

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Campus São Carlos

Análise e Desenvolvimento de Sistemas

Programação Orientada a Objetos (POOS3)

Classes e Objetos



Pablo Dalbem

dalbem@ifsp.edu.br

Objetivos

Compreender o desenvolvimento de software a partir de classes e objetos

Criar e instanciar classes de objetos

Entender a importância do encapsulamento

Saber usar um array de objetos

Conteúdo

Paradigmas de Programação

Classe

Objetos

Método construtor

Encapsulamento e modificadores de acesso

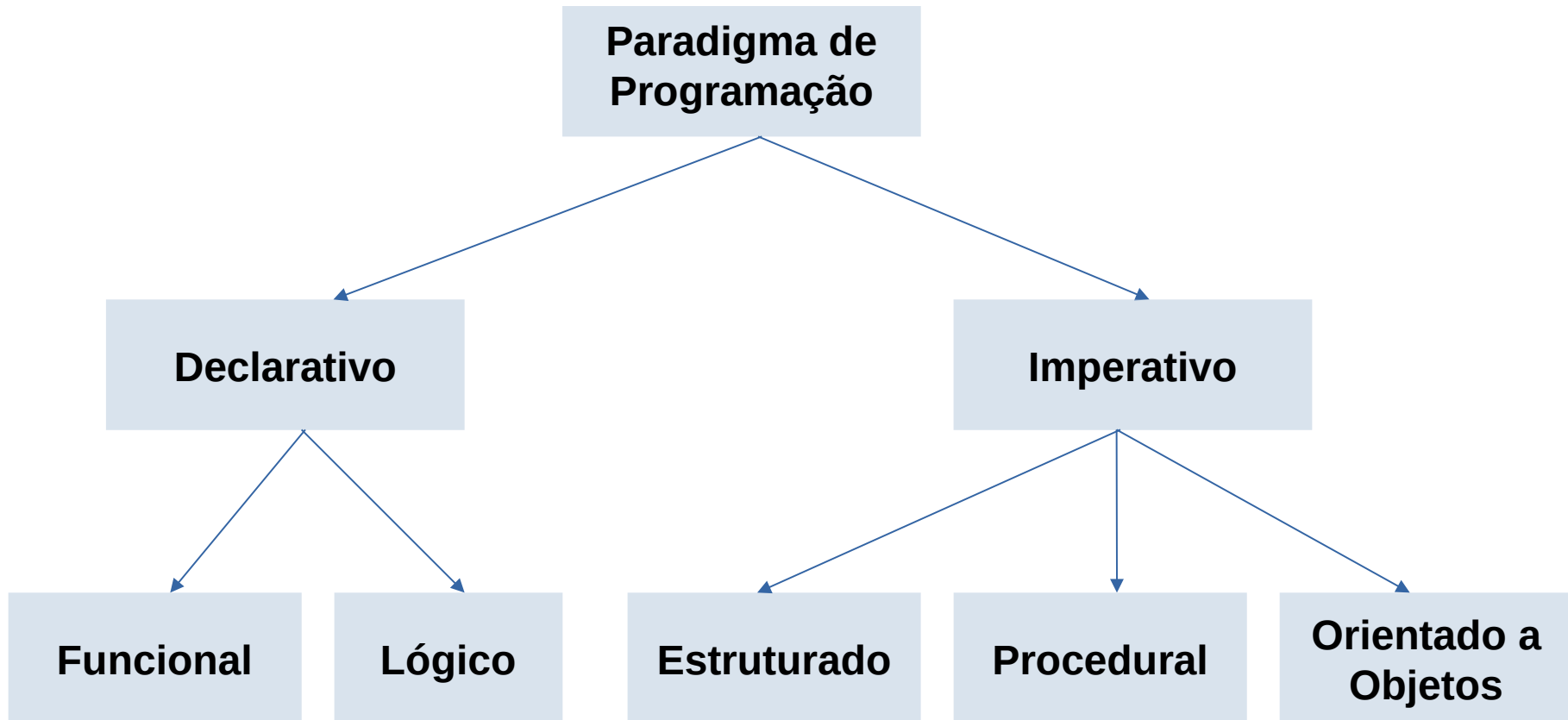
Exercício

Paradigma de Programação

Define o estilo de programação que a linguagem deve respeitar, em relação à estruturação e execução.

