# Teste Automatizado

# POO

Prof. Marcio Delamaro

# Teste

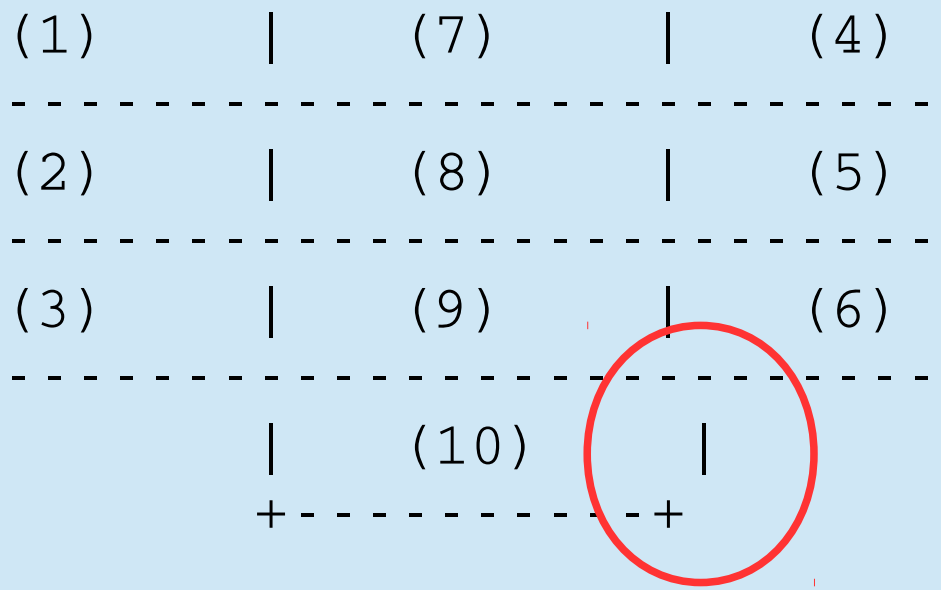
- Ato de executar um programa e verificar se os resultados produzidos estão corretos
- Manual: realizado passando-se os parâmetros e “olhando” se o resultado está certo
- Automatizado: escrevendo um programa que executa seu programa e verifica os resultados
- A vantagem do teste automatizado é que ele pode ser executado muitas vezes, com um mínimo de esforço
- Teste de regressão

# Classe Placar

- Voltando ao Bozo. Vamos testar a classe placar manualmente
- Executamos o Bozo
- Em cada jogada verificamos se o placar apresentado está correto

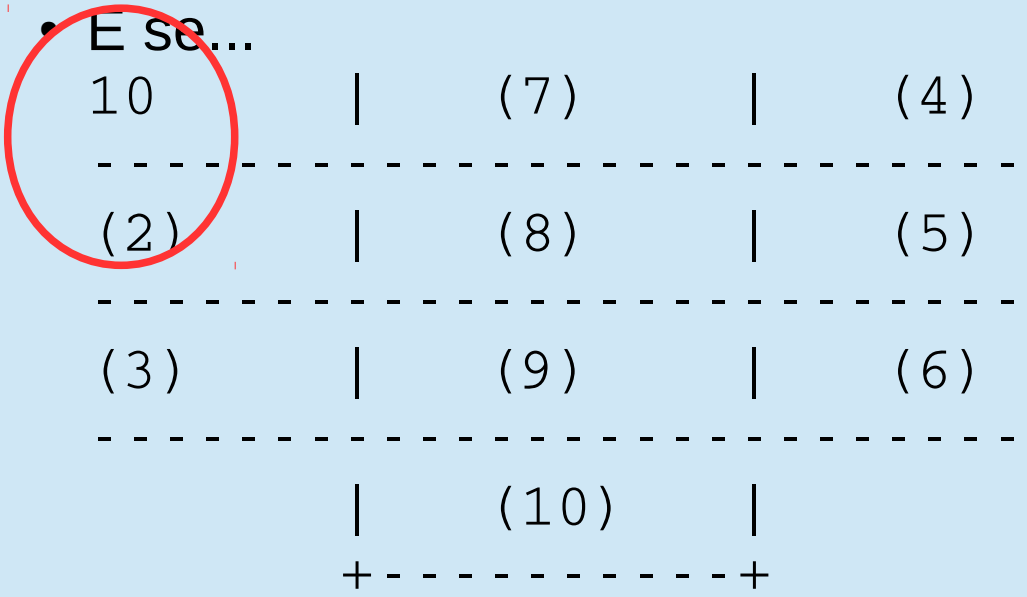
# Classe Placar

- Voltando ao Bozo. Vamos testar a classe placar manualmente
- Executamos o Bozo
- Em cada jogada verificamos se o placar apresentado está correto
- E se...



# Classe Placar

- Voltando ao Bozo. Vamos testar a classe placar manualmente
- Executamos o Bozo
- Em cada jogada verificamos se o placar apresentado está correto
- E se...



# 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

```
Placar pl = new Placar();
System.out.println("Score: “ + pl.getScore());
System.out.println(pl);

pl.add(1, new int[] {1, 2, 3, 4, 5 } );
System.out.println("Score: " + pl.getScore());
System.out.println(pl);

pl = new Placar();
pl.add(1, new int[] {1, 1, 3, 4, 5 } );
System.out.println("Score: " + pl.getScore());
System.out.println(pl);
```

# Resultado

```
Score: 0
(1)     |   (7)     |   (4)
--------------------------
(2)     |   (8)     |   (5)
--------------------------
(3)     |   (9)     |   (6)
--------------------------
        |   (10)    |
        +-----------+

Score: 1
1       |   (7)     |   (4)
--------------------------
(2)     |   (8)     |   (5)
--------------------------
(3)     |   (9)     |   (6)
--------------------------
        |   (10)    |
        +-----------+

Score: 2
2       |   (7)     |   (4)
--------------------------
(2)     |   (8)     |   (5)
--------------------------
(3)     |   (9)     |   (6)
--------------------------
        |   (10)    |
        +-----------+
```

# Resultado

```
Score: 0
(1)     |   (7)     |   (4)
--------------------------
(2)     |   (8)     |   (5)
--------------------------
(3)     |   (9)     |   (6)
--------------------------
        |   (10)    |
        +-----------+

Score: 1
1       |   (7)     |   (4)
--------------------------
(2)     |   (8)     |   (5)
--------------------------
(3)     |   (9)     |   (6)
--------------------------
        |   (10)    |
        +-----------+

Score: 2
2       |   (7)     |   (4)
--------------------------
(2)     |   (8)     |   (5)
--------------------------
(3)     |   (9)     |   (6)
--------------------------
        |   (10)    |
        +-----------+
```

Continuamos testando manualmente. Está correto o resultado apresentado?

