

Linguagem de Programação

Considerações sobre encapsulamento

Estrutura do programa Java

Declaração
Da Classe

Corpo da Classe

Atributos

Construtores

Métodos

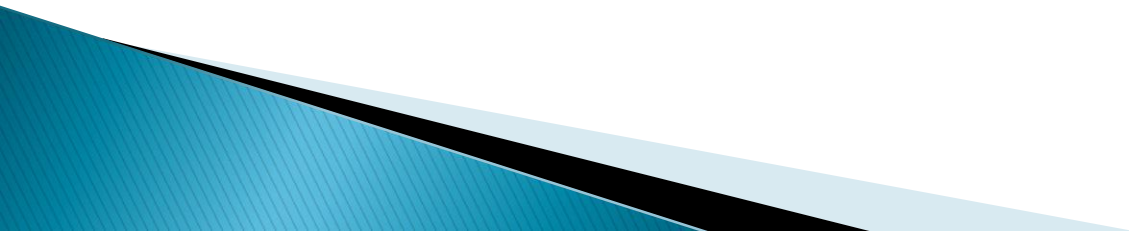
```
/* arquivo Produtos.java */
public class Produtos {

    public int codProd;
    public String descricaoProd;
    public double precoProd;

    public Produtos() {
    }
    public Produtos(int cod, String desc, double pr) {
    }
    public Produtos(Produtos p) {
    }

    public void imprimirProduto() {
        System.out.println("");
        System.out.println("Codigo do produto:" + this.codProd);
        System.out.println("Descricao do produto:" + this.descricaoProd);
        System.out.println("Preco do produto:" + this.precoProd);
    }
}
```

Encapsulamento



Encapsulamento

- ▶ Considerado um dos pilares da Programação Orientada a Objetos
- ▶ É o mecanismo usado para “esconder” uma ideia, de forma que não aconteça a exposição de detalhes internos para o usuário. Isso faz com que partes do sistema sejam o mais independentes possível

Encapsulamento

- ▶ Para aplicar o mecanismo de encapsulamento basta definir os atributos das classes com o modificador *private*. Obs: o símbolo (–) é usado em diagramas de classes da UML
- ▶ Quando alteramos o especificador para *private*, o acesso direto aos atributos não será mais possível
- ▶ Para acessar os atributos será necessário criar métodos *setters* e *getters*.

Encapsulamento

- ▶ Assim, o encapsulamento significa esconder todos os membros de uma classe
- ▶ `private` é um modificador de acesso (também chamado de modificador de visibilidade)
- ▶ Quando marcamos um atributo como privado, não permitimos o acesso ao mesmo de todas as outras classes
- ▶ Em programação orientada a objetos, é prática quase que **obrigatória** proteger seus atributos com *private*

Encapsulamento

- ▶ A palavra chave *private* também pode ser usada para modificar o acesso a um método
- ▶ Quando fazemos isso dizemos que a funcionalidade é apenas para auxiliar a própria classe. O uso de tal funcionalidade não ocorre para outras “pessoas”
- ▶ Objetivo: expor o mínimo possível de funcionalidades, para criar um baixo acoplamento entre as nossas classes

Exemplo da aula anterior

O exemplo abaixo usa a classe Produtos em uma outra classe definida como TestaProduto, que contém a função main e que poderá ser executada. Percebe-se que p1 é uma instância da classe Produtos e ocorre o acesso direto aos atributos pois foram definidos como public na classe Produtos.

```
/* Classe principal arquivo TestaProduto.java */
public class TestaProduto
{
    public static void main (String[] args)
    {
        Produtos p1 = new Produtos();          /* instancia o objeto p1*/
        p1.codProd=123;
        p1.descricaoProd="Protetor Solar";
        p1.precoProd=79.9;
        p1.imprimirProduto();
    }
}
```


Observação

Produtos **p1** = new Produtos();

No exemplo da classe Produtos nós inicializamos os atributos do objeto p1 em main. Porém, isso não é bom, pois quebramos o encapsulamento.

```
p1.codProd=123;  
p1.descricaoProd="Protetor Solar";  
p1.precoProd=79.9;
```

A classe TestaProduto está acessando diretamente o atributos da classe Produtos por meio do objeto p1.