Corresponde a um conjunto de características que, juntas, definem como o problema é resolvido computacionalmente.

Paradigma de Programação



Paradigma de Programação

Paradigma Funcional

Baseado no Cálculo Lambda

Programas são definições de funções matemáticas e de especificações de como aplicá-las. As execuções consistem em avaliar estas funções.

Linguagens: Lisp, Haskell, Clojure

Paradigma de Programação

Paradigma Funcional – Exemplo do cálculo do fatorial em LISP

```
(defun fatorial (n)
  (cond
    ((= n 0) 1)
    (t (* n (fatorial (- n 1)))))
  )
)
```

Paradigma de Programação

Paradigma Lógico

Utiliza o cálculo de predicados

Programa declara **fatos** (dados e relações entre eles) e **cláusulas lógicas** (regras de dedução).

Por meio de um mecanismo de inferência, é possível deduzir novas verdades a partir dos fatos conhecidos.

Linguagem: Prolog

Paradigma de Programação

Paradigma Lógico – Exemplo do cálculo do fatorial em Prolog

```
fatorial (0,1).
```

```
fatorial (N,X) :- M is N - 1,  
                  fatorial(M,Y),  
                  X is N * Y.
```

Paradigma de Programação

Paradigma Estruturado

Forma de programação na qual os programas são escritos com apenas 3 estruturas:

Estrutura sequencial

Estrutura de decisão

Estrutura de repetição

Linguagens: C, PHP, Python

Paradigma de Programação

Paradigma Estruturado – Exemplo do cálculo do fatorial em C

```
int fatorial(int n){  
    int fat;  
    if ( n < 2 )  
        return 1;  
    else  
        return n * fatorial(n - 1);  
}
```

Paradigma de Programação

Paradigma Orientado a Objetos

Baseada na composição e interação entre diversas unidades de software chamadas objetos.

Linguagens: Java, C++, C#, PHP, Python

Paradigma de Programação

Paradigma Orientado a Objetos – Exemplo do cálculo do fatorial em Java

```
public class Fatorial{  
    public static void main (String[] args){  
        System.out.println(calcularFatorial(5));  
    }  
  
    public static int calcularFatorial(int n){  
        int fat;  
        if ( n < 2 )  
            return 1;  
        else  
            return n * calcularFatorial(n - 1);  
    }  
}
```

Paradigma de Programação

Pilares da Programação Orientada a Objetos:

Abstração

Encapsulamento

Herança

Polimorfismo

Paradigma de Programação

Pilares da Programação Orientada a Objetos:

Abstração

Encapsulamento

Herança

Polimorfismo

Abstração

Refere-se à capacidade de:

- identificar as entidades do mundo real (domínio do problema) relacionadas com o problema que se está tentando resolver;
- extrair destas entidades suas características e comportamentos **importantes** para a solução do problema
- representar estas entidades computacionalmente (domínio da solução).

Por meio deste conceito, é possível criar classes de objetos concisas, sem complexidade desnecessária.

Objetos

Um objeto é algo do mundo real que possui um conjunto de **atributos e comportamentos**.

Pode ser concreto ou abstrato.



Objetos

Características principais de um objeto:

- **identidade própria** – cada objeto tem um identificador único
- **estado** – conjunto de seus atributos e seus valores atuais
- **comportamento** – conjunto de operações que o objeto oferece



Carro 1

Cor: verde

Marca: Ford

Ligado: não

Ligar

Desligar

Acelerar

Frear



Carro 2

Cor: vermelho

Marca: VW

Ligado: sim

Ligar

Desligar

Acelerar

Frear

Classes de Objetos

Definem de forma conceitual um conjunto de objetos que compartilham das mesmas características e comportamentos.

