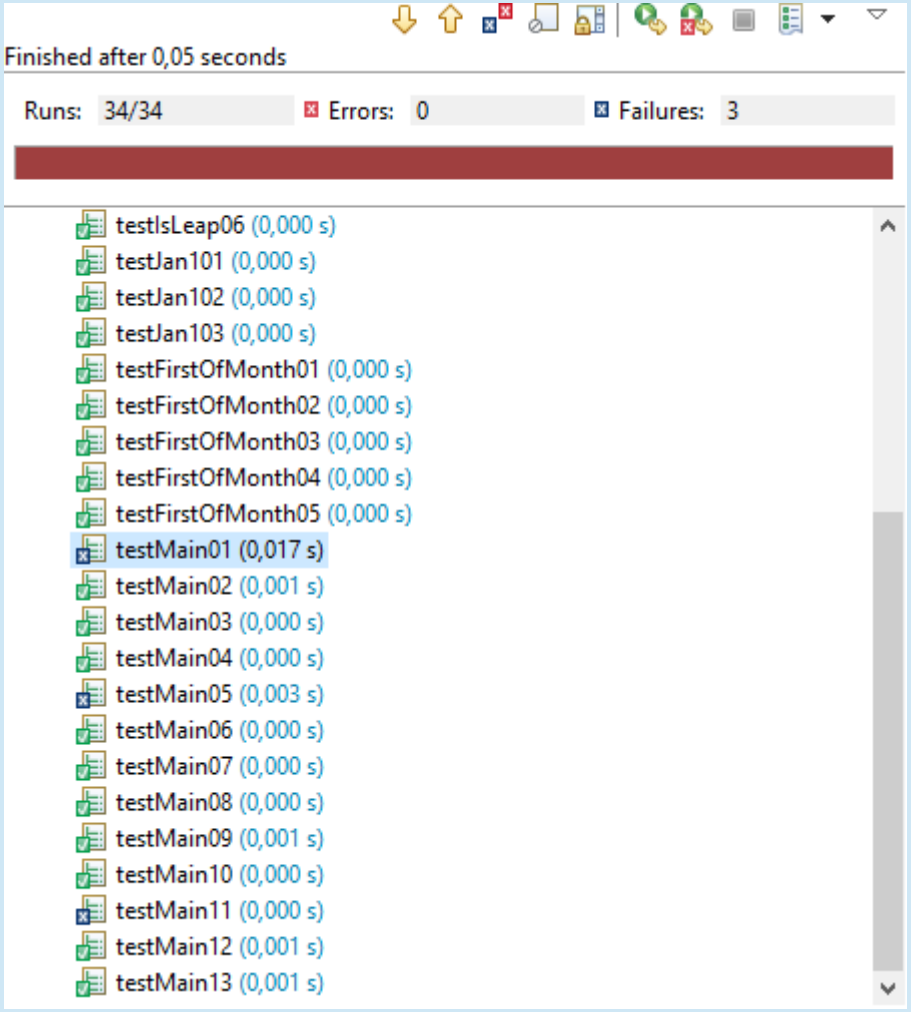
- assertEquals(expected, actual)
  - Dois valores (objetos) são iguais
- assertArrayEquals([] expecteds, [] actuals)
  - Dois arrays são iguais
- assertFalse/assertTrue(boolean)
- assertNull/assertNotNull(Object)
- assertSame/assertNotSame(Object, Object)
- fail(String)/fail()

Ao executar

- JUnit mostra quais casos de teste falharam e a diferença existente.

Ao executar

- JUnit mostra quais casos de teste falharam e a diferença existente.



Outras anotações

- @Before
  - Um método anotado é executado imediatamente antes de **cada** caso de teste
  - Útil para preparar o ambiente para o caso de teste
  - Criar um objeto, por exemplo

- @After
  - Executado ao fim de **cada** caso de teste

@Before

```
@Test
public void testMain02(){
        baErr.reset();
        String[] x = new String[] {"c"};
        Cal.main(x);
        String s = baErr.toString();
        assertEquals("Cal: c: ano invalido.\n", s);
}
@Test
public void testMain03(){
        baErr.reset();
        String[] x = new String[] {"0"};
        Cal.main(x);
        String s = baErr.toString();
        assertEquals("Cal: 0: ano invalido.\n", s);
}
```

@Before

```
static private ByteArrayOutputStream baOut, baErr;


@Before
public void setUp()
{
        cl = new Cal();
        baOut.reset();
        baErr.reset();
}
```

@Before

```
static private ByteArrayOutputStream baOut, baErr;

@Before

public void setUp()

{

        cl = new Cal();

        baOut.reset();

        baErr.reset();

}
```

```
@After
public void tearDown() {
        cl = null;
}
```

@BeforeClass

```
@BeforeClass
static public void beforeClassInit() {
        baOut = new ByteArrayOutputStream();
        psOut = new PrintStream(baOut);
        System.setOut(psOut);
        baErr = new ByteArrayOutputStream();
        psErr = new PrintStream(baErr);
        System.setErr(psErr);
}
```

Direciona as saídas convencional e de erro do console para as variáveis baOut e baErr

@AfterClass

```
@AfterClass
static public void afterClassFinalize() {
        psErr.close();
        psOut.close();
}
```

Fecha as variáveis baOut e baErr para as coisas voltarem ao normal.

