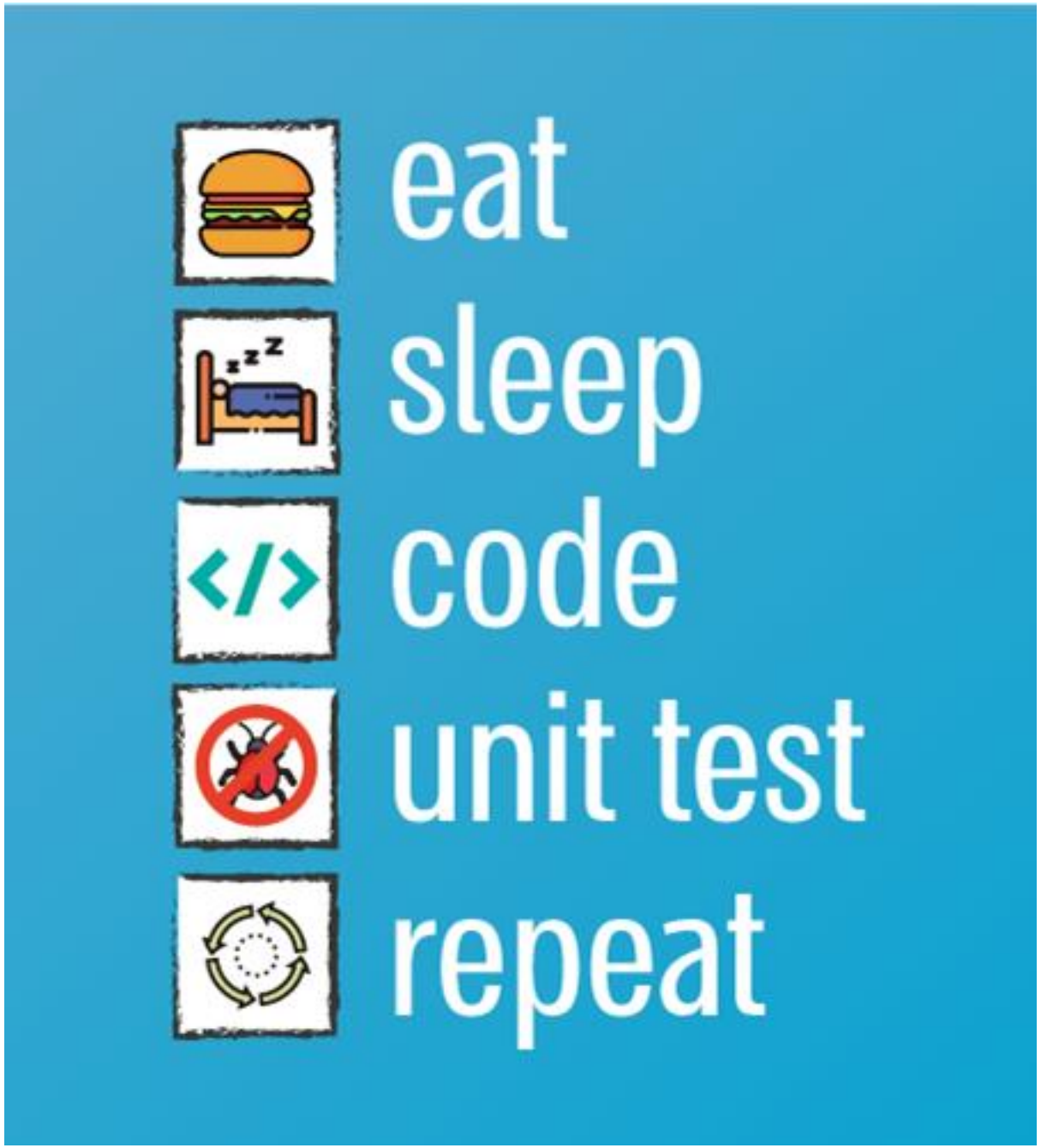
Teste de Software
Aula 01

JUnit


eat
sleep
code
unit test
repeat

Testes Manuais x Testes Automatizados

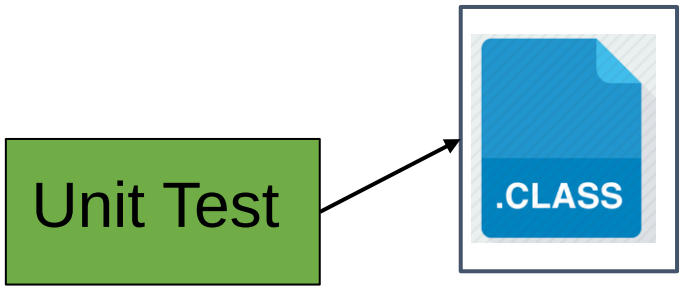
- Vimos na aula passada como fazer testes manuais com maior produtividade: utilizando o depurador da IDE
- Entretanto, se você quiser produzir código com bem menos erros, maior qualidade, reduzindo o tempo/custo de manutenção do software, e preservar a saúde do ser corpo e mente :-) ...

FAÇA TESTES AUTOMATIZADOS!

Testes de Unidade

Um teste de unidade não se preocupa com todo o sistema; ele está interessado apenas em **verificar se uma pequena parte dele funciona**.

