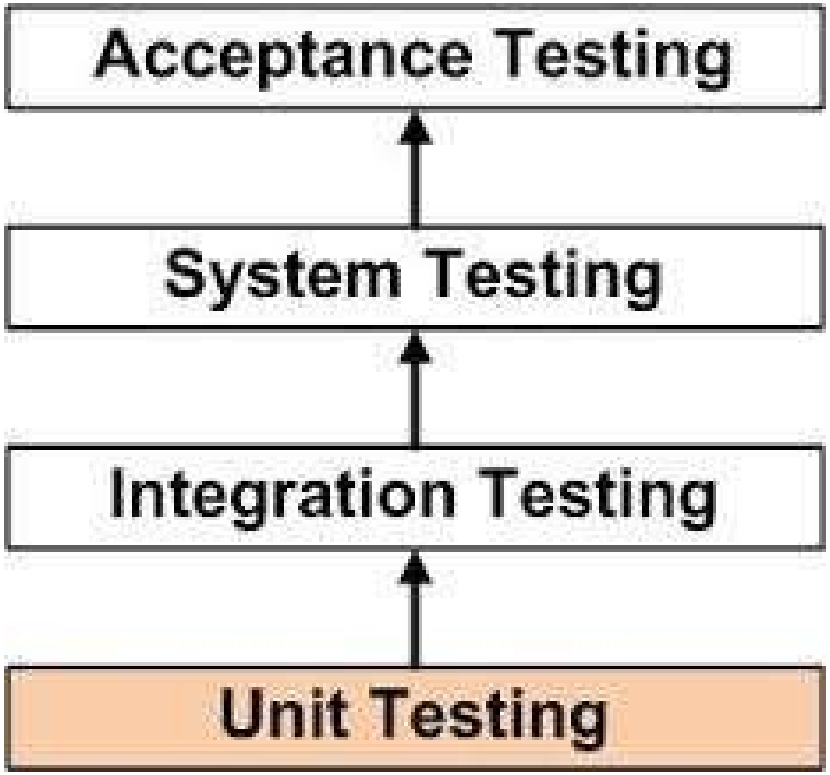
Testes Unitários

Delivoro Sistemas

Testes Unitários

- O que são Testes Unitários?
  - É um nível de teste de software no qual componentes individuais são testados (métodos)
  - O propósito é validar se cada unidade do software executa como esperado

Testes Unitários



Testes Unitários

- Os testes unitários tem como benefício:
  - Garantir que problemas serão descobertos cedo.
  - Facilitar a manutenção de código
  - Servir como documentação
  - **Ajudam a melhorar o design do seu código e torna-lo um melhor desenvolvedor**

Testes Unitários

```
public class Conta {

    private Double saldo = 0.0d;

    public Conta(Double saldo){
        this.saldo = saldo;
    }

    public boolean temSaldo(){
        return saldo > 0;
    }

}
```

- Como escrever um teste para essa classe???

Testes Unitários

- Criando método para testar conta com saldo

```
public class ContaTest {

    public void testaContaComSaldo() {
        Conta c = new Conta(10.0);
        if(c.temSaldo())
            System.out.println("Ok");
        else
            System.out.println("Erro");
    }

    public static void main(String[] args) {
        new ContaTest().testaContaComSaldo();
    }
}
```

```
<terminated> ContaTest [Java Application]
OK
```

Testes Unitários

- Pronto???
- A classe está testada ????

Testes Unitários

- Criando método para testar conta sem saldo

```
public class ContaTest {

    public void testaContaComSaldo() {
        Conta c = new Conta(10.0);
        if(c.temSaldo())
            System.out.println("OK");
        else
            System.out.println("Erro");
    }

    public void testaContaSemSaldo() {
        Conta c = new Conta(0.0);
        if(c.temSaldo())
            System.out.println("Erro");
        else
            System.out.println("OK");
    }

    public static void main(String[] args) {
        new ContaTest().testaContaComSaldo();
        new ContaTest().testaContaSemSaldo();
    }
}
```

Testes Unitários

- E agora???
- A classe está testada ????

