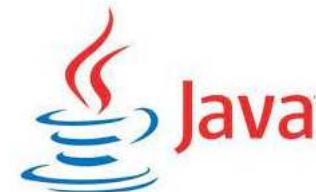




Residência em Tecnologia da Informação e Comunicação

TDD na prática

Professor:
Alvaro Degas Coelho



INSTITUIÇÃO EXECUTORA



COORDENADORA



APOIO



Um exemplo (didático)

- Implementação do jogo de Senha (Mastermind)
- Um “alfabeto”
 - Quais caracteres podem ser usados
- Uma senha
 - Qual sequência de caracteres deve ser descoberta
- Um feedback
 - Quais no lugar certo, quais no lugar errado, quais não estão na senha
- Um limite de tentativas





Iniciando o Desenvolvimento

- Três funcionalidades principais
 - Configurar o Jogo
 - Inserir uma senha
 - Jogar (tentar descobrir a senha)
- Desenvolver usando TDD
 - Antecipar situações de erro



Residência
em Software

Configurar o Jogo

- Solicita que o nome da configuração (podem haver várias)
- Solicita que o alfabeto de caracteres válidos seja inserido
- Solicita que o tamanho da senha seja definido
- Solicita o número de tentativas que o jogador deverá ter
- Armazena tudo isso num arquivo json

Classe Configuração

- Para se definir uma configuração
 - Definir o alfabeto
 - O que pode dar errado?
 - Definir o tamanho da senha
 - O que pode dar errado?
 - Definir o máximo de tentativas
 - O que pode dar errado?
 - Toda configuração cadastrada deve ser salva num arquivo JSON
 - O que pode dar errado?

Configuração
Nome
Alfabeto
TamanhoSenha
MaxTentativas
getNome e setNome
definirAlfabeto()
setTamanhoSenha()
setMaxTentativas()

Definir o Alfabeto

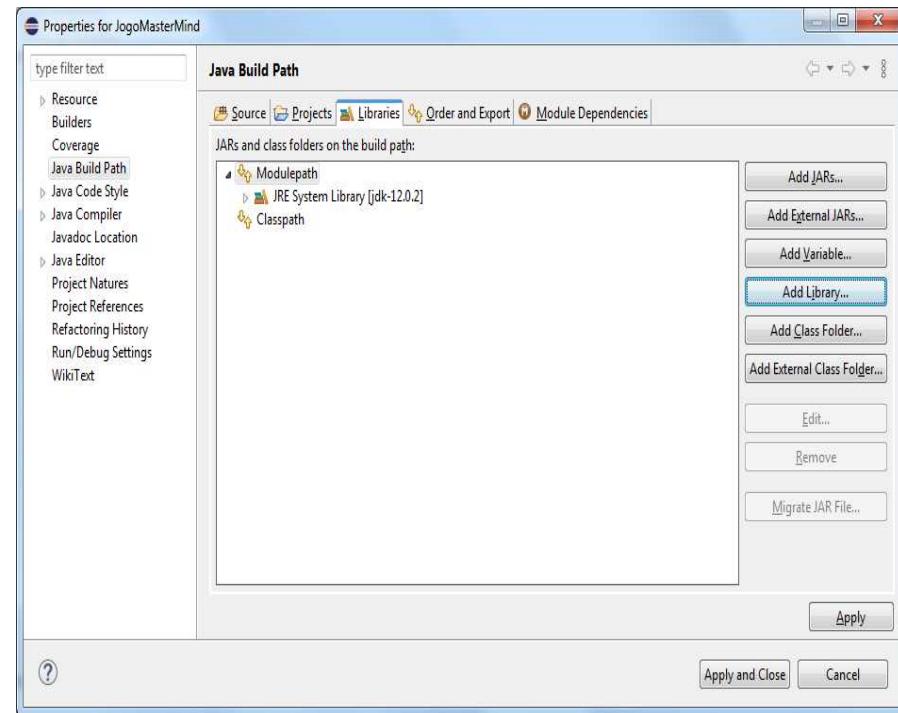
- O que pode dar errado?
 - O alfabeto não pode ser vazio
 - Não pode haver repetição de caracteres
- O que precisamos testar?
 - Se o programa registra um alfabeto corretamente
 - Se o programa proíbe que um alfabeto seja vazio (nem de apenas um caracter)
 - Se o programa proíbe que o alfabeto tenha caracteres repetidos

