

Mas a vereda dos justos é como a luz da aurora,
que vai brilhando mais e mais até ser dia
perfeito. Provérbios 4:18

Engenharia de Computação

LINGUAGEM DE PROGRAMAÇÃO ORIENTADA A OBJETOS

- ▶ Prof: José Antonio Gonçalves
- ▶ zag655@gmail.com

Nestes Slides:

- **Orientação a Objetos em Java:**
- Definições e aplicação do conceito de Encapsulamento;

Encapsulamento

Encapsulamento *(definição)*

Encapsulamento (definição): Possibilidade de se ocultar detalhes da implementação de uma classe.

Comportamento: será utilizado somente aquilo que o programador da classe permitir.

Manutenção: mantendo os mesmos serviços da classe, podemos alterar sua estrutura interna da classe (refinamentos de código, por exemplo) sem que outras classes (códigos) que dependam dela tenham que ser alteradas.

Segurança: Não se consegue corromper o estado de um objeto por distração, pois partes deste objeto está protegido pelo encapsulamento

Encapsulamento (*níveis*)

Pode ocorrer em 3 níveis de especificação de acesso:

PÚBLICO (public): todos têm acesso. Um atributo pode ter seu valor alterado a partir de qualquer outro código, mesmo sendo este de uma classe qualquer.

PROTEGIDO (protected): em Java tem acesso quem está no mesmo pacote ou classes que herdem a classe que contenha atributo ou método protegido

PRIVADO (private): Restrição total fora da classe. Só têm acesso membros da própria classe

Encapsulamento (*convenção*)

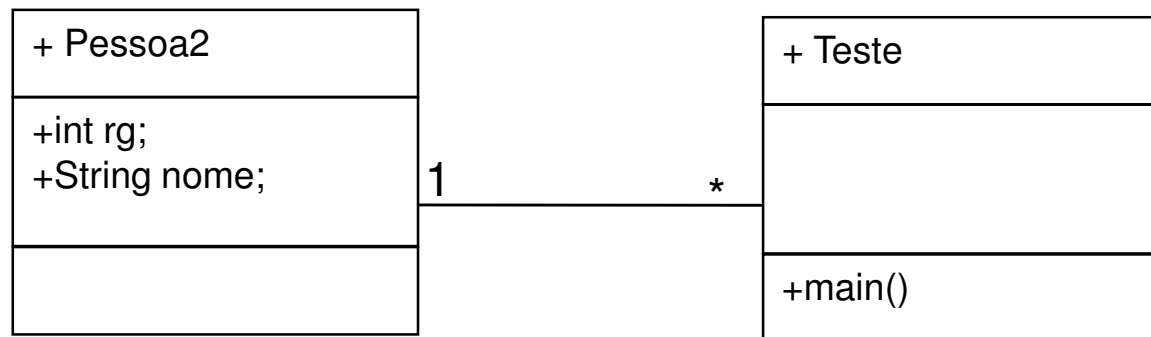
Enquanto programadores de uma classe podemos gerenciar o acesso a seus membros de acordo com a necessidade, porém existe Convenção que determina esta gerência. O mais usual é:

Atributos privados

Métodos públicos

Testando o encapsulamento

Observe o Diagrama de Classes a seguir. Depois observe o código.



Testando Encapsulamento

Classe Pessoa2

```
public class Pessoa2 {  
    public int rg;  
    public String nome;  
  
    public void mostraDados(){  
        System.out.println("\n RG: "+rg);  
        System.out.println("\n Nome: "+nome);  
    }  
}
```

Classe Teste

```
public class Teste {  
    public static void main(String arg[]){  
        Pessoa2 p3 = new Pessoa2();  
        p3.rg=50; //acessa o atributo RG diretamente (ERRADO)  
        p3.nome = "amor"; //acessa o atributo NOME diretamente (ERRADO)  
        p3.mostraDados();  
    }  
}
```

Testando Encapsulamento

Classe Pessoa2

```
public class Pessoa2 {  
    private int rg;  
    private String nome;  
  
    public void mostraDados(){  
        System.out.println("\n RG: "+rg);  
        System.out.println("\n Nome: "+nome);  
    }  
}
```

Altere a especificação
de acesso para
“private”, e teste o
programa

Classe Teste

```
public class Teste {  
    public static void main(String arg[]){  
        Pessoa2 p3 = new Pessoa2();  
        p3.rg=50; //acessa o atributo RG diretamente (ERRADO)  
        p3.nome = "amor"; //acessa o atributo NOME diretamente (ERRADO)  
        p3.mostraDados();  
    }  
}
```