Testes Unitários

- E se alguém criar uma conta passando nulo???

```
public void testaContaComSaldoNulo() {
     Conta c = new Conta(null);
     if (c.temSaldo())
          System.out.println("Erro");
     else
          System.out.println("OK");
}
```

```
Exception in thread "main" java.lang.NullPointerException
        at br.ufrn.sinfo.Conta.temSaldo(Conta.java:20)
        at br.ufrn.sinfo.ContaTest.testaContaComSaldoNulo(ContaTest.java:32)
        at br.ufrn.sinfo.ContaTest.main(ContaTest.java:41)
```

Testes Unitários

- Alterando a classe conta para passar no 3º teste

```
public class Conta {

    private Double saldo = 0.0d;

    public Conta(Double saldo){
        this.saldo = saldo;
    }

    public boolean temSaldo(){
        return saldo != null && saldo > 0;
    }

}
```

Testes Unitários

```
public class Conta1Test {

    public void testaContaComSaldo() {
        Conta1 c = new Conta1(10.0);
        if(c.temSaldo())
            System.out.println("OK");
        else
            System.out.println("Erro");
    }

    public void testaContaSemSaldo() {
        Conta1 c = new Conta1(0.0);
        if(c.temSaldo())
            System.out.println("Erro");
        else
            System.out.println("OK");
    }

    public void testaContaComSaldoNulo() {
        Conta1 c = new Conta1(null);
        if (c.temSaldo())
            System.out.println("Erro");
        else
            System.out.println("OK");
    }

    public static void main(String[] args) {
        new Conta1Test().testaContaComSaldo();
        new Conta1Test().testaContaSemSaldo();
        new Conta1Test().testaContaComSaldoNulo();
    }
}
```

```
OK
OK
OK
```

Os três testes passaram

- **Qualquer alteração na classe Conta, você terá mais confiança de que o comportamento do método temSaldo será mantido!!!**

Testes Unitários

- Algumas observações:
  - Para ter certeza de que o seu código não tem erro, deve-se testar todas as possíveis entradas do método
  - Para a variável saldo:
    - 4.94065645841246544e-324d a 1.79769313486231570e+308d
  - Geralmente, se testa alguns valores chaves que representam todas as possíveis entradas:
    - -1000, -1, 0, null, +1, +1000
  - **Tenha criatividade para imaginar as possiblidades de testes!!!**

Testes Unitários

- Não teste métodos triviais, tipo **get** e **set** padrões, só se esses métodos tiverem validações.

- Achou um bug? Não conserte sem antes escrever um teste que o pegue
- Preocupe-se com a qualidade do código dos seus testes, mantenha igual ao código das classes que estão sendo testadas, pois você vai precisar mantê-los do mesmo jeito.

Testes Unitários

- Para esse simples exemplo:
  - Classe Conta: 13 linhas de código
  - Classe Conta Teste: 31 linhas de código

- O número de linhas de código para se testar um sistema geralmente é maior que o número de linhas de código do sistema

JUnit

- Framework que facilita o desenvolvimento e execução de testes unitários em código JAVA.
  - Facilita a criação, execução automática de testes e a apresentação dos resultados