Começando a programar

- Criar o projeto (Mastermind? Senha?), o package para as configurações e duas classes
 - Classe Configuracao e classe ListaConfiguracoes
- Para começar vamos programar o método para definir o alfabeto de uma configuração
 - O alfabeto será um atributo do tipo String
 - Generate getters and setters
- Atenção no setAlfabeto!
 - Precisaremos programar bastante aqui

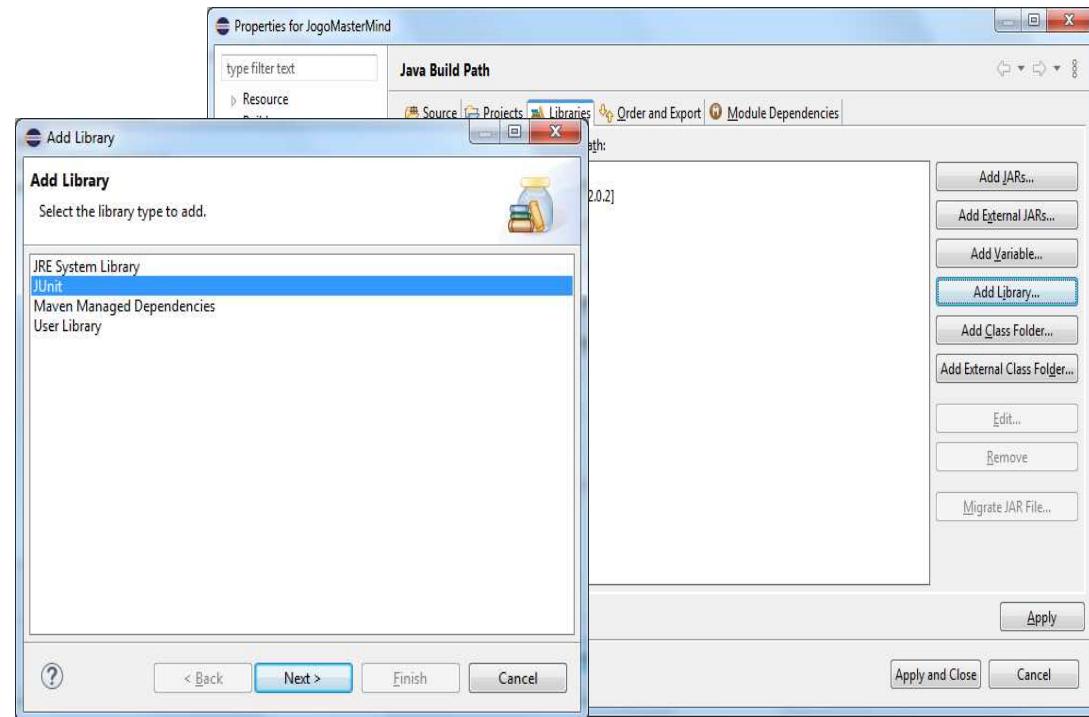
Preparando o ambiente JUNIT

- JUNIT é um *framework* com ferramentas que auxiliam a produção de testes automáticos.
- ATENÇÃO: um bug no Eclipse pode impedir que o Junit seja instalado automaticamente
- Instalação Manual
- Projetos -> Propriedades -> (aba) Libraries
 - AddLibrary