Podemos contornar esse problema, definindo os nossos próprios construtores e tornando os atributos da classe Produtos com o qualificador **private**.

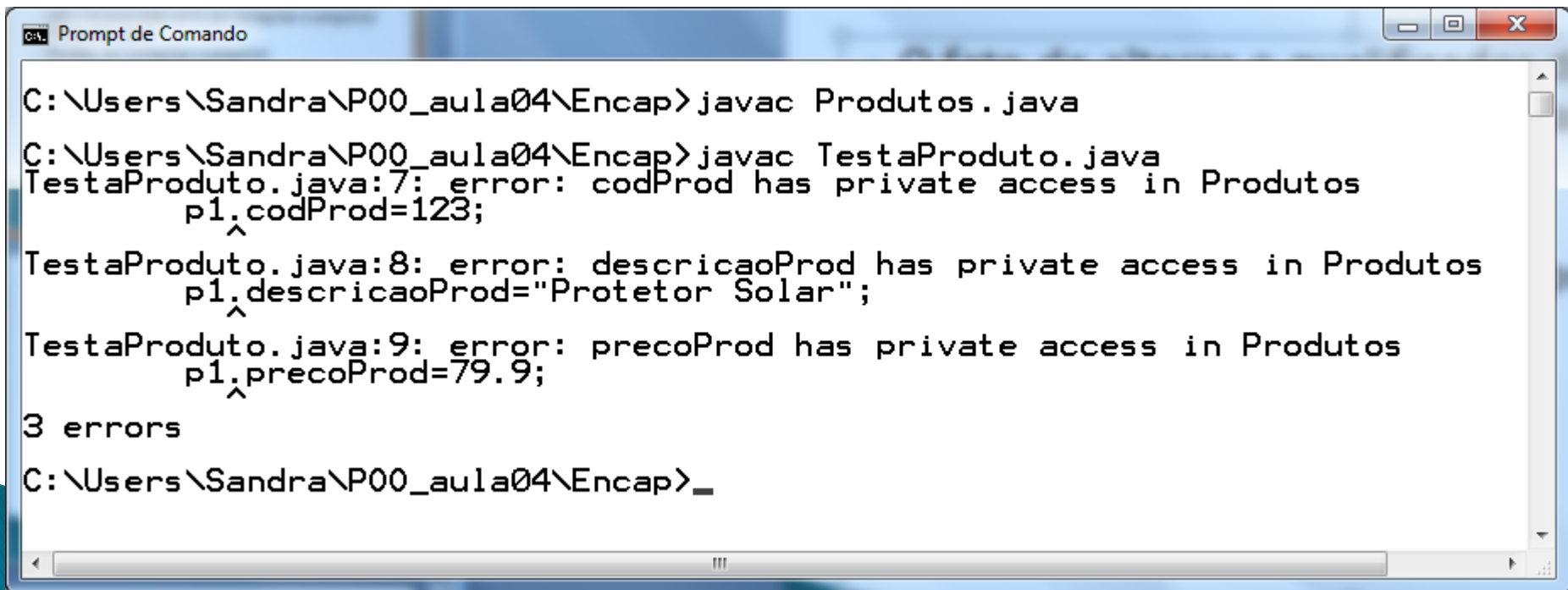
Esse qualificador diz que os atributos de codProd, descricaoProd e precoProd somente podem ser acessados por métodos da classe Produtos.

Mudando de public para private

```
TestaProduto.java x Produtos.java x
1  /* arquivo Produtos.java */
2  public class Produtos    {
3      private int codProd;
4      private String descricaoProd;
5      private double precoProd;
6
7      public Produtos () {
13     public Produtos(int cod, String desc, double pr){
19     public Produtos(Produtos p){
25
26     public void imprimirProduto()    {
32 }
```