# Resultado

```
Score: 0
(1)     |   (7)     |   (4)
--------------------------
(2)     |   (8)     |   (5)
--------------------------
(3)     |   (9)     |   (6)
--------------------------
        |   (10)    |
        +-----------+

Score: 1
1       |   (7)     |   (4)
--------------------------
(2)     |   (8)     |   (5)
--------------------------
(3)     |   (9)     |   (6)
--------------------------
        |   (10)    |
        +-----------+

Score: 2
2       |   (7)     |   (4)
--------------------------
(2)     |   (8)     |   (5)
--------------------------
(3)     |   (9)     |   (6)
--------------------------
        |   (10)    |
        +-----------+
```

Continuamos testando manualmente. Está correto o resultado apresentado?

Como podemos melhorar essa automatização?

# Melhorando

```
Placar pl = new Placar();
int k = pl.getScore();
if ( k != 0 )
   System.out.println(“Erro. Resultado esperado: 0.
Resultado obtido: “ + k);

String s = pl.toString();
String r = “(1)    |   (7)    |   (4) \n-----...”
if (! r.equals(s) )
      System.out.println(“Erro. Resultado esperado: “
+ r + “. Resultado obtido: “ + k);
```

# 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 PlacarTest {

  @Test
  public void testAdd() {
        fail("Not yet implemented");
  }
  @Test
  public void testGetScore() {
     fail("Not yet implemented");
  }
  @Test
  public void testToString() {
     fail("Not yet implemented");
  }
}
```

# Resultado

```
public class PlacarTest {

   @Test
   public void testAdd() {
          fail("Not yet implemented");
   }
   @Test
   public void testGetScore() {
       fail("Not yet implemented");
   }
   @Test
   public void testToString() {
       fail("Not yet implemented");
   }
}
```

Indica que cada um desses métodos é um caso de teste

# Resultado

```
public class PlacarTest {

  @Test
  public void testAdd() {
        fail("Not yet implemented");
  }
  @Test
  public void testGetScore() {
     fail("Not yet implemented");
  }
  @Test
  public void testToString() {
     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

# Placar.getScore

```
@Test
public void testGetScoreVazio() {
   Placar pl = new Placar();
   int k = pl.getScore();
   assertEquals(0, k);
}
```

# Placar.getScore

```
@Test
public void testGetScoreVazio() {
    Placar pl = new Placar();
    int k = pl.getScore();
    assertEquals(0, k);
}
```

Instanciação e chamada normal de método

# Placar.getScore

```
@Test
public void testGetScoreVazio() {
   Placar pl = new Placar();
   int k = pl.getScore();
   assertEquals(0, k);
}
```

Método que compara saída produzida com a esperada

# Placar.getScore

```
@Test
public void testGetScoreCheio() {
   Placar pl = new Placar();
   pl.add(1, new int[] {1, 2, 3, 4, 5} );
   pl.add(2, new int[] {1, 2, 3, 4, 5} );
   pl.add(3, new int[] {1, 2, 3, 4, 5} );
   pl.add(4, new int[] {1, 2, 3, 4, 5} );
   pl.add(5, new int[] {1, 2, 3, 4, 5} );
   pl.add(6, new int[] {1, 2, 3, 4, 5} );
   pl.add(7, new int[] {1, 2, 3, 4, 5} );
   pl.add(8, new int[] {1, 2, 3, 4, 5} );
   pl.add(9, new int[] {1, 2, 3, 4, 5} );
   pl.add(10, new int[] {1, 2, 3, 4, 5} );
   int k = pl.getScore();
   assertEquals(35, k);
}
```