Preparando o ambiente JUNIT

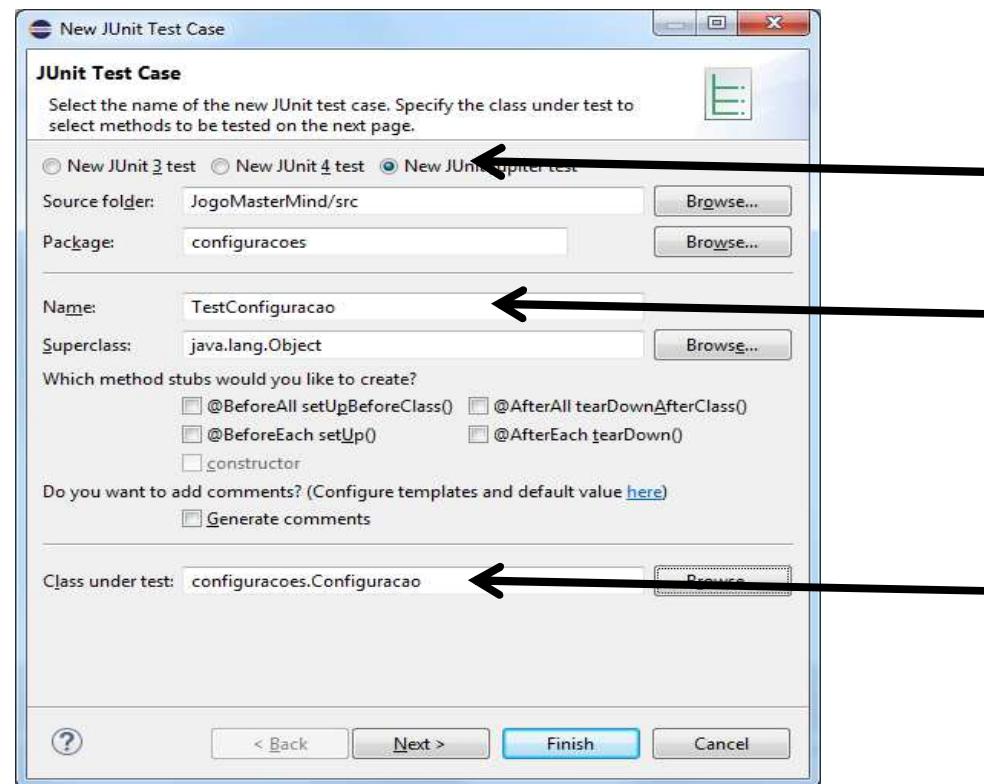
- Seleccione Junit
- Seleccione Junit 5
- Apply and Close



O primeiro teste de unidade

- Agora vamos criar nosso primeiro teste
- Para isto, vamos criar uma *Classe de Teste*
 - Tipicamente programa-se uma Classe de Teste para cada Classe do sistema
 - As classes de teste podem ser alocadas dentro da package onde está a classe a ser testada
 - Bom para compartilhar (para testar) métodos (use protected)
- No seu eclipse, selecione a package configuracoes
 - New -> Junit Test Case
 - Selecione “New Junit Jupiter Test”
 - Nome: TestConfiguracao
 - Classe UnderTest: configuracoes.Configuracao (pode buscar na lista)
 - Finish

O primeiro teste de unidade





Residência
em Software

Ainda tem erro?

- Deixe o Eclipse de ajudar
- Botão direito no ícone do erro (x vermelho)
 - Adds 'requires junit' to module-info.java file
- PRONTO!
 - Podemos programar nossa primeira classe de teste

```
1 package configuracoes;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5 import org.junit.jupiter.api.Test;
6
7 class TestConfig {
8
9     @Test
10    void test() {
11        fail();
12    }
13
14 }
```

The screenshot shows the Eclipse IDE interface with the Java code for `TestState.java`. A context menu is open at the bottom of the code editor, listing two options: "Add 'requires org.junit.jupiter.api' to module-info.java" and "Fix project setup...".

TestConfiguracao

- Estudaremos JUNIT em detalhes mais à frente
- Aqui: fazer o primeiro teste funcionar
- Observe sua classe TestConfiguracao

```
@Test  
void test() {  
    fail("Not yet implemented");  
}
```

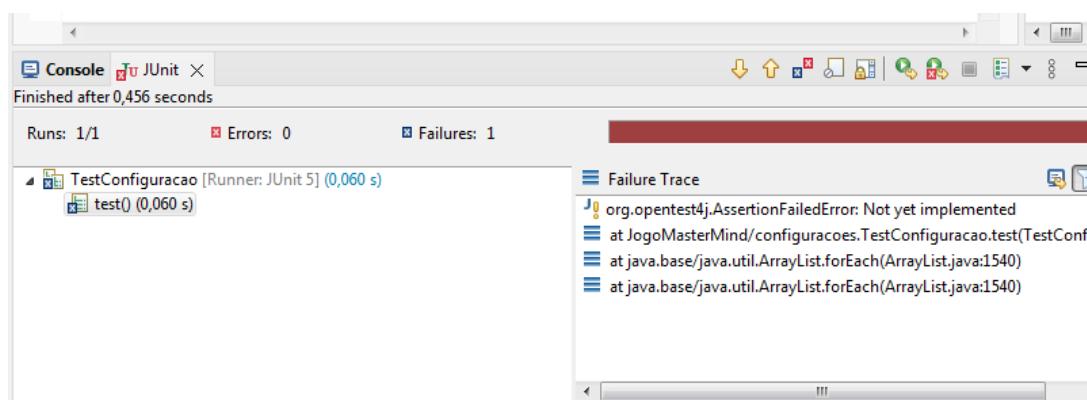
- Annotation: indicação no programa para algum aspecto específico que o compilador deve considerar
 - Começa com @
 - @Test indica que o método é um Teste de Unidade (definido na biblioteca Junit)
 - fail(): comando para *forçar* o teste a falhar.