Mudando de public para private

- ▶ O fato de alterar o qualificador dos atributos não irá acarretar erro ao compilar o arquivo Produtos.java
- ▶ Porém, ao compilar o arquivo TestaProduto.java teremos os seguintes erros:



```

C:\Users\Sandra\P00_aula04\Encap>javac Produtos.java
C:\Users\Sandra\P00_aula04\Encap>javac TestaProduto.java
TestaProduto.java:7: error: codProd has private access in Produtos
    p1.codProd=123;
    ^
TestaProduto.java:8: error: descricaoProd has private access in Produtos
    p1.descricaoProd="Protetor Solar";
    ^
TestaProduto.java:9: error: precoProd has private access in Produtos
    p1.precoProd=79.9;
    ^
3 errors
C:\Users\Sandra\P00_aula04\Encap>_

```

Mudando de public para private

- ▶ Por isso a necessidade de criar métodos para permitir o acesso “indireto” ao atributos que foram especificados como private
- ▶ Nesse caso é necessário criar os métodos setters (mutator) e getters (accessor) para cada atributos que foi definido como private

Métodos e seus tipos

Tipos de Métodos

- Os métodos são a parte funcional da POO, e podem executar operações, funções, etc.
- Tipos de métodos:
 - Método Accessor (ou Acessor) : lê o valor de um atributo (Iniciado por **get**);
 - Método Mutator (ou Modificador): modifica o valor de um atributo (Iniciado por **set**);
 - Método Worker: estão vinculados a métodos mais “robustos” que em geral executam tarefas mais específicas. Dica: usar verbo no infinitivo. Ex: imprimir, alterar;

Método get

- ▶ Para escrever um método get deve-se considerar que:
 - ▶ Todo método get precisa ter retorno
 - ▶ Todo método get não recebe parâmetro
 - ▶ Sintaxe:

```
public tipoRetornado getNomeAtributo() {  
    return atributo;  
}
```

- Exemplo:

```
public int getCodProd() {  
    //codProd é de instância, do tipo int  
    return codProd;  
}
```

- ▶ Para chamar o método get use:
 - ▶ nomeDoObjeto.getNomeAtributo();

Método set

- ▶ Para escrever um método set deve-se considerar que:
 - ▶ Todo método set não tem retorno
 - ▶ Todo método set recebe um parâmetro (do mesmo tipo de dado do atributo ao qual ele está especificado)

- ▶ Sintaxe:

```
public void setNomeAtributo(tipoDeDado nomeVariávelLocal)
{
    atributoEspecificado = nomeVariávelLocal ;
}
```

- Exemplo:

```
public void setCodProd(int c) {
    //c é uma variável local do tipo int; codProd é de instância
    codProd = c;
}
```

- ▶ Para chamar o método set use:
 - ▶ nomeDoObjeto.setNomeAtributo(conteúdoDeAcordoComTipo);

Tipos de Métodos – Exemplo

Pessoa

- String: nome
- String: cpf

- + getName: String
- + getCPF: String
- + setName: void
- + setCPF: void
- + validaCPF: boolean

```
public class Pessoa {  
    private String nome;  
    private String cpf;
```

Accessor

```
public String getName() {  
    //nome é de instância, do tipo String  
    return nome;}  
  
public String getCPF() {  
    //cpf é de instância, do tipo String  
    return cpf;}
```

Mutator

```
public void setName(String n) {  
    //n é uma variável local do tipo String; nome é de instância  
    nome = n;}  
  
public void setCPF(String c) {  
    //c é uma variável local do tipo String; cpf é de instância  
    cpf = c;}
```

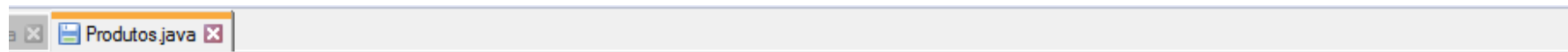
```
public boolean validaCPF() {  
    //variavel local  
    boolean validado = false;  
    //algoritmo que verifica se o CPF não é nulo  
    if (cpf != null){  
        validado = true;  
    } else {  
        validado = false;  
    }  
    return validado;  
}
```