# Assertions

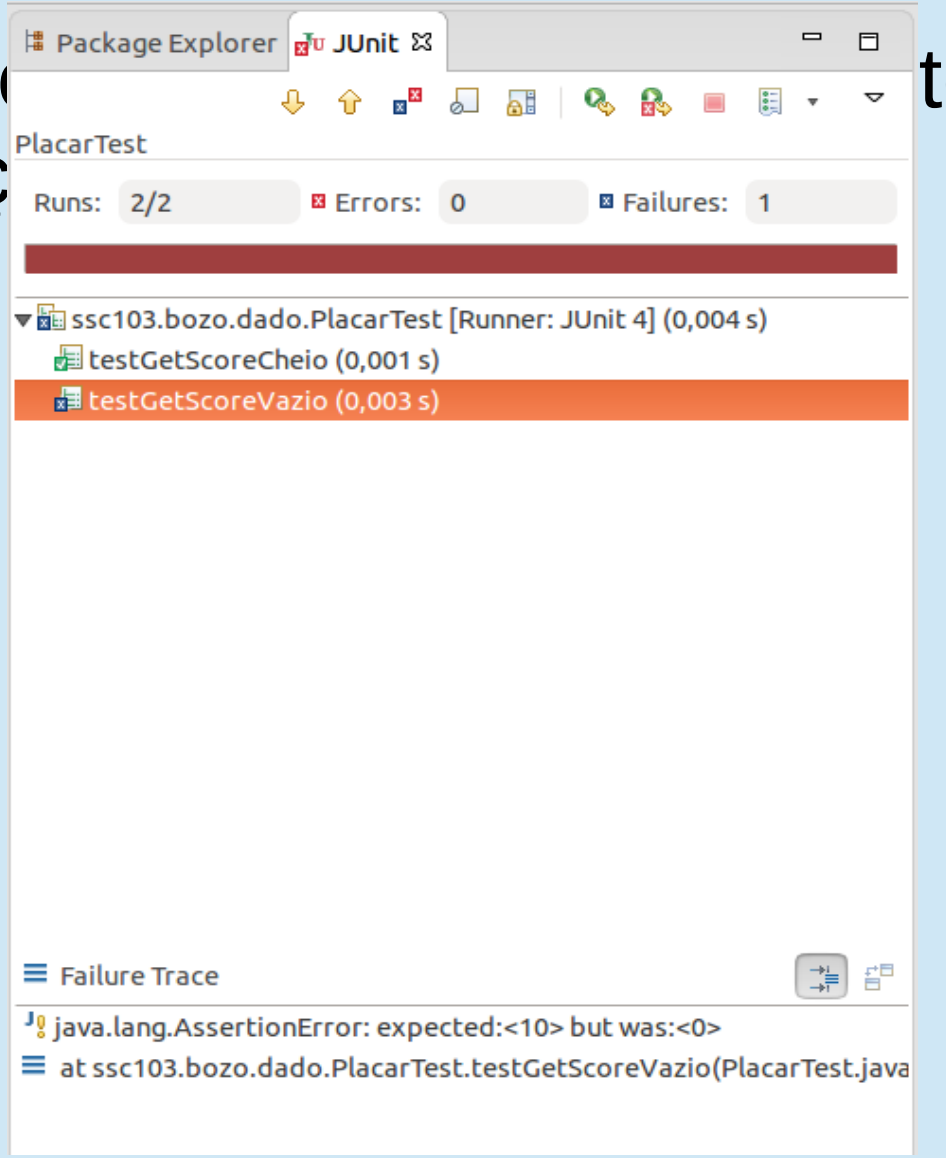
- assertEquals(expected,  actual)
  - Dois valores (objetos) são iguais
- assertArrayEquals([] expecteds, [] actuals)
  - Dois arrays são iguais
- assertFalse/assertTrue(boolean)
- assertNull/assertNotNull(Object)
- assertSame/assertNotSame(Object, Object)
- fail(String)/fail()

# Ao executar

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

# Ao executar

- JUnit m te falharam e a diferenç



# Outras anotaçõs

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

# @Before

```
@Test
public void testGetScoreVazio() {
  Placar pl = new Placar();
  int k = pl.getScore();
  assertEquals(10, k);
}

@Test
public void testGetScoreCheio() {
  Placar pl = new Placar();
  pl.add(1, new int[] {1, 2, 3, 4, 5} );
      …
```

# @Before

```
@Test
public void testGetScoreVazio() {
  Placar pl = new Placar();
  int k = pl.getScore();
  assertEquals(10, k);
}

@Test
public void testGetScoreCheio() {
  Placar pl = new Placar();
  pl.add(1, new int[] {1, 2, 3, 4, 5} );
      …
```

# @Before

```
private Placar pl;

  @Before
  public void setup() {
     pl = new Placar();
  }

  @Test
  public void testGetScoreVazio() {
     int k = pl.getScore();
     assertEquals(10, k);
  }
```

# @Before

```
private Placar pl;

  @Before
  public void setup() {
     pl = new Placar();
  }

  @Test
  public void testGetScoreVazio() {
     int k = pl.getScore();
     assertEquals(10, k);
  }
```

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

# @BeforeClass

```
private static Placar pl2;

    @BeforeClass
    public static void setupBeforeClass() {
       pl2 = new Placar();
    }

   @Test

   public void testToString1() {

   String r = "(1)     |    (7)     |    (4)\n" +
                ... +

              "+----------+\n";

   assertEquals(pl2.toString(), r);

   }
```

# @BeforeClass

```
@Test
public void testToString2() {
      p12.add(1, new int[] {1,1,1,1,1});
      String r = "5       |    (7)     |    (4)\n" +
               ... +
              "+----------+\n";
      assertEquals(p12.toString(), r);
   }
```

# Esperando exceção

- Quando um caso de teste deve gerar uma exceção

```
@Test
public void testAddPosInvalida() {
   pl.add(0, new int[] {1,1,1,1,1});
   assert????();
}
```

# Esperando exceção

- Quando um caso de teste deve gerar uma exceção

```
@Test(expected=BozoException.class)
public void testAddPosInvalida() {
   p1.add(0, new int[] {1,1,1,1,1});
}
```

# Métodos void

- Métodos que não retornam nada são um problema
- Podemos usar outros métodos para verificar o estado após a chamada
- Por exemplo, usar ***getScore()*** para poder testar o método ***add()***

# 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
- Método add
  - Para uma posição do placar são $6^5$ opções
  - Ordem de inserir no placar: 10!
  - Posições inválidas