Residência
em Software

Execute o Teste!

- Run As -> Junit Test (note que a biblioteca JUNIT “embute” o método main
 - As Anotations guiam a execução para lá
- Observe a saída na Aba JUNIT
 - Barrinha Vermelha: o teste falhou (era o que queríamos!)
 - Failure Trace: quais métodos (e linhas de código) estão envolvidos no momento da falha
 - Not Yet Implemented
 - TestConfiguration.Test()
 - Linha...



Programando o Teste

- Primeiro: ver se cadastra corretamente um alfabeto válido (`setAlfabeto()`)
 - Vamos criar um método `testSetAlfabeto()`
 - Não esquecer a annotation `@Test`
- Por exemplo, o alfabeto formado pelos caracteres ABCDEFGH
- Etapas do teste
 - Instanciar um objeto do tipo Configuration
 - Executar o método SetAlfabeto para inserir um alfabeto
 - Verificar se o resultado (`getAlfabeto()`) é igual ao que foi enviado
 - `assertEquals(...)` -> verifica se dois valores são iguais



Residência
em Software

Como ficou?

The screenshot shows a Java code editor with a tab bar at the top containing three tabs: 'Configuracao.java', 'ListaConfiguracoes.java', and '*TestConfiguracao.java'. The code in the editor is:

```
1 package configuracoes;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5 import org.junit.jupiter.api.Test;
6
7 class TestConfiguracao {
8
9     @Test
10    void testSetAlfabeto() {
11        Configuracao configuracao = new Configuracao();
12        String senha = "ABCDEFGHI";
13        configuracao.setAlfabeto(senha);
14        assertEquals(senha, configuracao.getAlfabeto());
15    }
16
17 }
18
```



Residência
em Software

Como ficou?

```
1 package configuracoes;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5 import org.junit.jupiter.api.Test;
6
7 class TestConfiguracao {
8
9     @Test
10    void testSetAlfabeto() {
11        Configuracao configuracao = new Configuracao();
12        String senha = "ABCDEFGH";
13        configuracao.setAlfabeto(senha);
14        assertEquals(senha, configuracao.getAlfabeto());
15    }
16
17 }
18
```

Instancia um
objeto do tipo
Configuracao

Executa
setAlfabeto()

Verifica se a
resposta é
adequada

O teste falhou?

- A depender de como você produziu este setAlfabeto()
 - Não tem problema
- Programar e repetir o teste até que funcione
 - Barrinha verde
- Em seguida passamos para a segunda parte do teste: não admitir um alfabeto nulo (ou de apenas um caracter)
 - Retorna uma runtime exception com a mensagem: “O alfabeto deve possuir mais de 1 caracter”



O teste falhou?

- Segunda parte do teste: não admitir um alfabeto vazio (ou de apenas um caracter)
- Possíveis aspectos
 - Parâmetro passado é nulo
 - Parâmetro é uma String com zero elementos
 - Parâmetro é uma string com 1 elemento

Não admitir um alfabeto vazio (ou de apenas um caracter)

- Testamos se uma String válida é enviada
- Testamos se uma string null é enviada
- Falta:
 - Testar uma string com um único caracter



Não admitir um alfabeto vazio (ou de apenas um caracter)