Worker

Tipos de Métodos – Exemplo

```
public class TestaPessoa {  
  
    public static void main(String args[]) {  
  
        Pessoa pessoa1 = new Pessoa();  
        pessoa1.setNome("Sandra");  
        pessoa1.setCPF("124");  
        System.out.println("Nome: "+pessoa1.getNome());  
        System.out.println("CPF: "+pessoa1.getCPF());  
        System.out.println("CPF valido: "+pessoa1.validaCPF());  
        System.out.println("");  
  
        Pessoa pessoa2 = new Pessoa();  
        pessoa2.setNome("Bianca");  
        pessoa2.setCPF("567");  
        System.out.println("Nome: "+pessoa2.getNome());  
        System.out.println("CPF: "+pessoa2.getCPF());  
        System.out.println("CPF valido: "+pessoa2.validaCPF());  
        System.out.println("");  
  
    }  
}
```

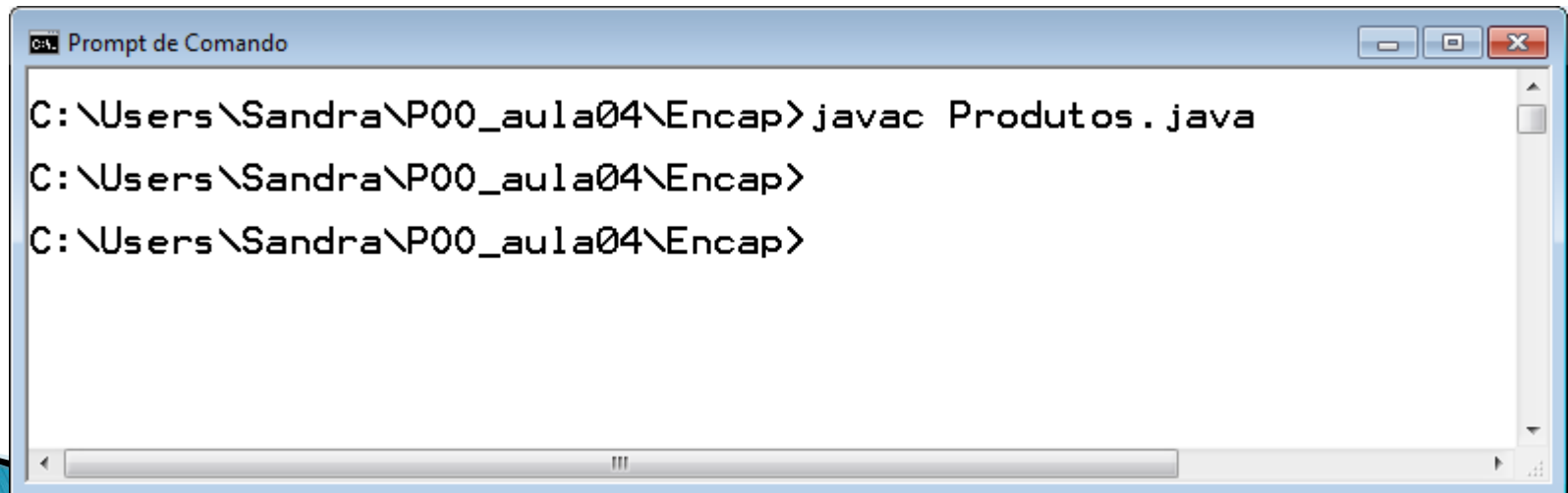
Getters e Setters para a classe Produto.java



```
public int getCodProd() {  
    //codProd é de instância, do tipo int  
    return codProd;  
}  
public String getDescricaoProd() {  
    //descricaoProd é de instância, do tipo String  
    return descricaoProd;  
}  
public double getPrecoProd() {  
    //precoProd é de instância, do tipo double  
    return precoProd;  
}  
public void setCodProd(int c) {  
    //c é uma variável local do tipo int; codProd é de instância  
    codProd = c;  
}  
public void setDescricaoProd(String d) {  
    //d é uma variável local do tipo String; descricaoProd é de instância  
    descricaoProd = d;  
}  
public void setPrecoProd(double p) {  
    //p é uma variável local do tipo double; precoProd é de instância  
    precoProd = p;  
}
```