Um teste de unidade testa uma única unidade do nosso sistema. Geralmente, em sistemas orientados a objetos, essa unidade é a **classe**.



Cenário de Exemplo: compras online

1. Seleciona produto -> carrinho de compras.
2. Finaliza a compra:
   a. Sistema fala com a operadora de cartão;
   b. Retira o produto do estoque;
   c. Dispara um evento p/ equipe de logística;
   d. Envia e-mail confirmando a compra.

Cenário de Exemplo: compras online

Software complexo.
Regras de negócio:

- Carrinho de compras
- Tipo de pagamento;
- Fechamento da compra.

Toda essa lógica em um único arquivo? Uma única classe?

De forma nenhuma! Not at all!!!

Cenário de Exemplo: compras online

Várias classes:

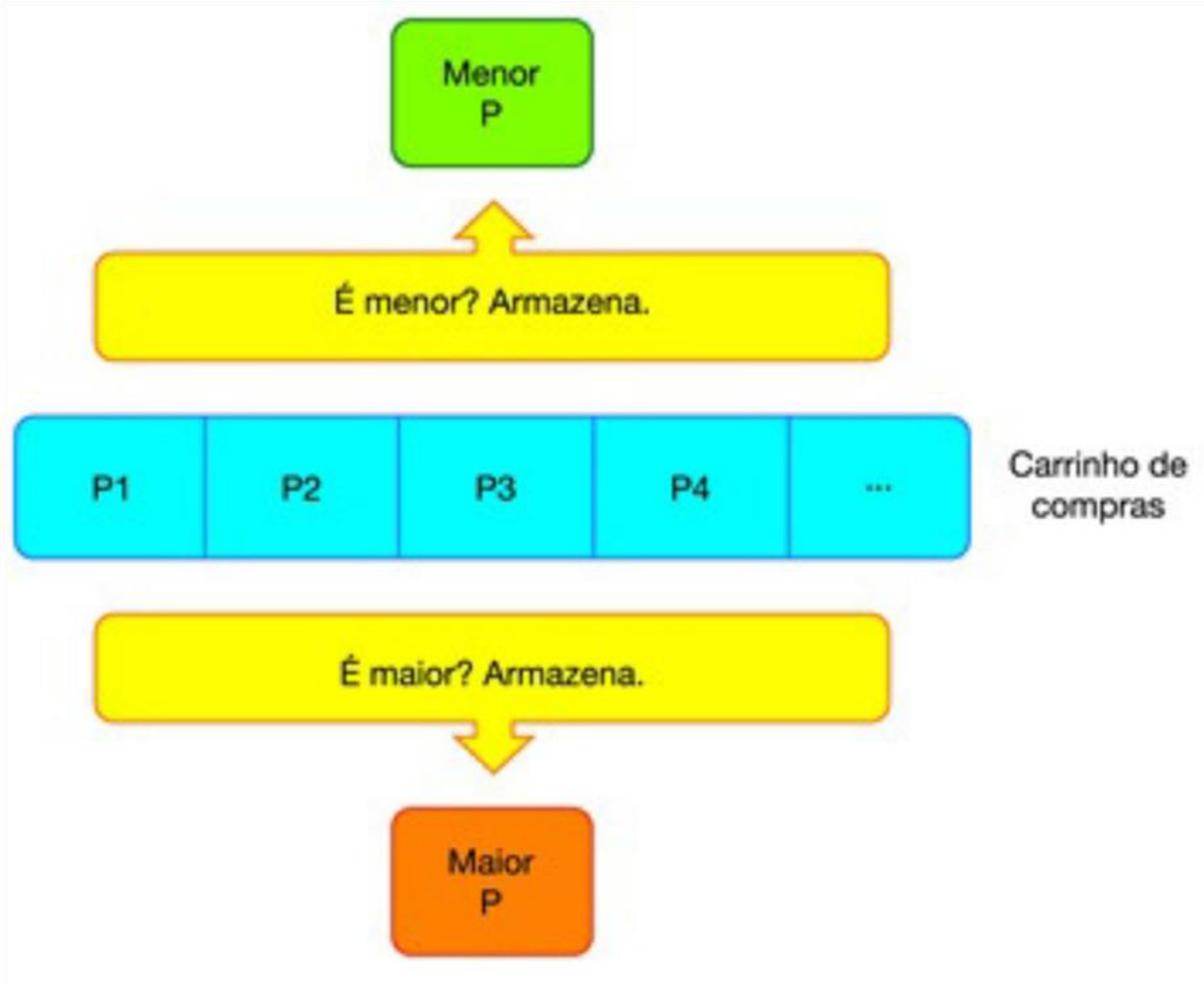
- CarrinhoDeCompras
- Pedido
- etc.

