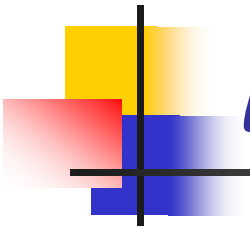
UNIVERSIDADE CATÓLICA DE PERNAMBUCO

Encapsulamento de Dados

Universidade Católica de Pernambuco
Ciência da Computação

Prof. Márcio Bueno
poonoite@marciobueno.com

Fonte: Material da Profª Karina Oliveira

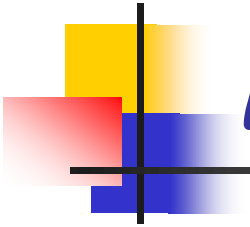


Modificadores de Visibilidade

- Especificam quais classes têm acesso aos membros (classe, atributos, métodos e construtores) de uma determinada classe
  - public
  - private
  - protected
  - "friendly"

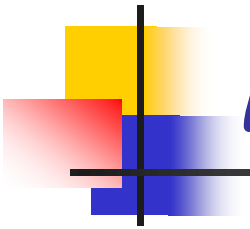
Modificadores de Visibilidade

- **public**
  - Classe pode ser instanciada por qualquer outra classe
  - Atributos e métodos são acessíveis (leitura, escrita) por objetos de qualquer classe.
- **private**
  - Atributos só podem ser acessados por objetos da mesma classe
  - Métodos só podem ser chamados por métodos da própria classe.



Modificadores de Visibilidade

- **protected**
  - Atributos e métodos são acessíveis dentro da própria classe, das subclasses e das classes que façam parte do mesmo pacote.
- **Nada especificado**
  - Classe é visível somente por classes do mesmo pacote
  - Atributos e métodos são acessíveis somente dentro das classes que pertencem ao mesmo pacote.
  - Este modo de acesso é também chamado de **default (friendly)** (amigável).



Modificadores de Visibilidade

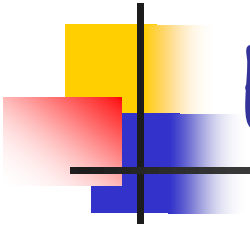
- Exemplo

```
public class Teste {
      public int atrib1;
      private int atrib2;
      protected int atrib3;
      int atrib4;

      public Teste() {
            atrib1 = atrib2 = atrib3 = atrib4 = 0;
      }
}
```

Modificadores de Visibilidade

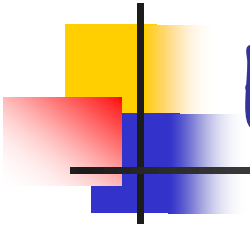
- Exemplo (Cont.):

```
public class Aplicacao {
      public static void main(String[] args) {
            Teste t = new Teste();
            t.atrib1 = 5;  // público
            t.atrib2 = 5; // privado
            t.atrib3 = 5; // protegido
            t.atrib4 = 5; // nada especificado
      }
}
```



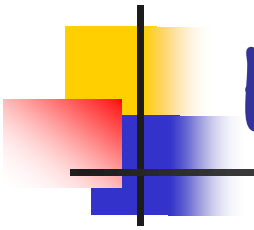
Encapsulamento de Dados

- **Encapsulamento** é o que se faz quando se restringe o acesso aos dados (atributos) de uma classe (*information hiding*).
- A idéia é fazer da classe uma **cápsula**, onde seus atributos só poderão ser acessados por determinados métodos.
- Técnica de encapsulamento – Pode-se alcançar o encapsulamento de dados configurando as classes da seguinte forma:
  - **Atributos PRIVATE (Dados encapsulados)**
  - **Métodos PUBLIC**



Encapsulamento de Dados

- Principais benefícios:
  - **Proteção dos atributos da classe** de acessos indevidos ou acidentais.
  - **Possibilidade de definir regras para alteração** dos valores mantidos pelos atributos.



Exemplo

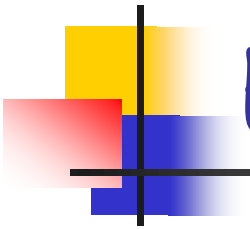
```
public class Circulo {
private double raio;
public Circulo(double r) { raio = r; }
        public Circulo( ) { raio = 2; }
        public double comprimento( ) {
        return ( 2 * 3.14 * raio);
}
        public double area( ) {
        return (3.14 * raio * raio);
}
}
```

Exemplo

- Acessando um objeto com atributos encapsulados.

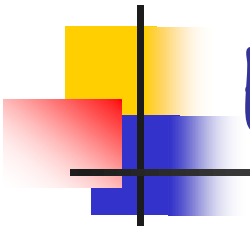
```
public class ExemploCirculo {
    public static void main(String args[ ]) {
            Circulo c1 = new Circulo(5);
            System.out.println("A área de c1 é: " + c1.area( ));
            System.out.println("O comp. de c1 é: " + c1.comprimento( ));
            System.out.println("O raio de c1 é: " + c1.raio);
            c1.raio = 10;
            System.out.println("Novo raio de c1 é: " + c1.raio);
    }

}
```

Então, como fazer para acessar os atributos?