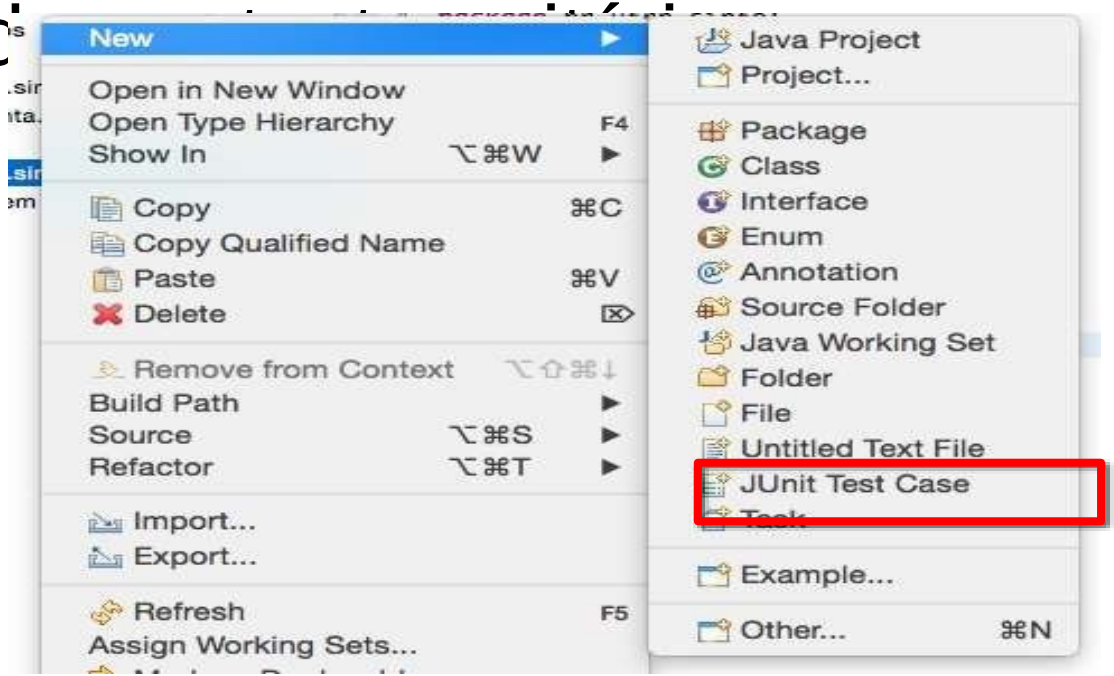
Junit

Junit

- Vamos criar o mesmo teste da classe Conta usado o JUnit

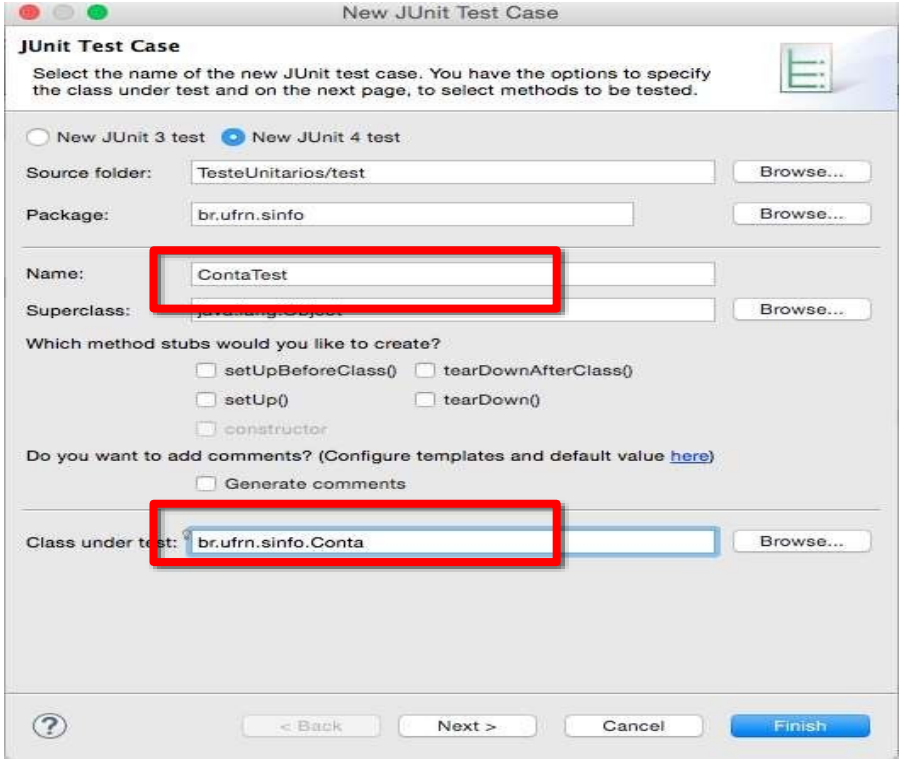
JUnit

- Criand
  JUnit



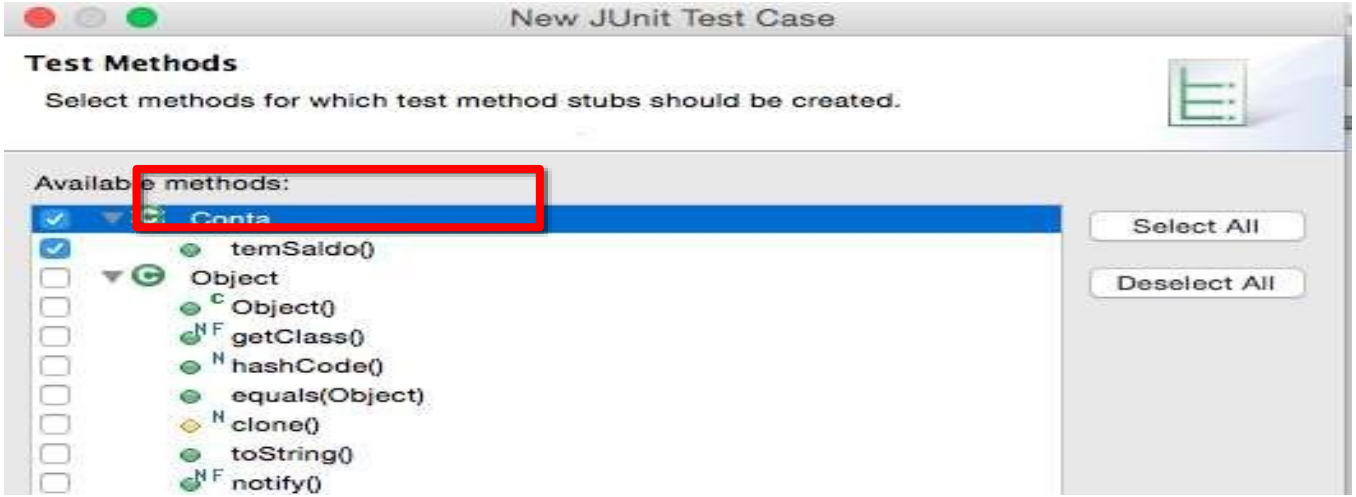
JUnit

- Criando um teste unitário com Junit
  - Defina o nome da classe
  - E a classe que vai ser testada



JUnit

- Criando um teste unitário com JUnit
  - Escolha os métodos a serem testados



JUnit

- Criando um teste com JUnit

```
public class ContaTest2 {

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

}
```