- Testamos se uma String válida é enviada
- Testamos se uma string null é enviada
- Falta:
 - Testar uma string com um único caracter

```
1 Configuracao.java  2 ListaConfiguracoes.java  3 TestConfiguracao.java X
 3 import static org.junit.jupiter.api.Assertions.*;
 4 |
 5 import org.junit.jupiter.api.Assertions;
 6 import org.junit.jupiter.api.Test;
 7
 8 class TestConfiguracao {
 9
10     @Test
11     void testSetAlfabeto() {
12         Configuracao configuracao = new Configuracao();
13
14         //Caso geral: salvar uma senha válida
15         String senha = "ABCDEFGHI";
16         try {
17             configuracao.setAlfabeto(senha);
18         } catch (Exception e1) {
19             // TODO Auto-generated catch block
20             fail("Gerou exceção indevida");
21             e1.printStackTrace();
22         }
23         assertEquals(senha, configuracao.getAlfabeto());
24
25         //*****Caso 1: tentar inserir uma senha null
26         senha = null;
27         try {
28             configuracao.setAlfabeto(senha);
29         } catch (Exception e) {
30             // temos que ter certeza que a mensagem está correta
31             assertEquals("O alfabeto deve possuir mais de 1 caracter", e.getMessage());
32         }
33         assertFalse(configuracao.getAlfabeto()==null);
34
35         //*****Caso 3: Tentar inserir uma senha com um único caractere
```



Residência
em Software

Programa aí!



Testar se não há repetição de Caracteres

- Difícil prever todos os casos possíveis
 - Uma boa experiência como programador ajuda muito
- Testar 2 caracteres iguais no começo da String
- Testar 2 caracteres iguais no final da String
- Testar 1 caracter no começo e outro no fim
- Testar 1 caracter em posição intermediária e outro no fim
- Testar 1 caracter no início e outro em posição intermediária
- Testar 2 caracteres em posições intermediárias adjacentes
- Testar 3 caracteres em posições intermediárias não adjacentes
- Em qualquer dos casos: Exception
 - Mensagem: “Não podem haver caracteres repetidos no alfabeto”



Residência
em Software

Programa aí!



Residência
em Software

Sugestão de Código

- Configuracao.setAlfabeto
 - Observe o método ChecaRepetido()
 - Protected

```
1 package configuracoes;
2
3 public class Configuracao {
4
5     private String alfabeto;
6
7     public String getAlfabeto() {
8         return alfabeto;
9     }
10
11    public void setAlfabeto(String alfabeto) throws Exception {
12        if ((alfabeto==null) || (alfabeto.length() < 2) ) {
13            Exception e = new Exception("O alfabeto deve possuir mais de 1 caracter");
14            throw e;
15        }
16        boolean r = checaRepetido(alfabeto);
17        if (r) {
18            Exception e = new Exception("Não podem haver caracteres repetidos no alfabeto");
19            throw e;
20        }
21        this.alfabeto = alfabeto;
22    }
23
24    protected boolean checaRepetido(String alfabeto) {
25        for (int i=0;i< alfabeto.length(); i++) {
26            char c1 = alfabeto.charAt(i);
27            for (int j=i+1;j<alfabeto.length(); j++) {
28                char c2 = alfabeto.charAt(j);
29                if (c1==c2)
30                    return true;
31            }
32        }
33        return false;
34    }
35
36}
37
```

O novo status do projeto

- A classe Configuração possui o método setAlfabeto() implementado
- Temos MUITA confiança de que o este método está implementado sem erros.
- Temos a possibilidade de verificar isso a qualquer momento.
- Conseqüências no desenvolvimento
 - Podemos ter muita confiança ao usar o método setAlfabeto()
 - Um andar está pronto. Podemos fazer outro acima dele
 - Podemos verificar se alterações futuras geraram erro no método setAlfabeto()
 - Permite evoluir o Software com mais segurança!



Residência
em Software

Próximos Passos?

Classe Configuração

- Para se definir uma configuração
 - Definir o alfabeto
 - O que pode dar errado?
 - Definir o tamanho da senha
 - O que pode dar errado?
 - Definir o máximo de tentativas
 - O que pode dar errado?
 - Toda configuração cadastrada deve ser salva num arquivo JSON
 - O que pode dar errado?

Configuração
Nome
Alfabeto
TamanhoSenha
MaxTentativas
getNome e setNome
definirAlfabeto()
setTamanhoSenha()
setMaxTentativas()



Residência
em Software

Classe Jogo

- Um jogo possui uma certa configuração

Configuração
Nome
Alfabeto
TamanhoSenha
MaxTentativas
getNome e setNome
definirAlfabeto()
setTamanhoSenha()
setMaxTentativas()

Jogo
Configuracao
Senha
Tentativas
Feedbacks
Resultado
Jogo(configuracao)
setSenha()
setTentativas()
iniciaJogo()
novaTentativa()
terminaJogo()