Classe Carro

Cor

Marca

Ligado

Ligar

Desligar

Acelerar

Frear



Objetos

Classes de Objetos

Servem como um **molde** para a criação de diversos objetos com características particulares.

Definir uma classe significa formalizar um tipo de dado e todas as operações associadas a esse tipo.

Declarar objetos significa criar variáveis do tipo definido e atribuir/modificar os valores dessas variáveis por meio das operações.

Classe em Java

```
public class Carro {
```

```
    String marca;  
    String cor;  
    boolean ligado;
```

Atributos

```
    public void acelerar(){  
        //TODO  
    }
```

```
    public void frear(){  
        //TODO  
    }
```

```
    public void ligar(){  
        //TODO  
    }
```

```
    public void desligar(){  
        //TODO  
    }  
}
```

Métodos

Classe em Java

```
public class Livro {  
    String titulo;  
    String autor;  
    int anoPublicacao;  
    double preco;
```



Atributos

```
    public void atualizarPreco(int reajuste) {  
        preco = preco + (preco * reajuste/100);  
    }  
}
```



Método

Método Construtor

O construtor é um método especial que permite a criação de uma instância (objeto) de uma classe.

Métodos construtores não possuem tipo de retorno.

Uma mesma classe pode possuir múltiplos construtores.

Construtores devem possuir o mesmo nome da classe e toda classe tem pelo menos um construtor.


Se (e somente se) o programador não implementar um construtor, o Java criará um construtor padrão (que não recebe argumentos) de forma implícita.

Método Construtor

```
public class Livro {  
    String titulo;  
    String autor;  
    int anoPublicacao;  
    double preco;
```

```
    public Livro(){}  
  
    public Livro(String titulo, String autor, int anoPublicacao, double preco) {  
        this.titulo = titulo;  
        this.autor = autor;  
        this.anoPublicacao = anoPublicacao;  
        this.preco = preco;  
    }  
  
    public void atualizarPreco(int reajuste) {  
        preco = preco + (preco * reajuste/100);  
    }  
}
```

Método
construtor
vazio



Método
construtor



Criando um Objeto

Para criar (instanciar) um objeto, utiliza-se o operador ***new***

Exemplo 1 – Construtor vazio.

```
public class Principal {  
    public static void main(String[] args) {  
        Livro livro1 = new Livro();  
  
        livro1.titulo = "Java";  
        livro1.autor = "Deitel";  
        livro1.anoPublicacao = 2016;  
        livro1.preco = 350.00;  
    }  
}
```

Objeto livro1 foi
instanciado usando
construtor vazio

Atribuição de
valores para os
atributos de livro1

Criando um Objeto

Para criar (instanciar) um objeto, utiliza-se o operador ***new***

Exemplo 2 – Construtor passando parâmetros.

```
public class Principal {  
    public static void main(String[] args) {  
        Livro livro1 = new Livro("Java", "Deitel", 2016, 350.00);  
    }  
}
```

Criando um Objeto

Para criar (instanciar) um objeto, utiliza-se o operador ***new***

Exemplo 3 – Declaro o objeto primeiro e o instancio depois (independente do tipo de construtor).