Esperando exceção

- Quando um caso de teste deve gerar uma exceção

```
@Test
public void testFirstOfMonth01() {
        assertEquals(null, cl.firstOfMonth(null, null));
??????????????????????
}
```

Esperando exceção

- Quando um caso de teste deve gerar uma exceção

```
@Test(expected = NullPointerException.class)
public void testFirstOfMonth01() {
       cl.firstOfMonth(null, null));
}
```

Métodos void

- Métodos que não retornam nada são um problema

- Podemos usar outros métodos para verificar o estado após a chamada

- Por exemplo, usar ***algum método*** para poder testar o método ***constructor da classe***

Quais casos de teste

- Mas quantos casos de teste devo definir?

- Quais casos de teste?

Quais casos de teste

- Mas quantos casos de teste devo definir?
- Quais casos de teste?
- Essa é uma pergunta muito difícil de responder

Cobertura de código

- Sem entrar no mérito do quão bom ou ruim

- Um requisito básico é que cada um dos comandos do programa sejam executados

- Ou melhor ainda, cada desvio seja executado pelo menos uma vez
  - if
  - while
  - switch

Eclemma

- Existem várias ferramentas que verificam a cobertura de código
- Executando um conjunto de teste ela diz o que foi e o que não foi executado
- Eclemma é uma ferramenta simples
- Trabalha integrada com o Junit
- Trabalha integrado com o Eclipse

Como funciona

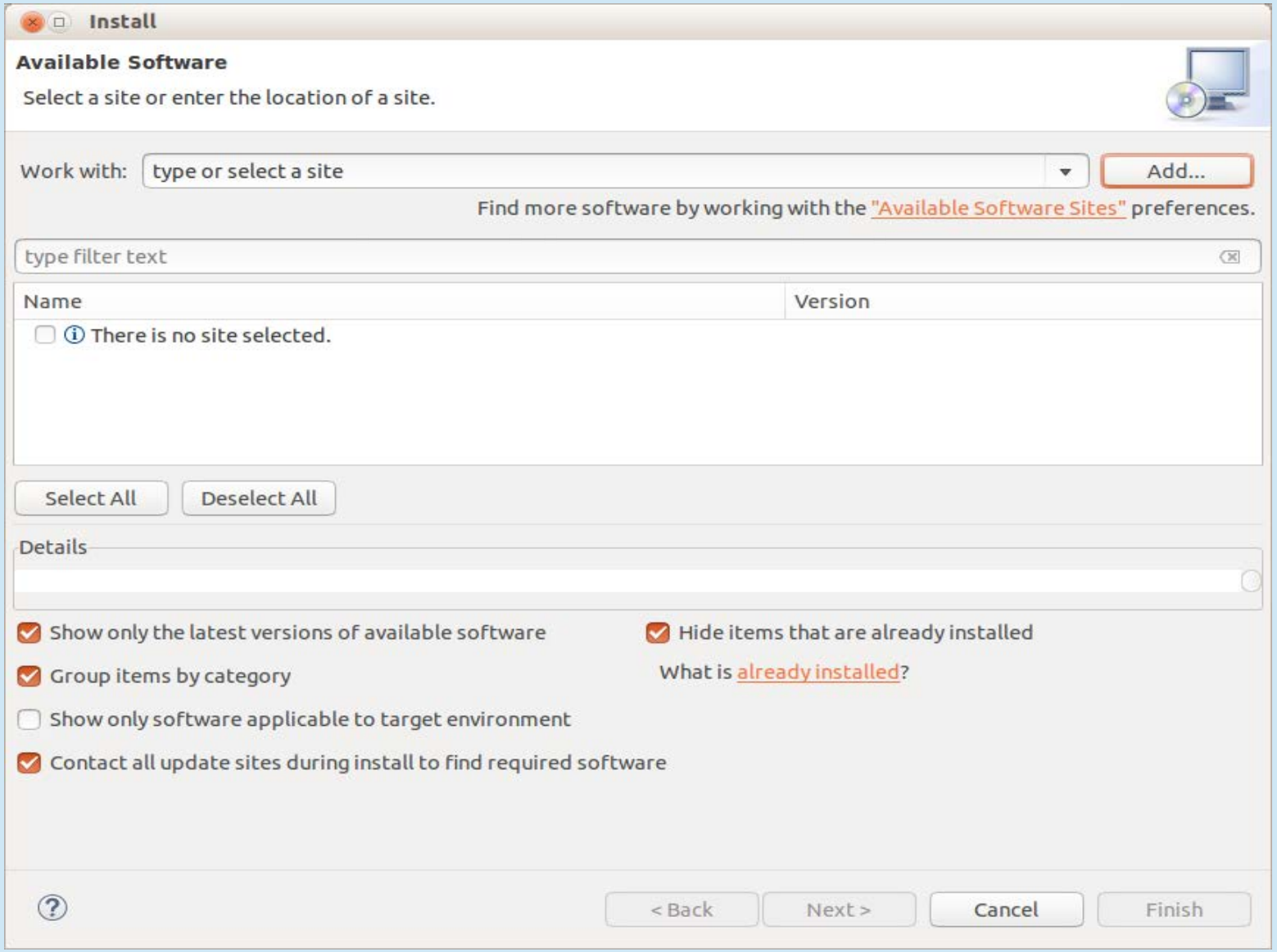
- Instalar: Help >> Install new >> http://update.eclemma.org

- Executar:
  - Selecionar o arquivo de teste desejado
  - Executar com “Code coverage”