# 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

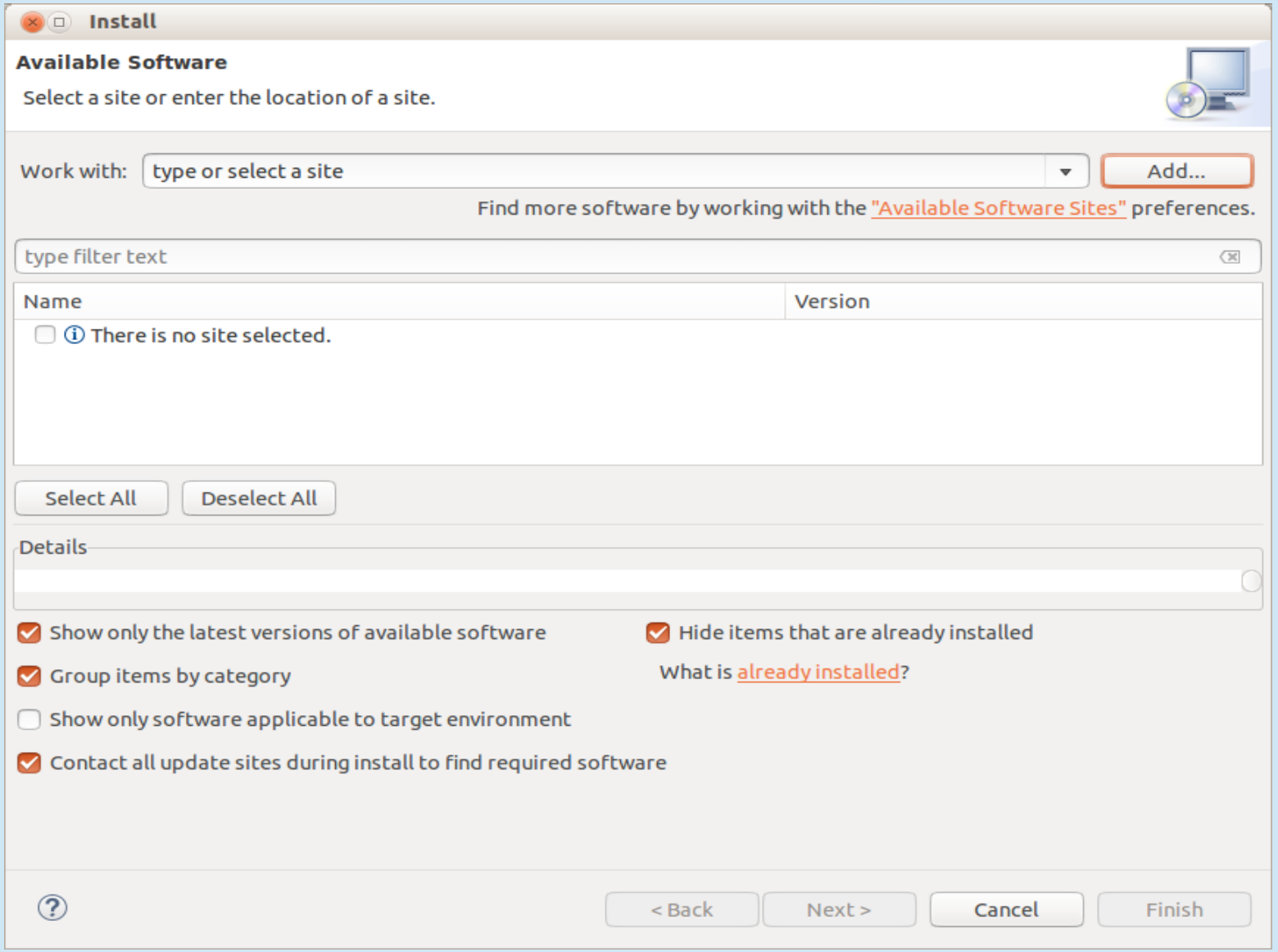
# Eclemma

- 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

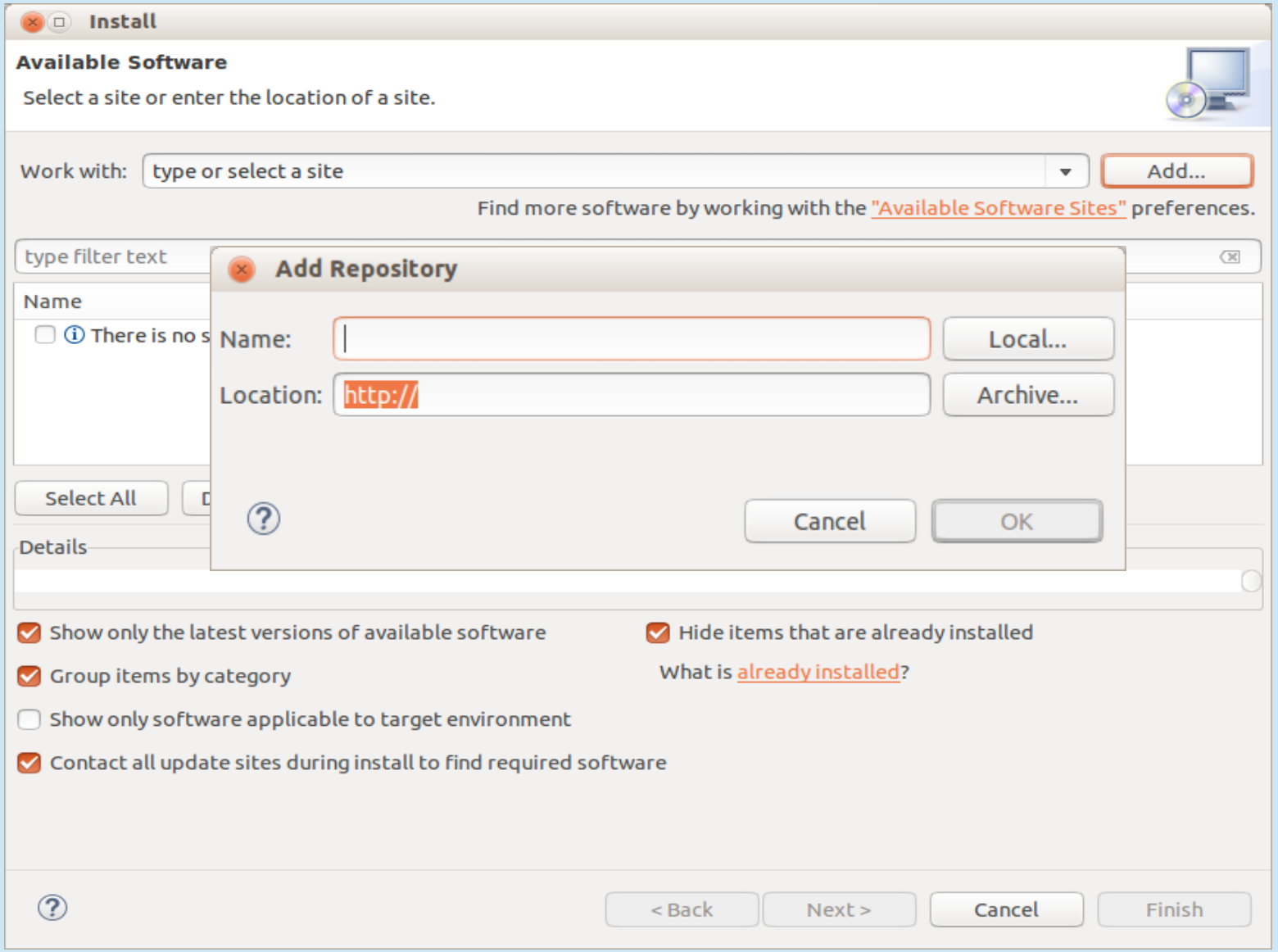
# Como funciona

- 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