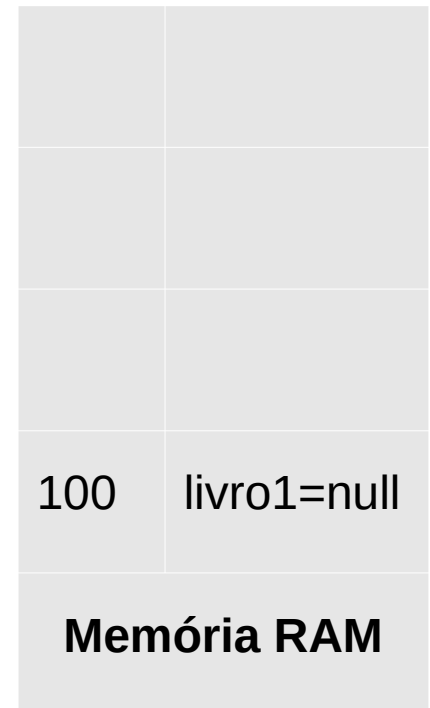
```
public class Principal {  
    public static void main(String[] args) {  
        Livro livro1;  
        livro1 = new Livro("Java", "Deitel", 2016, 350.00);  
    }  
}
```

Criando um Objeto

Detalhe na criação de um objeto

Objeto livro1 foi declarado.
Na memória RAM, um espaço
é reservado.

```
public class Principal {  
    public static void main(String[] args) {  
        Livro livro1;  
        livro1 = new Livro("POO", "Sierra", 2018, 150);  
    }  
}
```



Criando um Objeto

Detalhe na criação de um objeto

```
public class Principal {  
    public static void main(String[] args) {  
        Livro livro1;  
        livro1 = new Livro("POO", "Sierra", 2018, 150);  
    }  
}
```

Operador *new* alocou memória para guardar o objeto e retornou o endereço para livro1

700	titulo="POO" autor="Sierra" AnoPublicacao=2018 preco=150
100	livro1 = 700
Memória RAM	

Invocando Métodos

```
public class Principal {  
    public static void main(String[] args) {  
        Livro livro2 = new Livro("Java", "Deitel", 2016, 350.00);  
        livro2.atualizarPreco(10);  
        System.out.println(livro2.preco);  
  
        Livro livro3;  
        livro3.atualizarPreco(5); //ERRO! livro3 não foi instanciado, vale null  
  
        Livro livro1 = new Livro();  
        livro1.atualizarPreco(10);  
        System.out.println(livro1.preco); //Qual a saída?  
    }  
}
```

Considere o Problema

Como saber quantas instâncias da classe Livro existem no nosso programa? A solução abaixo funcionaria?

```
public class Livro {  
    String titulo;  
    String autor;  
    int anoPublicacao;  
    double preco;  
    int quantidadeObjetos=0;  
  
    public Livro() { this.quantidadeObjetos++; }  
  
    public Livro(String titulo, String autor, int anoPublicacao, double preco) {  
        this.titulo = titulo;  
        this.autor = autor;  
        this.anoPublicacao = anoPublicacao;  
        this.preco = preco;  
        this.quantidadeObjetos++;  
    }  
  
    // Restante da classe
```

Considere o Problema

Como saber quantas instâncias da classe Livro existem no nosso programa? A solução abaixo funcionaria?

```
public class Principal {  
    public static void main(String[] args) {  
        Livro livro1 = new Livro("Java", "Deitel", 2016, 350.00);  
        Livro livro2 = new Livro("POO", "Sierra", 2018, 150.00);  
    }  
}
```

titulo: Java
autor: Deitel
anoPublicacao: 2016
preco: 350.00
quantidadeObjetos: 1

titulo: POO
autor: Sierra
anoPublicacao: 2018
preco: 150.00
quantidadeObjetos: 1

Já percebemos que a solução não funciona, pois temos 2 instâncias da classe Livro, porém o atributo quantidadeObjetos vale 1.