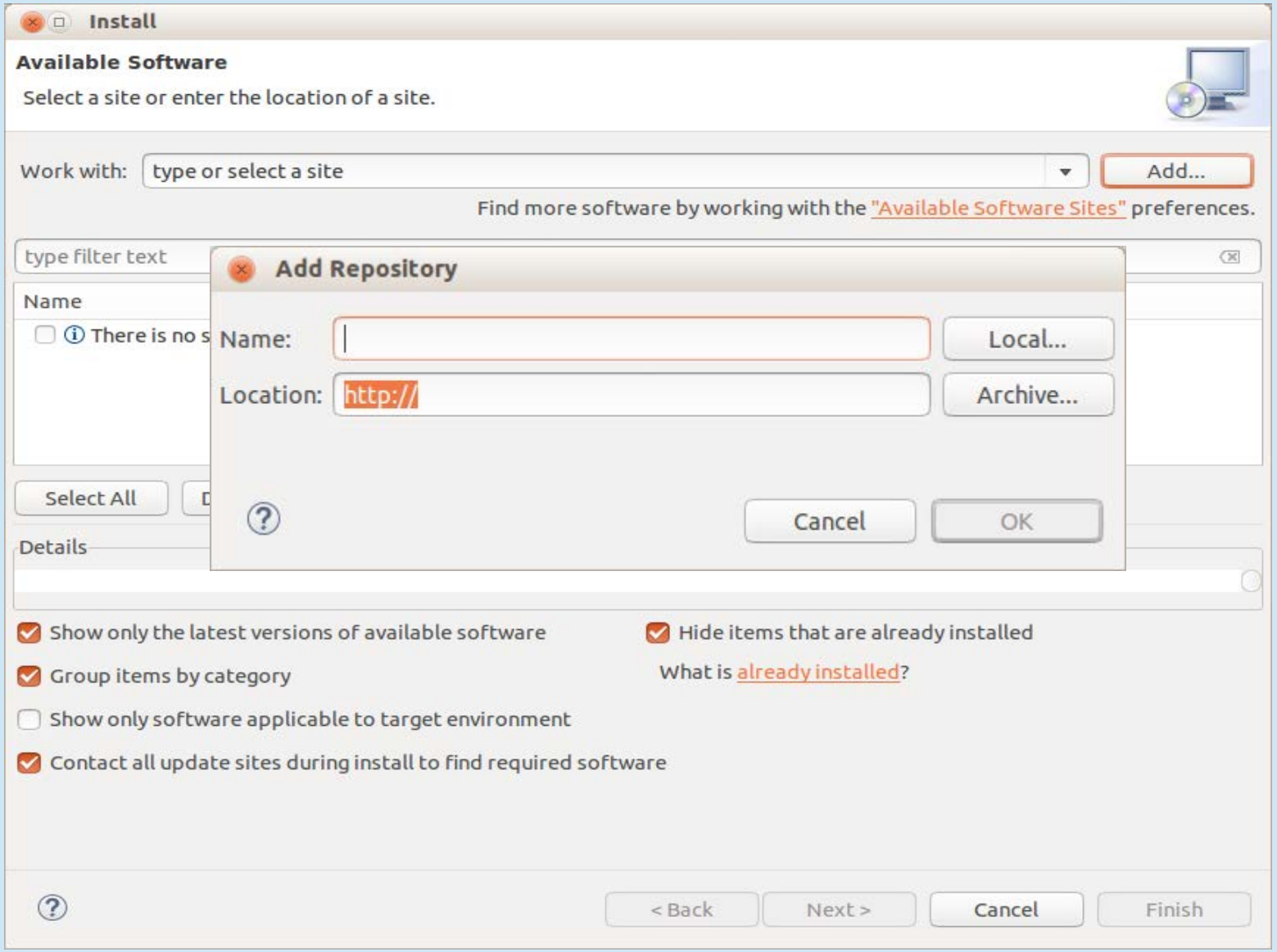
- Verificar o relatório de execução

- Criar novos casos de teste para cobrir o que não foi coberto

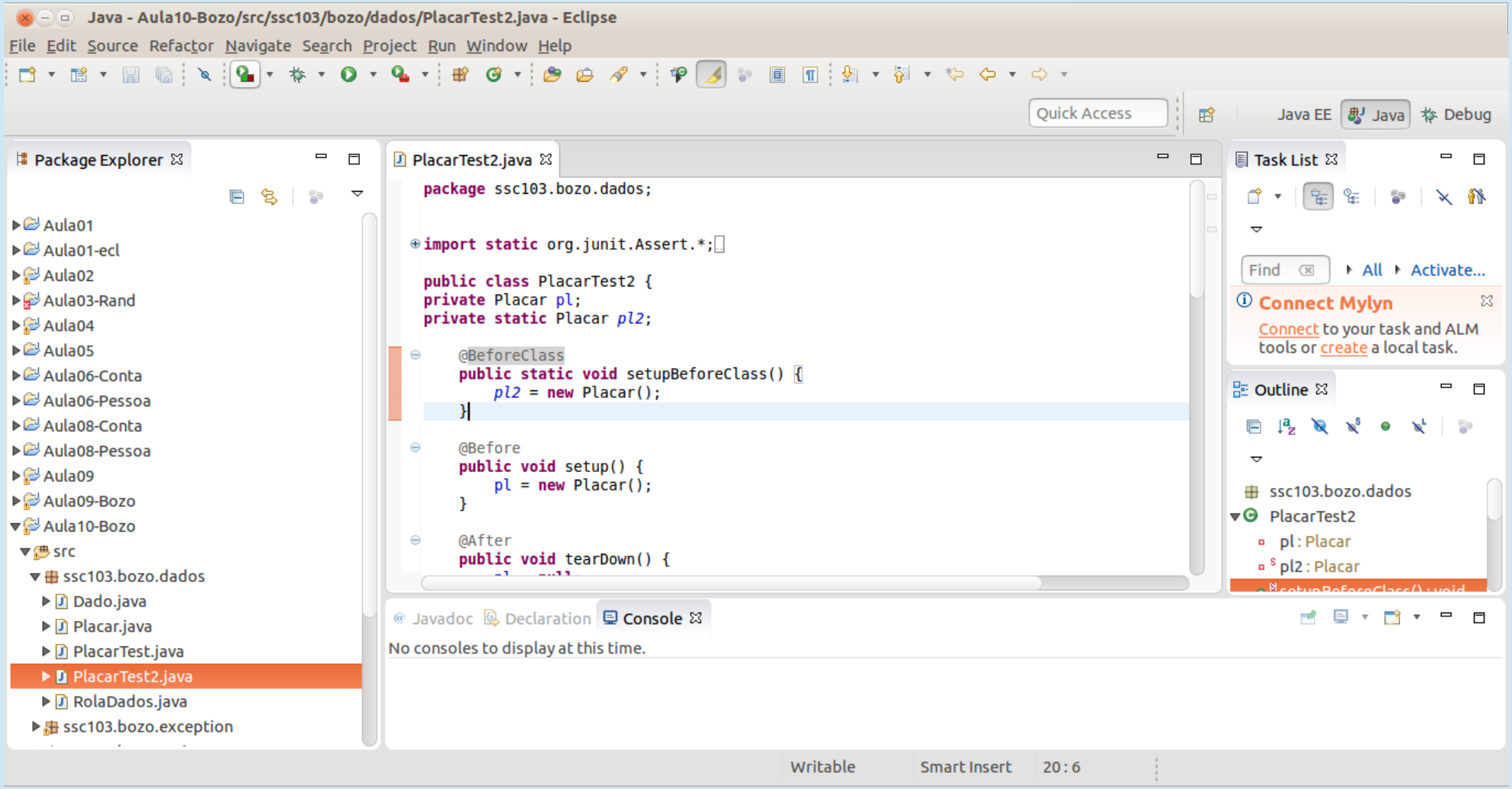
Instalar


Install
Available Software
Select a site or enter the location of a site.
Work with:
type or select a site
Add...
Find more software by working with the "Available Software Sites" preferences.
type filter text
Name
Version
There is no site selected.
Select All
Deselect All
Details
Show only the latest versions of available software
Hide items that are already installed
Group items by category
What is already installed?
Show only software applicable to target environment
Contact all update sites during install to find required software
< Back
Next >
Cancel
Finish