A partir da Versão 4 do JUnit todo método com um @Test é um teste unitário a ser executado

Substitui o método main()

JUnit

- Criando um teste com JUnit

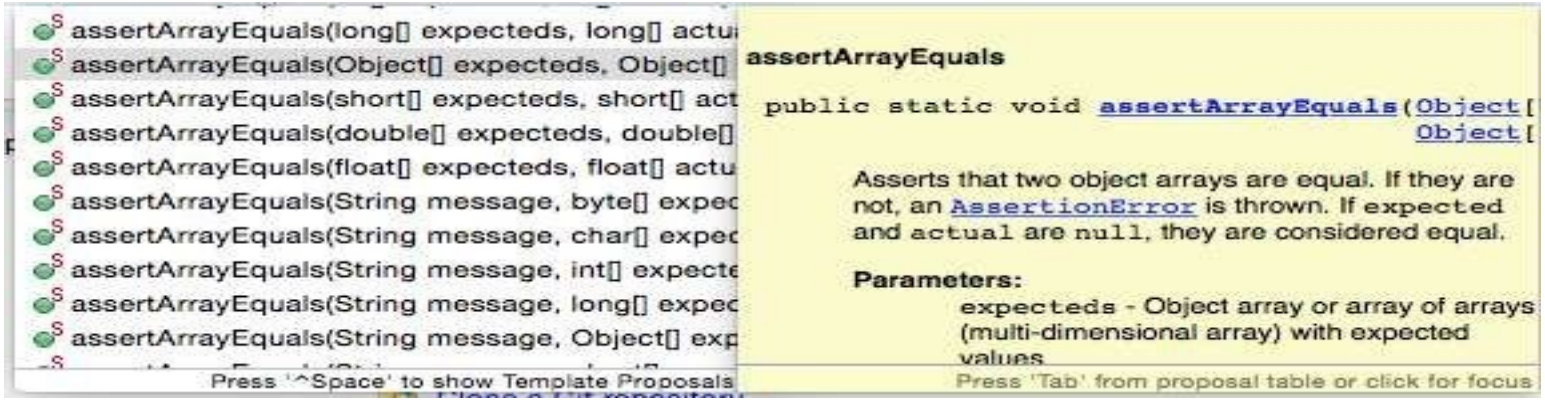
```
public class ContaTest2 {

    @Test
    public void testaContaComSaldo() {
        Conta c = new Conta(10.0);
        assertTrue(c.temSaldo());
    }

    @Test
    public void testaContaSemSaldo() {
        Conta c = new Conta(0.0);
        assertFalse(c.temSaldo());
    }

    @Test
    public void testaContaComSaldoNulo() {
        Conta c = new Conta(null);
        assertFalse(c.temSaldo());
    }
}
```