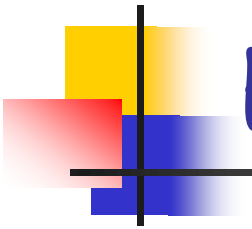
Encapsulamento de Dados

- Como acessar os atributos?
  - **Métodos get e set**
    - São métodos definidos para cada atributo da classe.
    - Método **get** utilizado para recuperar o valor mantido por um atributo.
    - Método **set** utilizado para alterar o valor mantido por um atributo.



Encapsulamento de Dados

- Como acessar os atributos?
  - **Métodos get e set**
    - Sintaxe (Convenção):
      - `public <tipo_atributo> getNomeAtributo( )`
      - `public void setNomeAtributo(<tipo_atributo> novoValor)`
    - Sintaxe para atributos booleanos:
      - `public boolean isNomeAtributo( )`
      - `public void setNomeAtributo(boolean novoValor)`



Exemplo

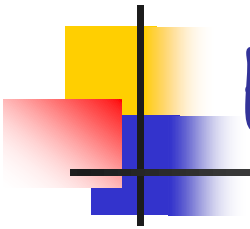
- Métodos get/set para todos os atributos

```
public class Circulo {
        private double raio;
        public void setRaio(double r) { raio = r; }

        public double getRaio( ) { return raio; }
        public Circulo(double r) { raio = r; }
        public Circulo( ) { raio = 2; }
public double comprimento( ) { return ( 2 * 3.1415 * raio); }
public double area( ) { return (3.1415 * raio * raio); }

}
```

Exemplo

- Erros de compilação: Acessando um objeto com atributos encapsulados

```
public class ExemploCirculo {
    public static void main(String args[ ]) {
            Circulo c1 = new Circulo(5);
            System.out.println("A área de c1 é: " + c1.area( ));
            System.out.println("O comp. de c1 é: " + c1.comprimento( ));
            System.out.println("O raio de c1 é: " + c1.raio);
            c1.raio = 10;
            System.out.println("Novo raio de c1 é: " + c1.raio);
    }

}
```



Exemplo

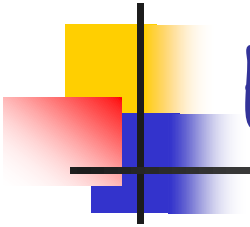
- Exemplo Corrigido: Acessando um objeto com atributos encapsulados.

```
public class ExemploCirculo {
    public static void main(String args[ ]) {
            Circulo c1 = new Circulo(5);
            System.out.println("A área de c1 é: " + c1.area( ));
            System.out.println("O comp. de c1 é: " + c1.comprimento( ));
            System.out.println("Raio de c1: " + c1.getRaio( ));
            c1.setRaio(10);
            System.out.println("Novo raio c1:" + c1.getRaio( ));
    }

}
```

Encapsulamento de Dados

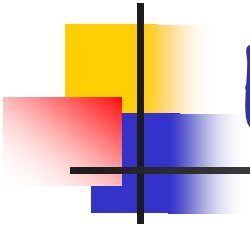
- Regras para alteração do valor do atributo podem ser adicionadas ao método set.

```
public class Circulo {
        private double raio;
        public void setRaio(double r) {
                if (r > 0)  raio = r;
                else System.out.println("Informar raio > 0!");
        }
        public double getRaio( ) { return raio; }
        public double comprimento( ) { return ( 2 * 3.1415 * raio); }
        public double area( ) { return (3.1415 * raio * raio); }
        public Circulo(double r) { this.setRaio(r); }
        public Circulo( ) { this.setRaio(2); }
}
```



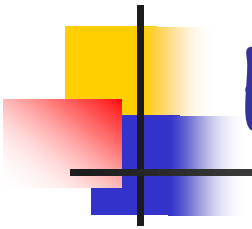
Exercício

- Escreva a classe Departamento com as seguintes definições:
  - Dois atributos:
    - codigo (inteiro) → não pode receber valores menores que zero.
    - nome (string) → não pode receber valores nulos ou string vazia.
  - Métodos de acesso aos atributos (get / set);
  - Um construtor que receba valores para todos os atributos da classe.
  - Um método public String toString( ) que retorna todos os dados do departamento em um formato string.
  - OBS: Usar a técnica de encapsulamento sugerida no curso



Exercício

- Escreva a classe Funcionário com as seguintes definições:
  - Três atributos:
    - matricula (inteiro) → não pode receber valores menores que zero.
    - nome (string) → não pode receber valores nulos ou string vazia.
    - depto (utilize a classe Departamento implementada anteriormente) → não pode receber valores nulos.
  - Métodos de acesso aos atributos (get / set);
  - Um construtor que receba valores para todos os atributos da classe.
  - Um método string toString( ) que retorna todos os dados do funcionário em um formato string.
  - OBS: Usar a técnica de encapsulamento sugerida no curso.



Exercícios

- Implementar uma aplicação que cria um objeto do tipo Funcionario, a partir de dados fornecidos pelo usuário e, ao final, imprime os valores dos atributos do objeto Funcionario criado.