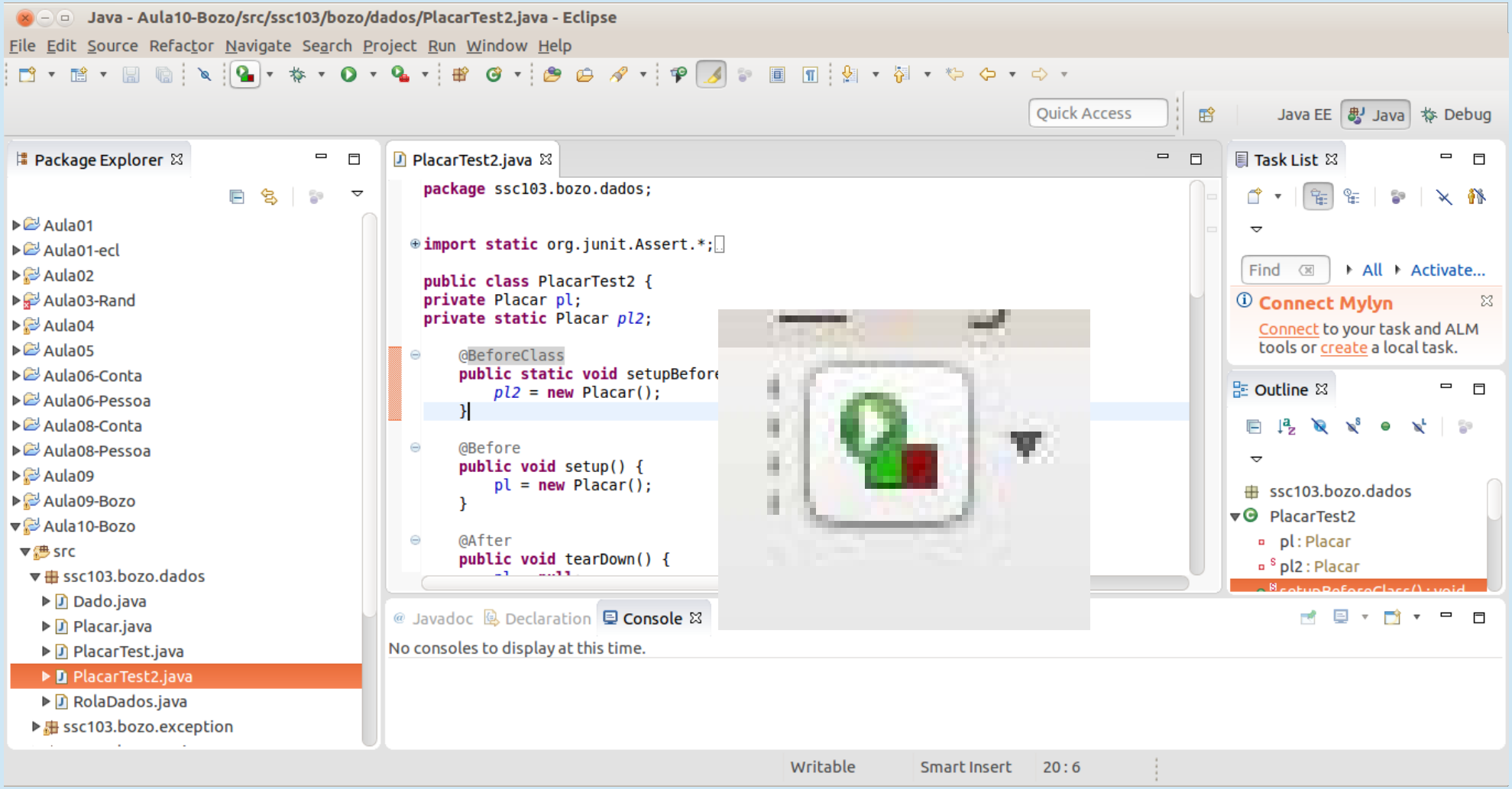
Instalar


Install
Available Software
Select a site or enter the location of a site.
Work with:
type or select a site
Add...
Find more software by working with the "Available Software Sites" preferences.
type filter text
Name
There is no s
Add Repository
Name:
Local...
Location:
http://
Archive...
Select All
Cancel
OK
Details
Show only the latest versions of available software
Hide items that are already installed
Group items by category
What is already installed?
Show only software applicable to target environment
Contact all update sites during install to find required software
< Back
Next >
Cancel
Finish

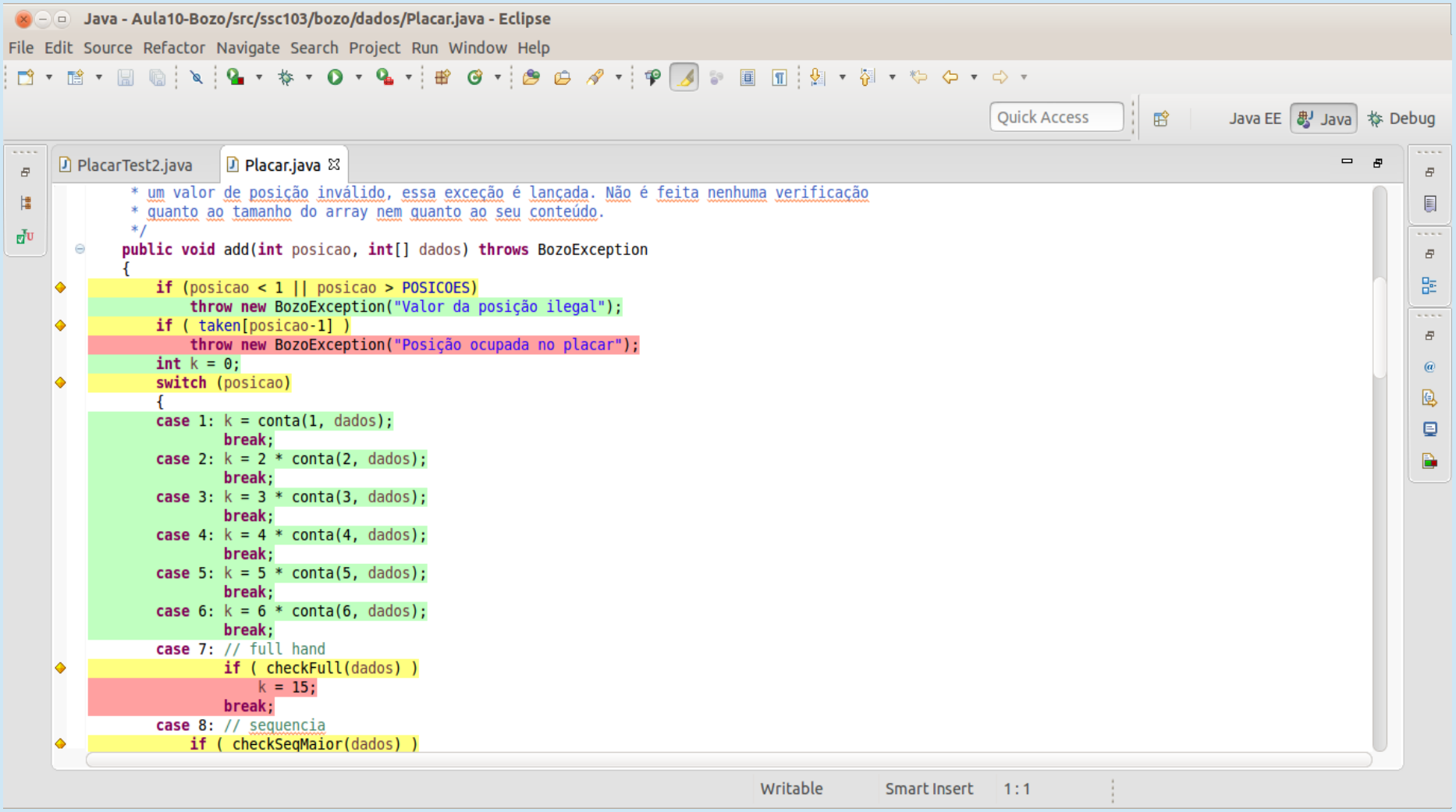
Execução


Java - Aula10-Bozo/src/ssc103/bozo/dados/PlacarTest2.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java EE
Java
Debug
Package Explorer
Aula01
Aula01-ecl
Aula02
Aula03-Rand
Aula04
Aula05
Aula06-Conta
Aula06-Pessoa
Aula08-Conta
Aula08-Pessoa
Aula09
Aula09-Bozo
Aula10-Bozo
src
ssc103.bozo.dados
Dado.java
Placar.java
PlacarTest.java
PlacarTest2.java
RolaDados.java
ssc103.bozo.exception
PlacarTest2.java
package ssc103.bozo.dados;
import static org.junit.Assert.*;
public class PlacarTest2 {
private Placar pl;
private static Placar pl2;
@BeforeClass
public static void setupBeforeClass() {
pl2 = new Placar();
}
@Before
public void setup() {
pl = new Placar();
}
@After
public void tearDown() {
Task List
Find
All
Activate...
Connect Mylyn
Connect to your task and ALM tools or create a local task.
Outline
ssc103.bozo.dados
PlacarTest2
pl : Placar
pl2 : Placar
Javadoc
Declaration
Console
No consoles to display at this time.
Writable
Smart Insert
20 : 6