A ideia é termos baterias de testes **para cada uma** dessas classes.

Cenário de Exemplo: compras online

Essa mesma loja virtual precisa encontrar, dentro do seu carrinho de compras, os produtos de maior e menor valor.

Um possível algoritmo para esse problema seria percorrer a lista de produtos no carrinho, comparar um a um, e guardar sempre a referência para o menor e o maior produto encontrado até então.


Menor
P
É menor? Armazena.
P1
P2
P3
P4
...
Carrinho de compras
É maior? Armazena.
Maior
P

```
public class MaiorEMenor {

    private Produto menor;
    private Produto maior;

    public void encontra(CarrinhoDeCompras carrinho) {
        for(Produto produto : carrinho.getProdutos()) {
            if(menor == null ||
                produto.getValor() < menor.getValor()) {
                menor = produto;
            }
            else if (maior == null ||
                    produto.getValor() > maior.getValor()) {
                maior = produto;
            }
        }
    }

    public Produto getMenor() {
```

Exercício:

1) Implemente as classes Produto, CarrinhoDeCompras e MaiorMenor

Testando a classe MaiorEMenor

```
public class TestaMaiorEMenor {
    public static void main(String[] args) {
        CarrinhoDeCompras carrinho = new CarrinhoDeCompras();
        carrinho.adiciona(new Produto("Liquidificador", 250.0));
        carrinho.adiciona(new Produto("Geladeira", 450.0));
        carrinho.adiciona(new Produto("Jogo de pratos", 70.0));

        MaiorEMenor algoritmo = new MaiorEMenor();
        algoritmo.encontra(carrinho);

        System.out.println("O menor produto: " +
            algoritmo.getMenor().getNome());
        System.out.println("O maior produto: " +
            algoritmo.getMaior().getNome());
    }
}
```

Exercício:

1) Execute o teste.
2) A saída do método está correta?
3) Faça outros testes.

Testando a classe MaiorEMenor (outra vez)

Se você testou bem, percebeu que o algoritmo do método `encontra(...)` possui um bug.

Considere o cenário de teste abaixo:

```
CarrinhoDeCompras carrinho = new CarrinhoDeCompras();
carrinho.adiciona(new Produto("Geladeira", 450.0));
carrinho.adiciona(new Produto("Liquidificador", 250.0));
carrinho.adiciona(new Produto("Jogo de pratos", 70.0));
```

Foi alterada apenas a ordem de inserção!

Qual a saída do programa?

Encontrando o bug

O seguinte erro ocorre:

```
O menor produto: Jogo de pratos
Exception in thread "main" java.lang.NullPointerException
        at testeUnidade.TestaMaiorEMenor.main(TestaMaiorEMenor.java:17)
```

Consegue identificar onde está o bug?