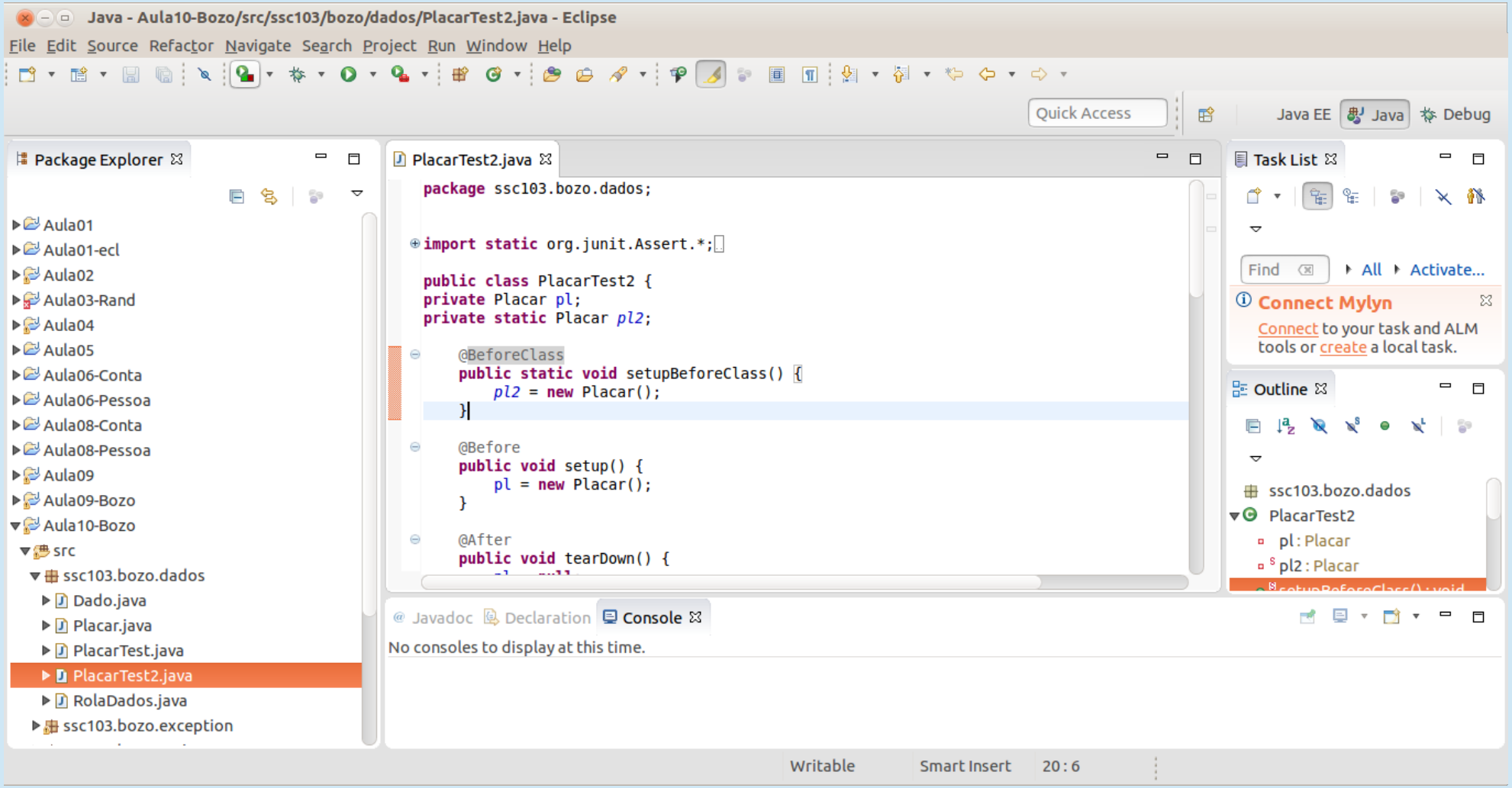
# Instalar


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

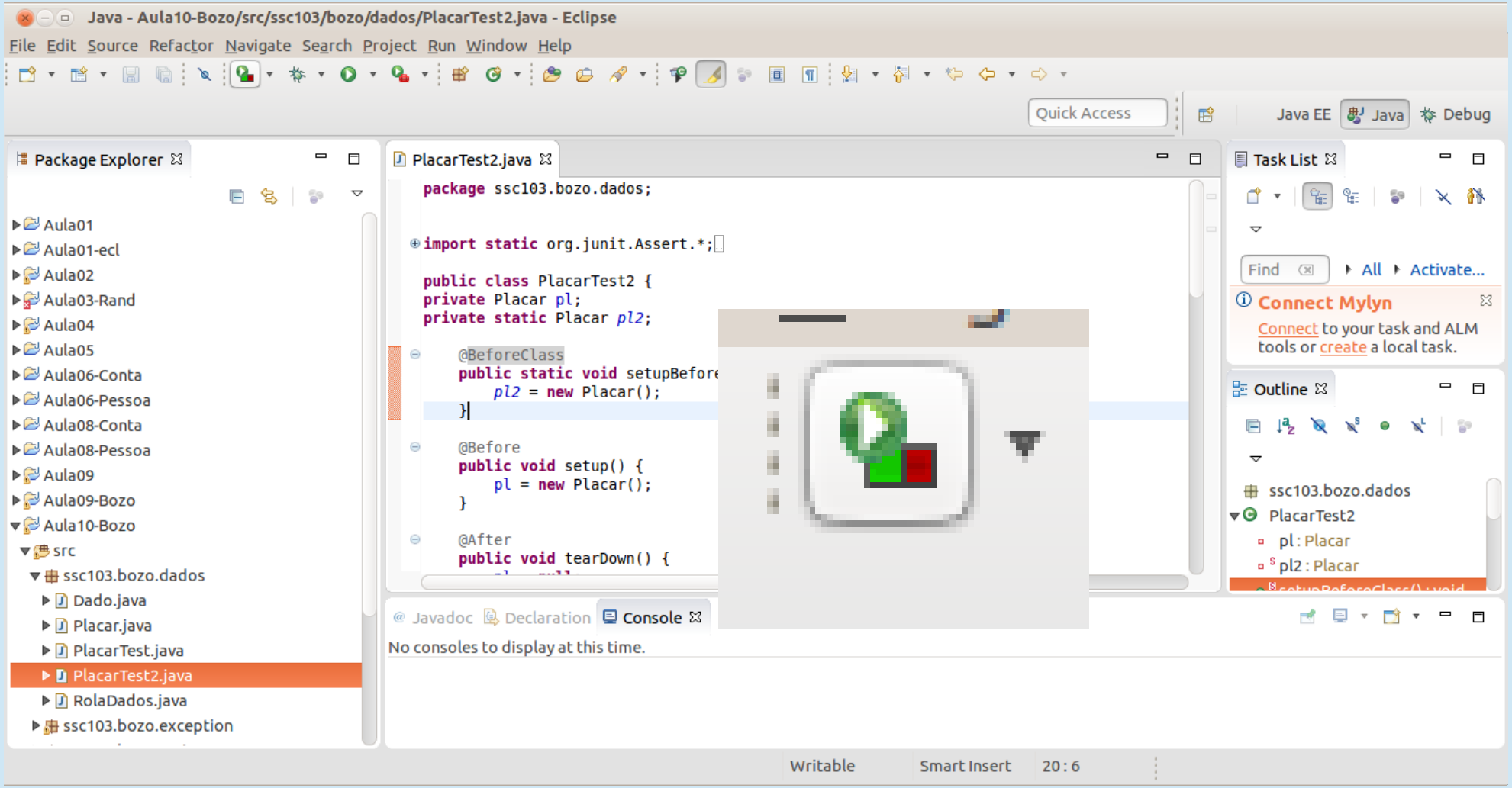
# Instalar


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

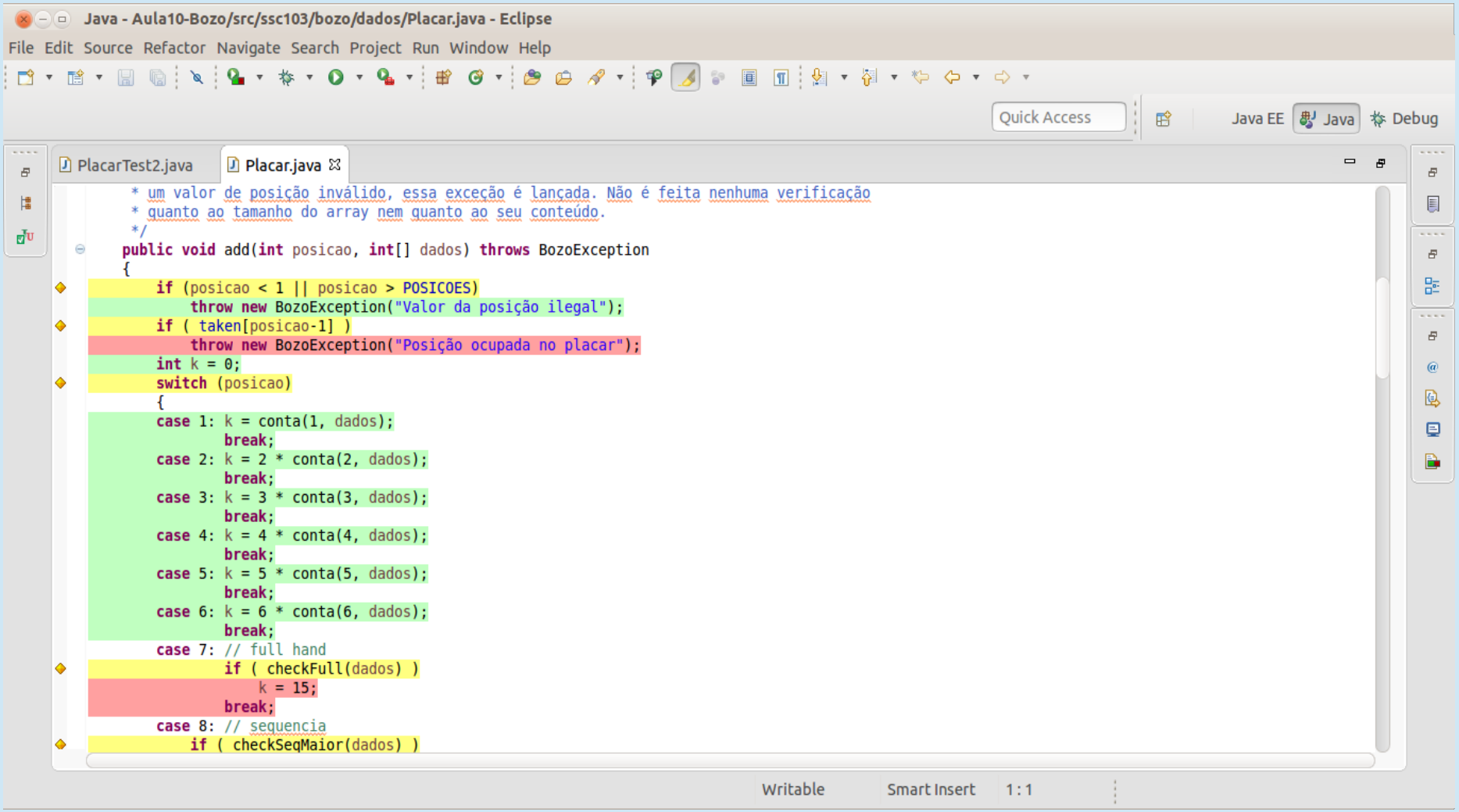
# Execução


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