Gets e Setters para a classe Produto.java

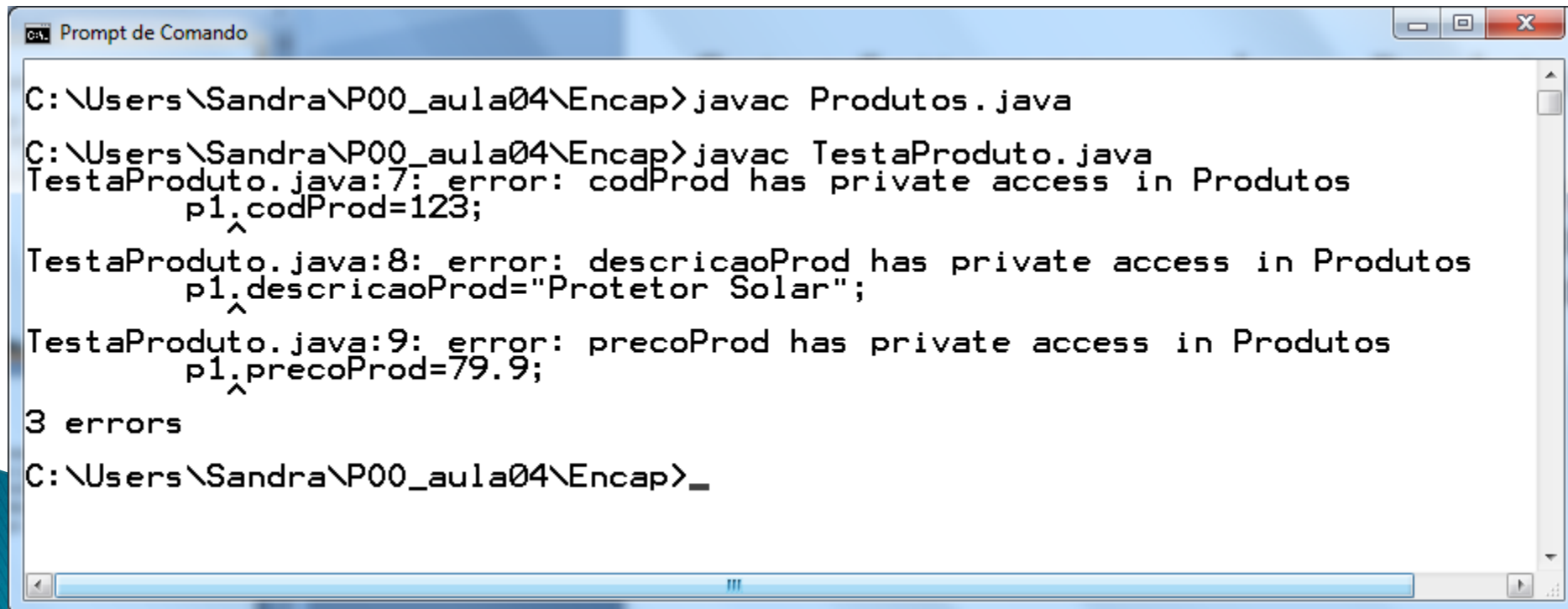
- ▶ O fato de inserir métodos getters e setters na classe Produtos.java não irá gerar erros no processo de compilação (apenas se existirem erros de sintaxe, erros de lógica ou tipos de dados)



```
Ca\ Prompt de Comando
C:\Users\Sandra\P00_aula04\Encap>javac Produtos.java
C:\Users\Sandra\P00_aula04\Encap>
C:\Users\Sandra\P00_aula04\Encap>
```

