

An abstract, light blue geometric pattern consisting of interconnected lines and star-like shapes, resembling a complex network or a stylized molecular structure, is positioned in the upper left corner of the slide.

Interfaces e Unidades de Testes

Interfaces

```
public interface Drawable {  
    void draw();  
}
```

Operações públicas por omissão.

Operações apenas declaradas. Não se define qualquer método. Não é necessário usar o qualificador `abstract`.

```
public Square implements Drawable {  
    public void draw() {  
        ...  
    }  
}
```

Definição obrigatória

Interfaces

```
List<Drawable> figures = ...  
  
figures.add(new Square(3));  
figures.add(new Circle(4));  
figures.add(new Rectangle(3, 5));  
  
for(Drawable figure: figures)  
    figure.draw();
```

Interfaces genéricas

Interface genérica. T é um parâmetro. O correspondente argumento tem de ser um tipo.

```
public interface Comparable<T> {  
    int compareTo(T object);  
}
```

```
public interface Queue<E> {  
    E element();  
    void add(E e);  
    void remove();  
}
```

Nota: A interface queue é um pouco diferente na biblioteca do Java!

Implementação

```
public class Square implements Comparable<Square> {  
    public double area();  
    ...  
    public int compareTo(final Square another) {  
        if (area() > another.area())  
            return 1;  
        if (area() < another.area())  
            return -1;  
        return 0.0;  
        // porque não pode ser simplesmente:  
        // return area() - another.area(); ?  
    }  
    ...  
}
```

Tem de devolver:

- um número > 0 se a instância implícita for maior que o argumento,
- Um número < 0 se for menor
- 0 se forem iguais

Implementação

```
public class HeightComparator implements Comparator<Person> {  
    public int compare(final Person one, final Person another) {  
        if (one.height() > another.height())  
            return 1;  
        if (one.height() < another.height())  
            return -1;  
        return 0.0;  
    }  
    ...  
}
```

Tem de devolver:

- um número > 0 se a instância implícita for maior que o argumento,
- Um número < 0 se for menor
- 0 se forem iguais

Utilização

```
List<Square> squares = ...  
squares.add(new Square(4));  
squares.add(new Square(2));  
squares.add(new Square(3));  
squares.add(new Square(1));  
Collections.sort(squares);
```

{4, 2, 3, 1}

{1, 2, 3, 4}

Utilização

```
List<Person> persons = ...  
persons.add(new Person("João",177));  
persons.add(new Person("Maria",178));  
persons.add(new Person("Manel",173));  
persons.add(new Person("Sara",175));  
Collections.sort(persons, new HeightComparator());
```

{João, Maria, Manuel, Sara}

{Manel, Sara, João, Maria}

Implementação

```
public class FeetSizeComparator implements Comparator<Person> {  
    ...  
    public int compare(final Person one, final Person another) {  
        if (one.feetSize() > another.feetSize())  
            return 1;  
        if (one.feetSize() < another.feetSize())  
            return -1;  
        return 0.0;  
    }  
    ...  
}
```

Utilização

```
List<Person> persons = ...  
persons.add(new Person("João",39));  
persons.add(new Person("Maria",38));  
persons.add(new Person("Manel",41));  
persons.add(new Person("Sara",40));  
Collections.sort(persons, new HeightComparator());
```

{João, Maria, Manuel, Sara}

{Maria, João, Sara, Manel}

JUnit

Java - Rogue_V0_2/src/pt/iul/ista/poo/rogue/tests/PositionTest.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer JUnit

Finished after 0,03 seconds

Runs: 3/3 Errors: 1 Failures: 1

pt.iul.ista.poo.rogue.tests.PositionTest [Runner:

- testPlus (0,000 s)
- testConstructor1 (0,010 s)
- testConstructor2 (0,000 s)

Failure Trace

java.lang.AssertionError: expected: <2> but was: <3>
at pt.iul.ista.poo.rogue.tests.PositionTest.testConstr

```
package pt.iul.ista.poo.rogue.tests;

import org.junit.Assert;

public class PositionTest {

    private Position position = new Position(2, 3);

    @Before
    public void prepare() {

    }

    @Test
    public void testConstructor1() {
        Assert.assertEquals(2, position.getX());
    }

    @Test
    public void testConstructor2() {
        Assert.assertEquals(3, position.getY());
    }

    @Test
    public void testPlus() {
        Assert.assertEquals(new Position(5, 5), position.plus(new Vector2D(3, 2)));
        Assert.assertEquals(new Position(0, 0), position.plus(new Vector2D(-2, -3)));
    }

}
```

Anotações JUnit

@Test Método que contém um teste

@Before

@After Métodos (public void) a executar sempre antes / depois de um teste da classe

@BeforeClass

@AfterClass Métodos (public static void, e sem argumentos) a executar uma vez antes / depois dos restantes testes da classe

@ignore Método a ignorar nos testes

@Test(expected= ...Exception.class) Método que deve falhar lançando a exceção indicada

@Test(timeout=100) Método que deve falhar caso não tenha um resultado ao fim de 100ms

Principais métodos e classes

- Métodos static da classe Assert (verificar condições que devem ser verdadeiras para que o teste tenha sucesso): `assertTrue`, `assertFalse`, `assertEquals`, `assertSame`, `assertArrayEquals`, `assertNull`, `assertNotNull`
- Métodos static da classe Assume (verificar pré-condições)

Referências

- Y. Daniel Liang, *Introduction to Java Programming*, 7.^a edição, Prentice-Hall, 2010.
- Junit (<http://junit.org/>)