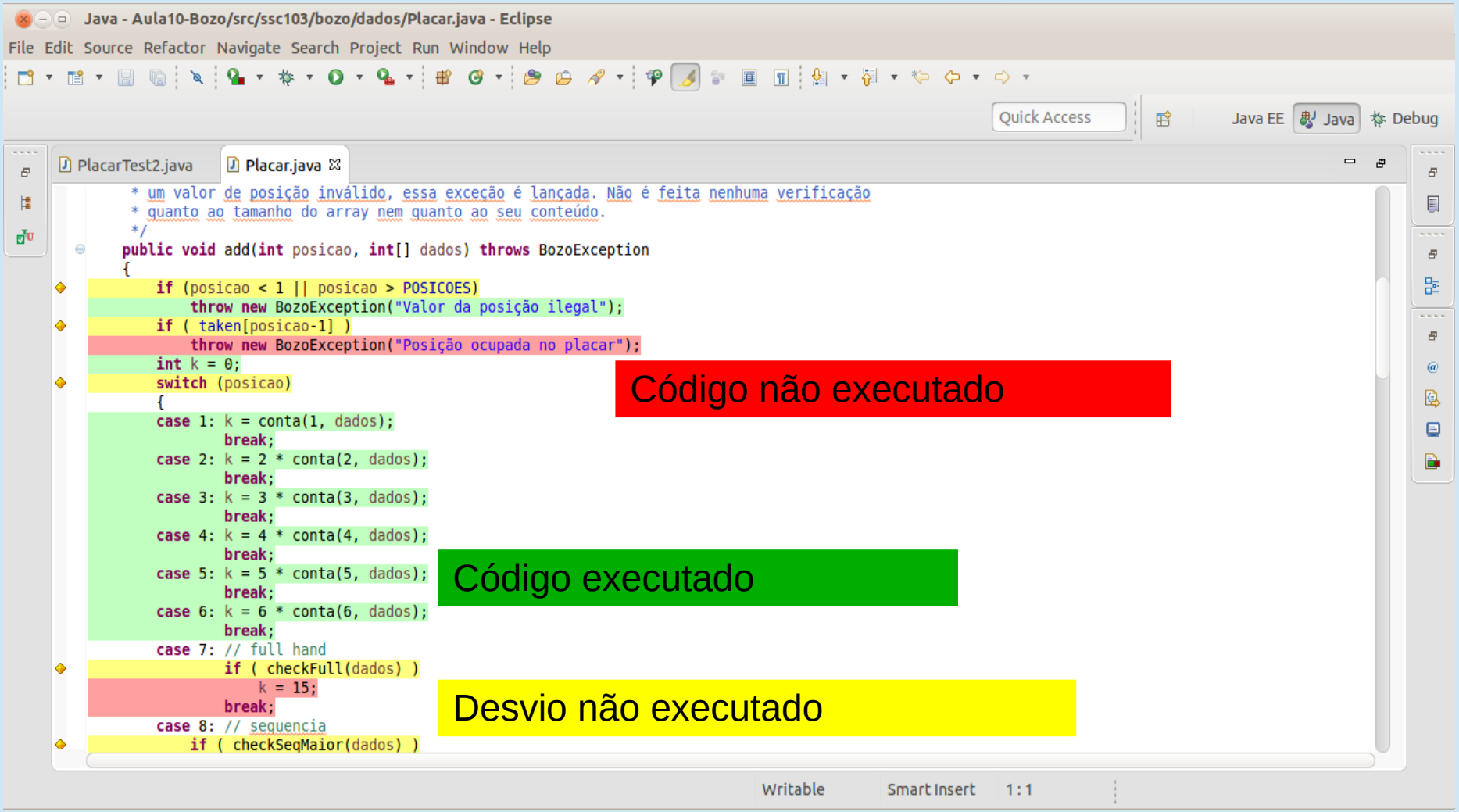
# Execução


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

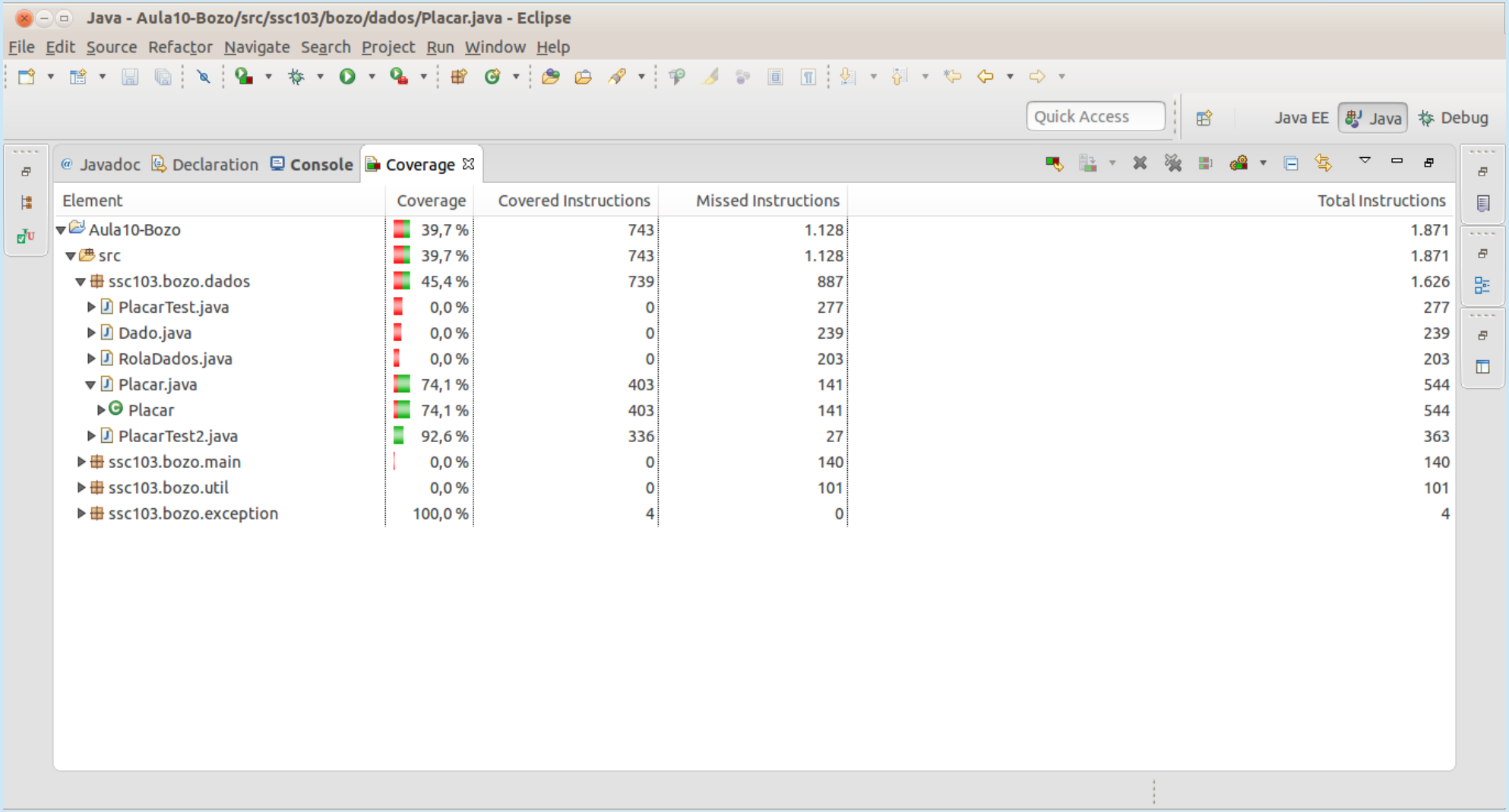
# Resultados


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

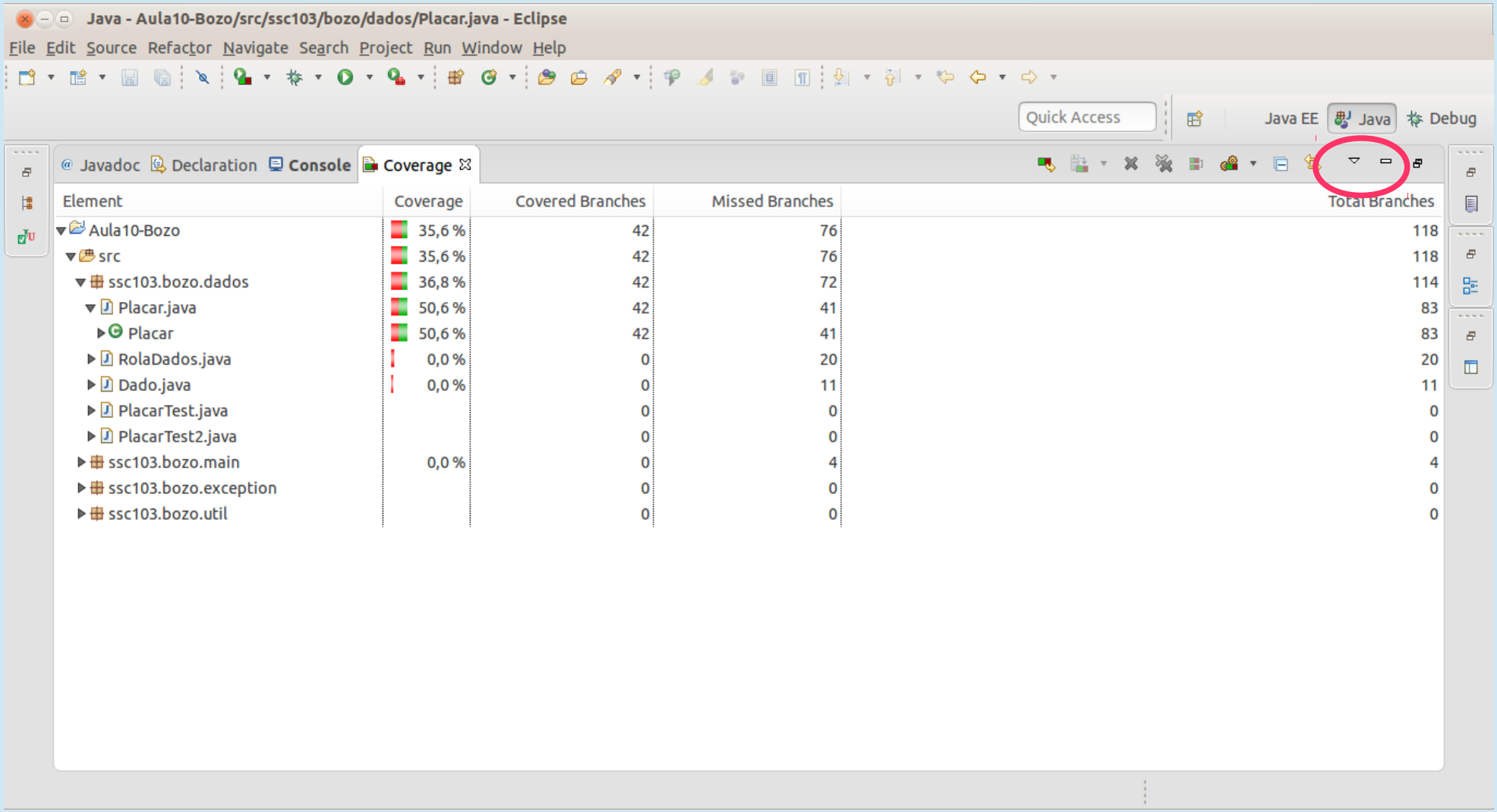

Java - Aula10-Bozo/src/ssc103/bozo/dados/Placar.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java EE
Java
Debug
PlacarTest2.java
Placar.java
* um valor de posição inválido, essa exceção é lançada. Não é feita nenhuma verificação
* quanto ao tamanho do 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 *Placar* até atingir 100% de cobertura de desvios para todos os métodos

# Exercício

- Complete os testes da classe *Placar* até atingir 100% de cobertura de desvios
- Faça o mesmo para a sua classe de processamento de imagens do Chain Codes
- Entregue o seu arquivo de teste junto com a implementação do Chain Codes