Que tal depurar o código?

Exercício:

1) Depure o código e tente achar onde está o bug.

Encontrando o bug

BUG: Se os produtos, por algum motivo, forem adicionados no carrinho em ordem decrescente, a classe não consegue calcular corretamente.

Será que o desenvolvedor, ao escrever a classe, perceberia esse bug? Será que ele faria todos os testes necessários para garantir que a classe realmente funcione?

Consequências do bug

Imagine se a loja virtual colocasse esse código em produção.

Quantas compras seriam perdidas por causa desse problema?

Conclusão:

Para diminuir a quantidade de bugs levados para o ambiente de produção, é necessário testar o código constantemente.

Mas fazer isso manualmente gasta muito tempo...

Solução: Testes Automatizados

Faça o computador trabalhar para você! Ensine-o como testar seu programa.

Como isso é possível?

Elaborando um teste automático

Usuário está comprando dois produtos.

Usuário vai ao carrinho de compras

Usuário verifica se os dois produtos estão no carrinho.

Um teste automático **descreve um cenário**, **executa uma ação** e **valida uma saída**.

Convertendo o teste manual em automático

```
public class TestaMaiorEMenor {
    public static void main(String[] args) {
        CarrinhoDeCompras carrinho = new CarrinhoDeCompras();
        carrinho.adiciona(new Produto("Liquidificador", 250.0));
        carrinho.adiciona(new Produto("Geladeira", 450.0));
        carrinho.adiciona(new Produto("Jogo de pratos", 70.0));

        MaiorEMenor algoritmo = new MaiorEMenor();
        algoritmo.encontra(carrinho);

        System.out.println("O menor produto: " +
            algoritmo.getMenor().getNome());
        System.out.println("O maior produto: " +
            algoritmo.getMaior().getNome());
    }
}
```

Cenário

Ação

Validação (manual…)

Convertendo o teste manual em automático (JUnit)

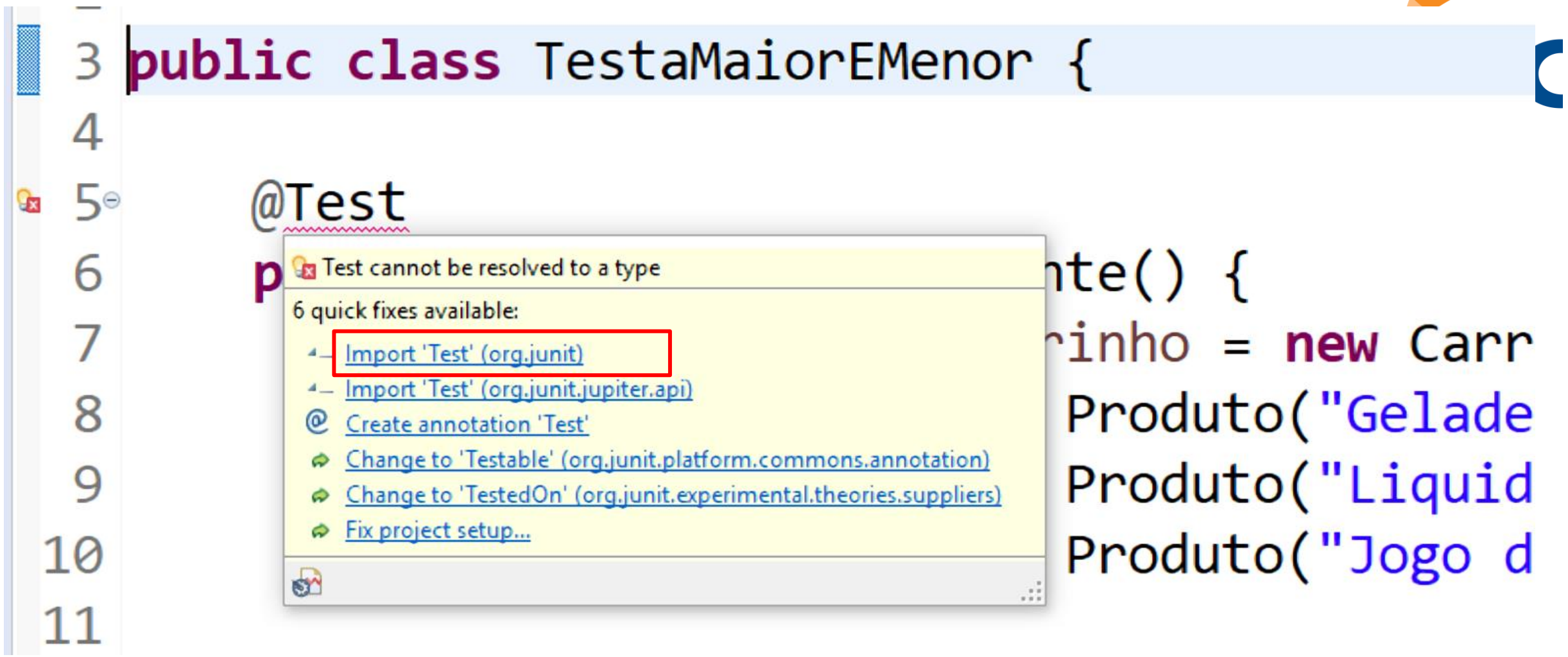
Para tornar todo o processo automático devemos utilizar um **framework de testes**.