Gets e Setters para a classe Produto.java

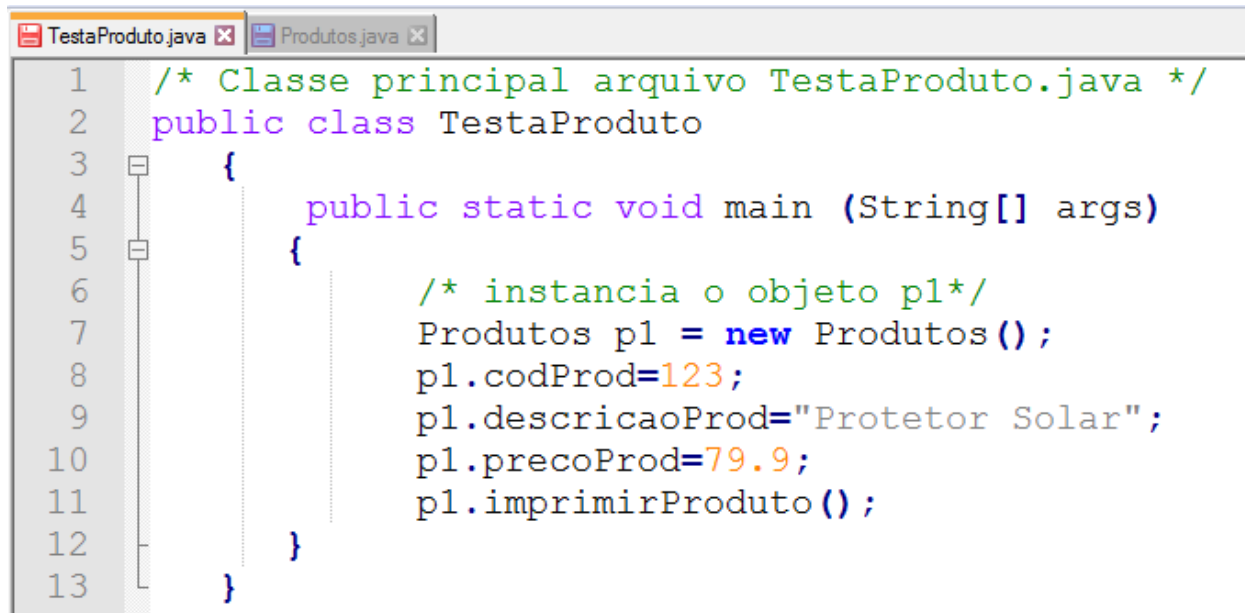
- ▶ Porém ao compilar a classe TestaProduto.java aparecerão erros (“*has private access in*”) conforme observado a seguir, pois o acesso direto ao atributo não é mais possível:



```
CA: Prompt de Comando
C:\Users\Sandra\P00_aula04\Encap>javac Produtos.java
C:\Users\Sandra\P00_aula04\Encap>javac TestaProduto.java
TestaProduto.java:7: error: codProd has private access in Produtos
    p1.codProd=123;
    ^
TestaProduto.java:8: error: descricaoProd has private access in Produtos
    p1.descricaoProd="Protetor Solar";
    ^
TestaProduto.java:9: error: precoProd has private access in Produtos
    p1.precoProd=79.9;
    ^
3 errors
C:\Users\Sandra\P00_aula04\Encap>_
```

Por que?

- ▶ A classe TestaProduto.java esta fazendo o acesso direto aos atributos

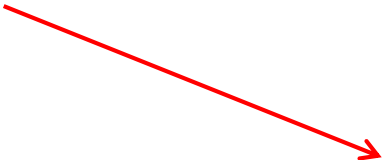


```
1  /* Classe principal arquivo TestaProduto.java */
2  public class TestaProduto
3  {
4      public static void main (String[] args)
5      {
6          /* instancia o objeto p1*/
7          Produtos p1 = new Produtos();
8          p1.codProd=123;
9          p1.descricaoProd="Protetor Solar";
10         p1.precoProd=79.9;
11         p1.imprimirProduto();
12     }
13 }
```

- ▶ É necessário acessar os atributos por meio dos métodos (getters e setters) criados

Usando métodos criados

```
TestaProduto1.java x Produtos.java x
1  /* Classe principal arquivo TestaProduto.java */
2  public class TestaProduto1
3  {
4      public static void main (String[] args)
5      {
6          /* instancia o objeto p1*/
7          Produtos p1 = new Produtos();
8          p1.setCodProd(123);
9          p1.setDescricaoProd("Protetor Solar");
10         p1.setPrecoProd(79.9);
11         p1.imprimirProduto();
12     }
13 }
```



Obs: o uso dos getters foi definido dentro do método worker imprimirProduto()