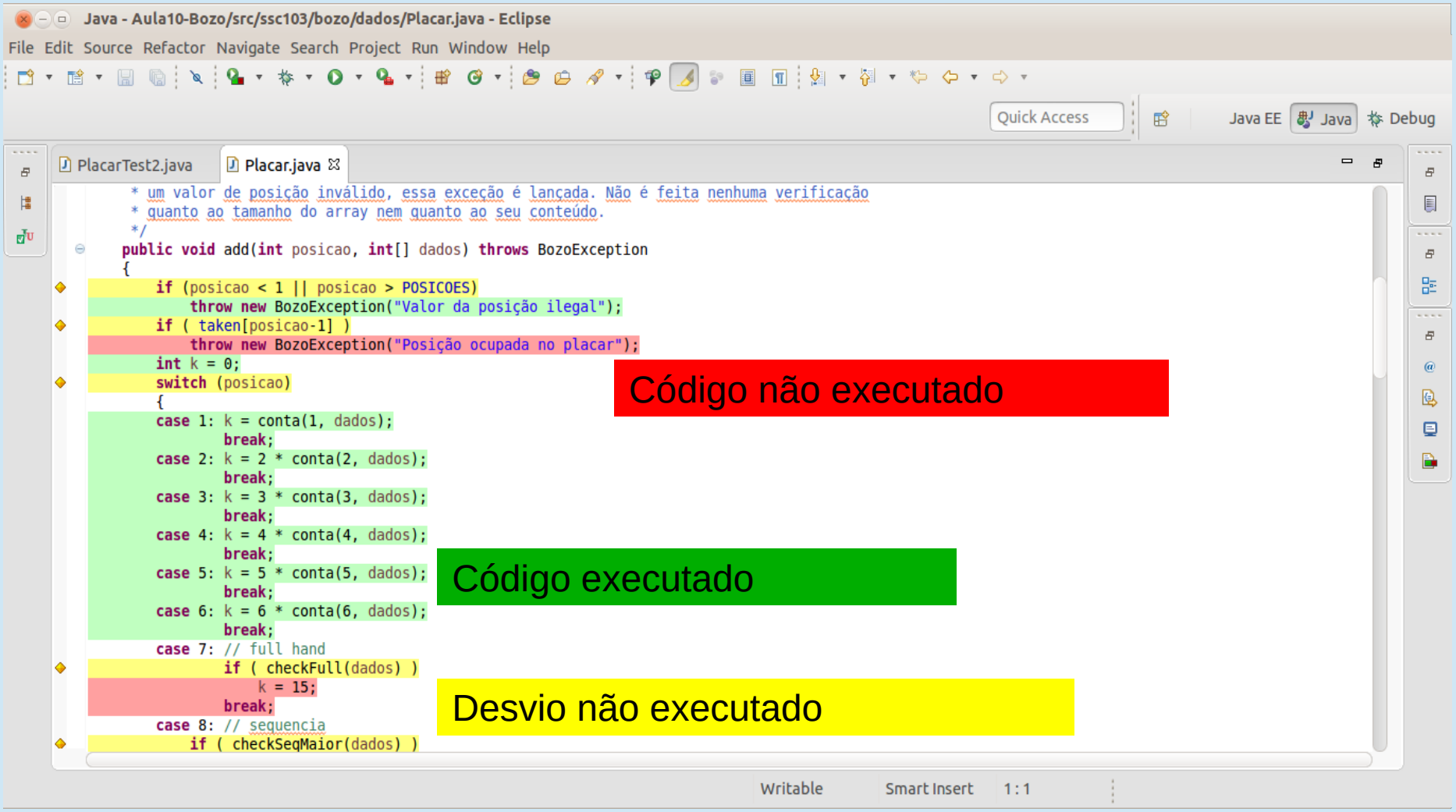
Execução


Java - Aula10-Bozo/src/ssc103/bozo/dados/PlacarTest2.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java EE
Java
Debug
Package Explorer
Aula01
Aula01-ecl
Aula02
Aula03-Rand
Aula04
Aula05
Aula06-Conta
Aula06-Pessoa
Aula08-Conta
Aula08-Pessoa
Aula09
Aula09-Bozo
Aula10-Bozo
src
ssc103.bozo.dados
Dado.java
Placar.java
PlacarTest.java
PlacarTest2.java
RolaDados.java
ssc103.bozo.exception
PlacarTest2.java
package ssc103.bozo.dados;
import static org.junit.Assert.*;
public class PlacarTest2 {
private Placar pl;
private static Placar pl2;
@BeforeClass
public static void setupBefore
pl2 = new Placar();
}
@Before
public void setup() {
pl = new Placar();
}
@After
public void tearDown() {
Task List
Find
All
Activate...
Connect Mylyn
Connect to your task and ALM tools or create a local task.
Outline
ssc103.bozo.dados
PlacarTest2
pl : Placar
pl2 : Placar
Javadoc
Declaration
Console
No consoles to display at this time.
Writable
Smart Insert
20 : 6