O mais famoso framework de testes de unidade para o Java é o **JUnit**: https://junit.org (versão atual: 5)

Vejamos como utilizar o JUnit nesse exemplo.

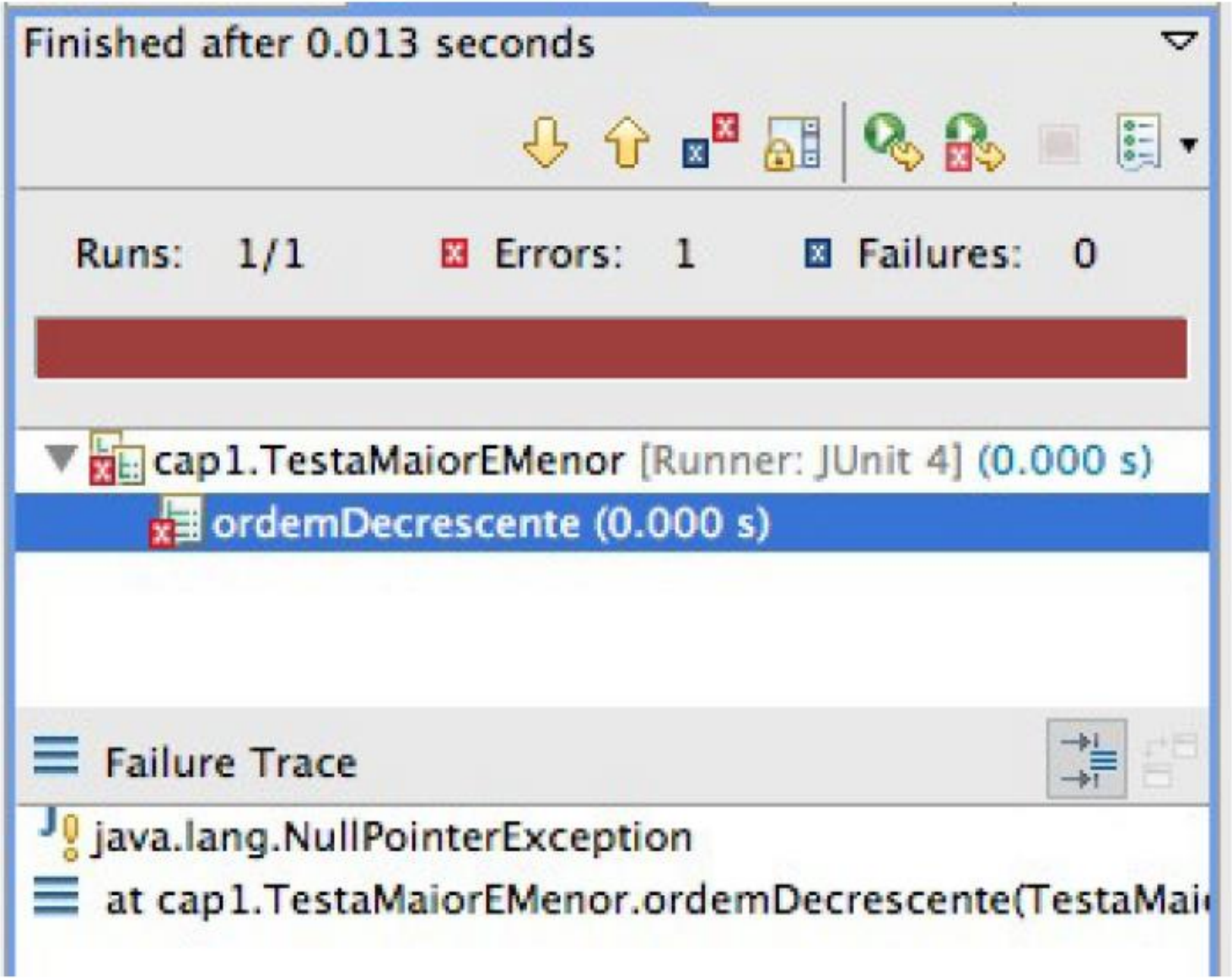
OBS: O Eclipse já vem com o JUnit integrado (plugin).

```java
import org.junit.Assert;
import org.junit.Test;

public class TestaMaiorEMenor {

    @Test
    public void ordemDecrescente() {
        CarrinhoDeCompras carrinho = new CarrinhoDeCompras();
        carrinho.adiciona(new Produto("Geladeira", 450.0));
        carrinho.adiciona(new Produto("Liquidificador", 250.0));
        carrinho.adiciona(new Produto("Jogo de pratos", 70.0));

        MaiorEMenor algoritmo = new MaiorEMenor();
        algoritmo.encontra(carrinho);

        Assert.assertEquals("Jogo de pratos", algoritmo.getMenor().getNome());
        Assert.assertEquals("Geladeira", algoritmo.getMaior().getNome());
    }
}
```