Observação

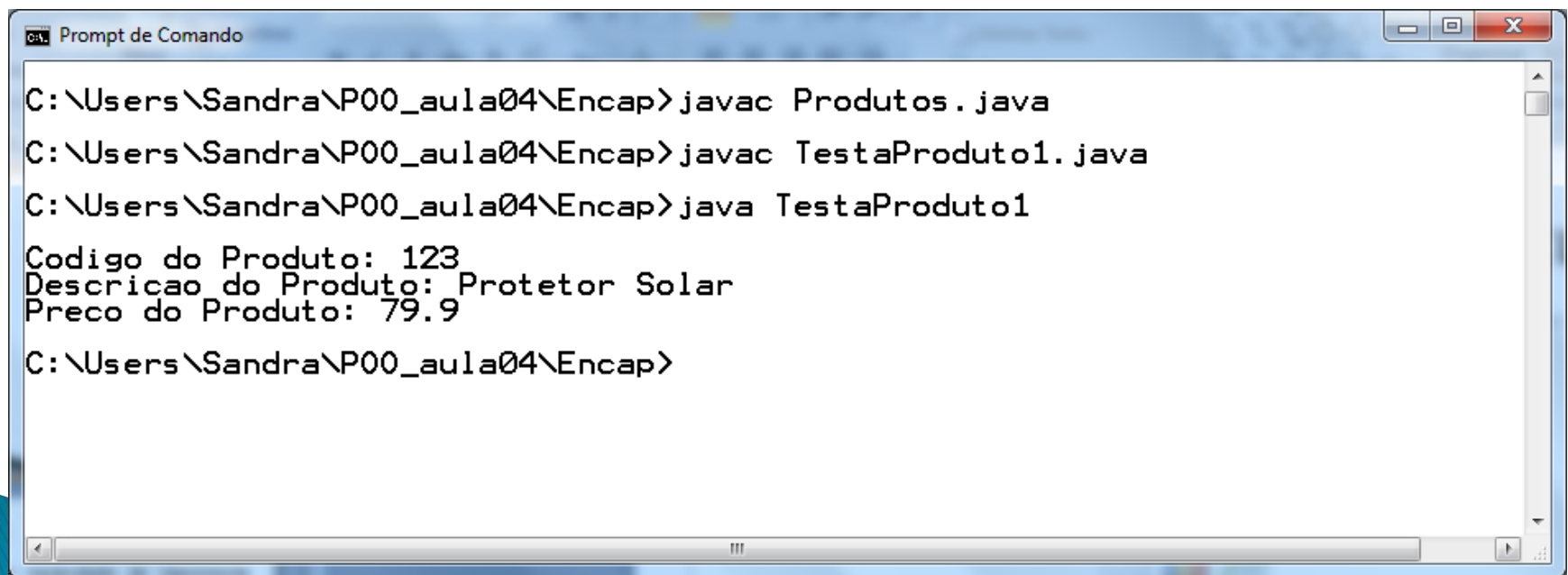
- ▶ Também é possível alterar o código do método worker definido por imprimirProduto()

```
public void imprimirProduto()    {  
    /*System.out.println("");  
    System.out.println("Codigo do produto:" + this.codProd);  
    System.out.println("Descricao do produto:" + this.descricaoProd);  
    System.out.println("Preco do produto:" + this.precoProd);  
    */  
    System.out.println("");  
    System.out.println("Codigo do Produto: " + this.getCodProd());  
    System.out.println("Descricao do Produto: " + this.getDescricaoProd());  
    System.out.println("Preco do Produto: " + this.getPrecoProd());  
}
```

Obs: o uso dos getters foi definido dentro do método worker imprimirProduto()

Visualizando o resultado

- Obs: foi criada uma cópia (TestaProduto1.java) do arquivo original TestaProduto.java apenas para apresentar as alterações necessários quando se usa o private nos atributos



```
C:\Users\Sandra\P00_aula04\Encap>javac Produtos.java
C:\Users\Sandra\P00_aula04\Encap>javac TestaProduto1.java
C:\Users\Sandra\P00_aula04\Encap>java TestaProduto1
Codigo do Produto: 123
Descricao do Produto: Protetor Solar
Preco do Produto: 79.9
C:\Users\Sandra\P00_aula04\Encap>
```

Exercício

- ▶ Considere a necessidade de implementar classes para um sistema de vendas. Implemente as classes:
 - Clientes (codCli (int), nomeCli (String), enderecoCli (String), telefoneCli (String));
 - Pedidos (numeroPed (int), dataPed (String), valorPed (double); e
 - Produtos (codProd (int), Descrição (String), Preço (double)). Para produtos use o exemplo dos slides.