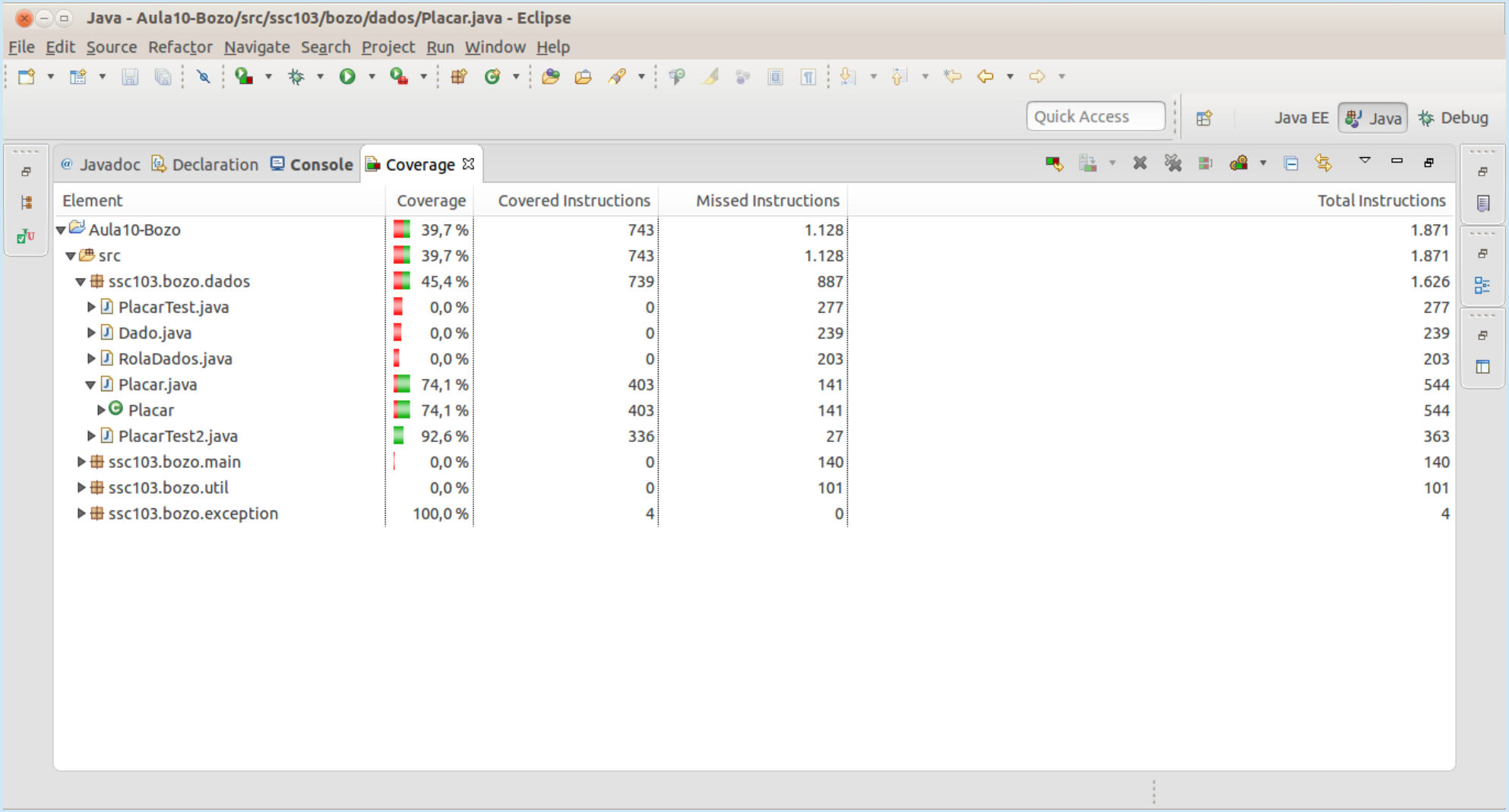
Resultados


Java - Aula10-Bozo/src/ssc103/bozo/dados/Placar.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java EE
Java
Debug
PlacarTest2.java
Placar.java
* um valor de posição inválido, essa exceção é lançada. Não é feita nenhuma verificação
* quanto ao tamanho do array nem quanto ao seu conteúdo.
*/
public void add(int posicao, int[] dados) throws BozoException
{
if (posicao < 1 || posicao > POSICOES)
throw new BozoException("Valor da posição ilegal");
if (taken[posicao-1])
throw new BozoException("Posição ocupada no placar");
int k = 0;
switch (posicao)
{
case 1: k = conta(1, dados);
break;
case 2: k = 2 * conta(2, dados);
break;
case 3: k = 3 * conta(3, dados);
break;
case 4: k = 4 * conta(4, dados);
break;
case 5: k = 5 * conta(5, dados);
break;
case 6: k = 6 * conta(6, dados);
break;
case 7: // full hand
if (checkFull(dados))
k = 15;
break;
case 8: // sequencia
if (checkSeqMaior(dados))
Writable
Smart Insert
1 : 1