3 public class TestaMaiorEMenor {
4
5 @Test
6 p
7
8
9
10
11
Test cannot be resolved to a type
6 quick fixes available:
Import 'Test' (org.junit)
Import 'Test' (org.junit.jupiter.api)
Create annotation 'Test'
Change to 'Testable' (org.junit.platform.commons.annotation)
Change to 'TestedOn' (org.junit.experimental.theories.suppliers)
Fix project setup...
nte() {
rinho = new Carr
Produto("Gelade
Produto("Liquid
Produto("Jogo d

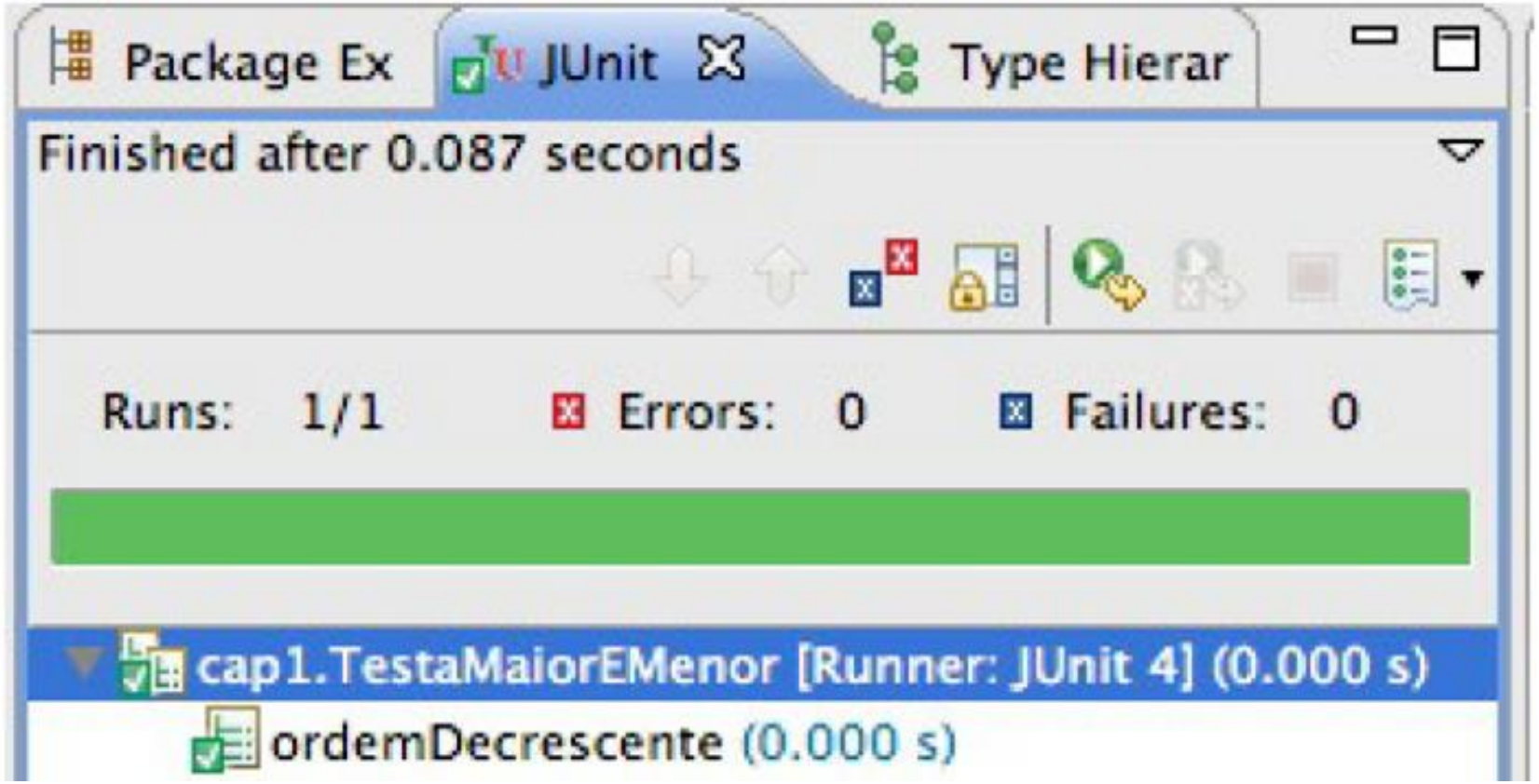
```java
import org.junit.Assert;
import org.junit.Test;

public class TestaMaiorEMenor {

    @Test
    public void ordemDecrescente() {
        CarrinhoDeCompras carrinho = new CarrinhoDeCompras();
        carrinho.adiciona(new Produto("Geladeira", 450.0));
        carrinho.adiciona(new Produto("Liquidificador", 250.0));
        carrinho.adiciona(new Produto("Jogo de pratos", 70.0));

        MaiorEMenor algoritmo = new MaiorEMenor();
        algoritmo.encontra(carrinho);

        Assert.assertEquals("Jogo de pratos", algoritmo.getMenor().getNome());
        Assert.assertEquals("Geladeira", algoritmo.getMaior().getNome());
    }
}
```

Exercício: faça essas alterações e execute a classe de testes