Encapsulamento

Relembrando:

Enquanto programadores de uma classe podemos gerenciar o acesso a seus membros de acordo com a necessidade, porém existe Convenção que determina esta gerência. O mais usual é:

Atributos privados

Métodos públicos

Encapsulamento e os Métodos getters e setters *(definição)*

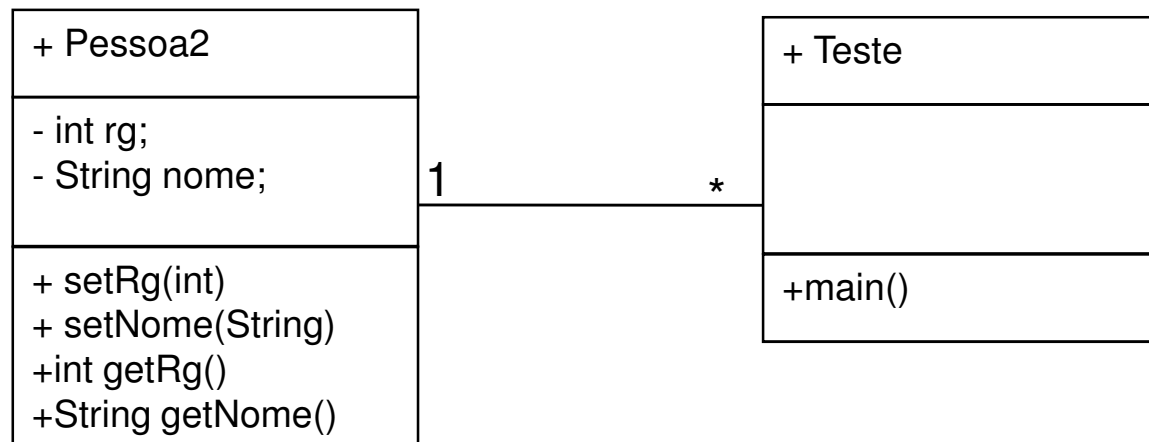
Métodos convencionais

set: possibilita alterar o estado de um objeto permitindo a alteração de valores de seus atributos através da passagens de parâmetros;

get: recupera valor de um atributo através do “retorno” do mesmo.

Encapsulamento e os Métodos getters e setters

**Observe a alteração de Diagrama de Classes a seguir.
Depois observe o código, também alterado.**



Nota: lembre-se...

- privado (private)
- + publico (public)

Encapsulamento e os Métodos getters e setters (aplicação)

Classe Pessoa2

```
public class Pessoa2 {  
    private int rg;  
    private String nome;  
  
    public void setRg(int rg){  
        this.rg=rg;  
    }  
    public void setNome(String nome){  
        this.nome=nome;  
    }  
    public int getRg(){  
        return rg;  
    }  
    public String getNome(){  
        return nome;  
    }  
  
}
```

Observe a “troca de mensagens” nas linhas 4,5,6 e 7 da classe Teste.

Classe Teste

```
1. public class Teste {  
2.     public static void main(String arg[]){  
3.         Pessoa2 p3 = new Pessoa2();  
4.         p3.setRg(50);  
5.         p3.setNome("amor");  
6.         System.out.println("\n RG: "+p3.getRg() );  
7.         System.out.println("\n NOME: "+p3.getNome() );  
8.     }  
9.  
10. }
```

Troca de mensagem: quando um Objeto se comunica com outro através de métodos

Encapsulamento (*contextualizando*)

Em uma breve revisão e definição de uma **premissa**:

- A compilação de um código fonte Java, gera um arquivo com a extensão “.class” (ou byteCode) que será interpretado pela JVM (máquina virtual java);
- Para alterar a funcionalidade de um código precisamos do código fonte. Assim, altera-se o código fonte, compila-se novamente este **gerando um novo byteCode** (com as características alteradas);
- O que repassamos aos “clientes” é o byteCode, no qual é impossível alterar suas funcionalidades. **Só podemos usá-lo e da maneira que foi implementado.**
- **Logo:** graças as especificações de acesso (public, **protected** e **private**) torna-se impossível acessar um objeto de uma forma não definida pelo programador da classe da qual foi instanciado o objeto.
- Embora o encapsulamento seja definição durante a codificação da classe seu uso efetivo se dá em nível de objeto.