Resultados

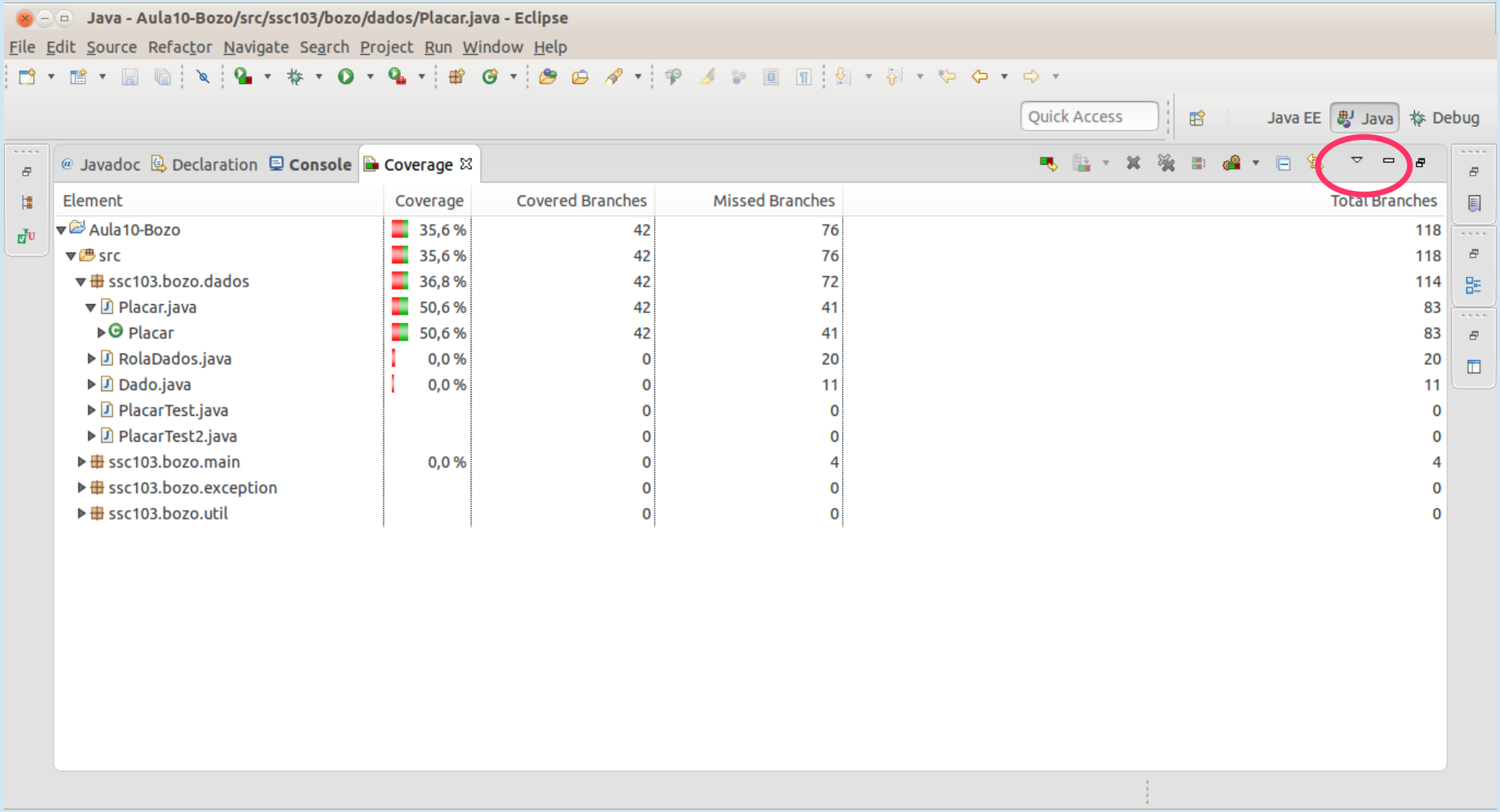

Java - Aula10-Bozo/src/ssc103/bozo/dados/Placar.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java EE
Java
Debug
PlacarTest2.java
Placar.java
* um valor de posição inválido, essa exceção é lançada. Não é feita nenhuma verificação
* quanto ao tamanho do array nem quanto ao seu conteúdo.
*/
public void add(int posicao, int[] dados) throws BozoException
{
if (posicao < 1 || posicao > POSICOES)
throw new BozoException("Valor da posição ilegal");
if (taken[posicao-1])
throw new BozoException("Posição ocupada no placar");
int k = 0;
switch (posicao)
{
case 1: k = conta(1, dados);
break;
case 2: k = 2 * conta(2, dados);
break;
case 3: k = 3 * conta(3, dados);
break;
case 4: k = 4 * conta(4, dados);
break;
case 5: k = 5 * conta(5, dados);
break;
case 6: k = 6 * conta(6, dados);
break;
case 7: // full hand
if (checkFull(dados))
k = 15;
break;
case 8: // sequencia
if (checkSeqMaior(dados))
Código não executado
Código executado
Desvio não executado
Writable
Smart Insert
1 : 1

Resultados – instruções



Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
Aula10-Bozo	39,7 %	743	1.128	1.871
src	39,7 %	743	1.128	1.871
ssc103.bozo.dados	45,4 %	739	887	1.626
PlacarTest.java	0,0 %	0	277	277
Dado.java	0,0 %	0	239	239
RolaDados.java	0,0 %	0	203	203
Placar.java	74,1 %	403	141	544
Placar	74,1 %	403	141	544
PlacarTest2.java	92,6 %	336	27	363
ssc103.bozo.main	0,0 %	0	140	140
ssc103.bozo.util	0,0 %	0	101	101
ssc103.bozo.exception	100,0 %	4	0	4

Resultados – desvios



Exercício

- Complete os testes da classe *Cal* até atingir 100% de cobertura de desvios para todos os métodos (Ignorar o Main)

Exercício para casa

- Complete os testes da classe *Cal* até atingir 100% de cobertura de desvios para todos os métodos

- Inclusive o metodo Main

ICMC USP SÃO CARLOS

Teste Automatizado

Prof. Marcio Delamaro

Colaborador: Stevão Andrade