Finished after 0.013 seconds
Runs: 1/1
Errors: 1
Failures: 0
cap1.TestaMaiorEMenor [Runner: JUnit 4] (0.000 s)
ordemDecrescente (0.000 s)
Failure Trace
java.lang.NullPointerException
at cap1.TestaMaiorEMenor.ordemDecrescente(TestaMai

Corrigindo o bug e executando o teste novamente

Remover o "else" irá resolver nosso problema:



Criando mais testes

Ainda são necessários muitos testes para garantir que a classe MaiorEMenor funcione corretamente. Até agora, o único cenário testado são produtos em ordem decrescente. Muitos outros cenários precisam ser testados. Dentre eles:

I. Produtos com valores em ordem crescente;
II. Produtos com valores em ordem variada;
III. Um único produto no carrinho.

Exercício: crie outros três métodos que testem esses três cenários.

Vantagens de usar teste automatizado

Ganho de tempo

Software com menos erros => mais qualidade

Segurança ao fazer mudanças no código (testes de regressão)

Lembre-se que algum dia outro programador irá trabalhar em cima do seu código e vice-versa. O que é simples para você, pode não ser para ele, e vice-versa.

Referências

Mauricio Aniche. Test-Driven Development: Teste e Design no Mundo Real. Casa do Código, 2014.