Junit possui vários métodos “assertions”

Substitui os testes feitos anteriormente “if ... else .. println()”

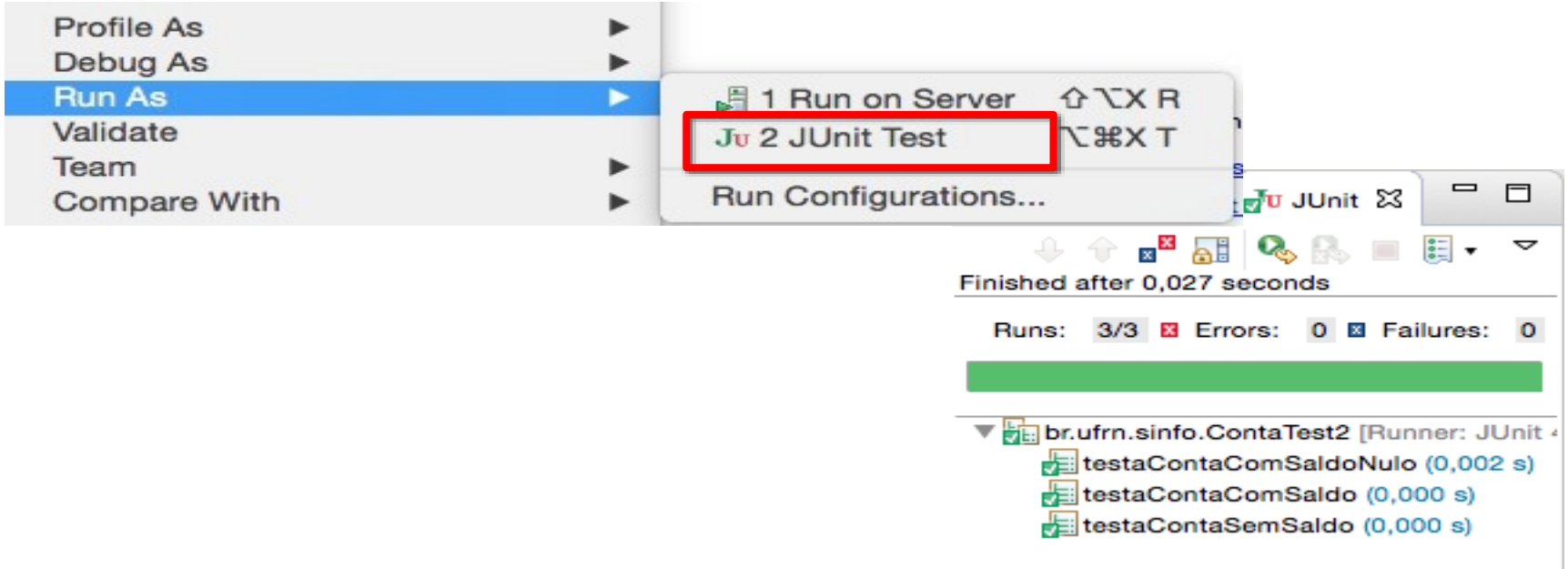
JUnit

- Criando um teste com Junit
  - Existem vários métodos assertX, para validar os



JUnit

- Criando um teste com Junit



Comandos Assert Mais usados

fail(String)	Faz o método falhar
assertTrue(boolean, condition)	Testa se a condição é verdadeira
assertFalse(boolean, condition)	Testa se a condição é falsa
assetEquals(expected, actual)	Teste se dois valores são iguais
assertEquals(expected, actual, tolerance)	Para float ou double, a tolerância é o número de casas decimais a ser verificado
assertNull(object)	Verifica se o objeto é nulo
assertNotNull(object)	Verifica se o objeto não é nulo

Junit – Exercício 1

- Faça testes unitários para a classe ValidaCPF método isCPF() disponível em:
- https://pastebin.com/fA6DmDG4
- O objetivo é melhorar o código e torná-lo mais robusto e a prova de erros inesperados
- Pense em todos os tipos de entrada
  - CPFs válidos
  - CPFs inválidos
    - Explore tamanho, caracteres, etc
  - CPFs nulos