Considere o Problema

Vimos até agora que cada objeto criado possui seus próprios atributos e métodos. Assim, a alteração no valor de um atributo de um objeto não afeta o valor do mesmo atributo em outros objetos. Neste caso, dizemos que os atributos e métodos pertencem à instância (objeto).

```
public class Livro {  
    String titulo;  
    String autor;  
    int anoPublicacao;  
    double preco;  
    int quantidadeObjetos=0;  
    ...  
}
```

```
public class Principal {  
    public static void main(String[] args) {  
        Livro livro1 = new Livro("Java", "Deitel", 2016, 350.00);  
        Livro livro2 = new Livro("POO", "Sierra", 2018, 150.00);  
    }  
}
```

Método e Atributo *static*

O método ou atributo **static** são os mesmos para todos os objetos da classe. São compartilhados. Agora, a alteração no valor de um atributo static fica visível para todos os atributos instanciados. Portanto, dizemos que métodos ou atributos **static** são da classe (e não da instância).

Quando o método ou o atributo for **static**, não é preciso instanciar um objeto para acessá-lo.

Exemplos de métodos static da API Java:

```
public class Principal {  
    public static void main(String[] args) {  
        double raiz = Math.sqrt(4);  
        double valor = Math.pow(2,2);  
    }  
}
```

Método e Atributo *static*

Voltando ao exemplo da classe Livro:

```
public class Livro {  
    String titulo;  
    String autor;  
    int anoPublicacao;  
    double preco;  
    static int quantidadeObjetos=0;  
  
    public Livro() { this.quantidadeObjetos++; }  
  
    public Livro(String titulo, String autor, int anoPublicacao, double preco) {  
        this.titulo = titulo;  
        this.autor = autor;  
        this.anoPublicacao = anoPublicacao;  
        this.preco = preco;  
        this.quantidadeObjetos++;  
    }  
}
```

// Restante da classe.

Método e Atributo *static*

Voltando ao exemplo da classe Livro:

```
public class Principal {  
    public static void main(String[] args) {  
        Livro livro1 = new Livro("Java", "Deitel", 2016, 350.00);  
        Livro livro2 = new Livro("POO", "Sierra", 2018, 150.00);  
        System.out.println( Livro.quantidadeObjetos );  
    }  
}
```

titulo: Java
autor: Deitel
anoPublicacao: 2016
preco: 350.00

titulo: POO
autor: Sierra
anoPublicacao: 2018
preco: 150.00

quantidadeObjetos: 2

Sobrecarga de Método

Sobrecarga (*overloading*) de método é quando a classe possui métodos com mesmo nome, mas assinaturas diferentes.

Oferece maior flexibilidade na hora de invocar um determinado comportamento da classe.

Por exemplo, na API Java, a classe `ArrayList` possui os métodos:

`add(Object e)`

`add(int index, Object e)`

`remove(int index)`

`remove(Object e)`

Sobrecarga de Método

Outro exemplo: **Sobrecarga de método construtor**

```
public Livro(){} 
```

```
public Livro(String titulo, String autor, int anoPublicacao, double preco) {  
    this.titulo = titulo;  
    this.autor = autor;  
    this.anoPublicacao = anoPublicacao;  
    this.preco = preco;  
}
```

```
public Livro(String titulo, double preco) {  
    this.titulo = titulo;  
    this.preco = preco;  
}
```

Arrays de Objetos

Para criar o array:

```
Livro[] livros = new Livro[5];
```

Para inserir referências de objetos da classe Livro no array:

```
Livro[] livros = new Livro[5];
```

```
livros[0] = new Livro("PHP", "João", 2020, 100);
```

```
livros[1] = new Livro("C++", "Stroustrup", 2013, 600);
```

```
Livro livroJava = new Livro("Java", "Deitel", 2016, 350.00);
```

```
livros[2] = livroJava;
```

```
livros[3] = new Livro();
```

```
livros[3].titulo = "Kotlin";
```

```
livros[4].titulo = "C#"; // ERRO! Posição 4 do vetor aponta para null
```

Arrays de Objetos

É comum ouvirmos "array de objetos". Porém, quando criamos uma array de alguma classe, ele tem referências para o endereço dos objetos.

```
Livro[] livros = new Livro[5];
```

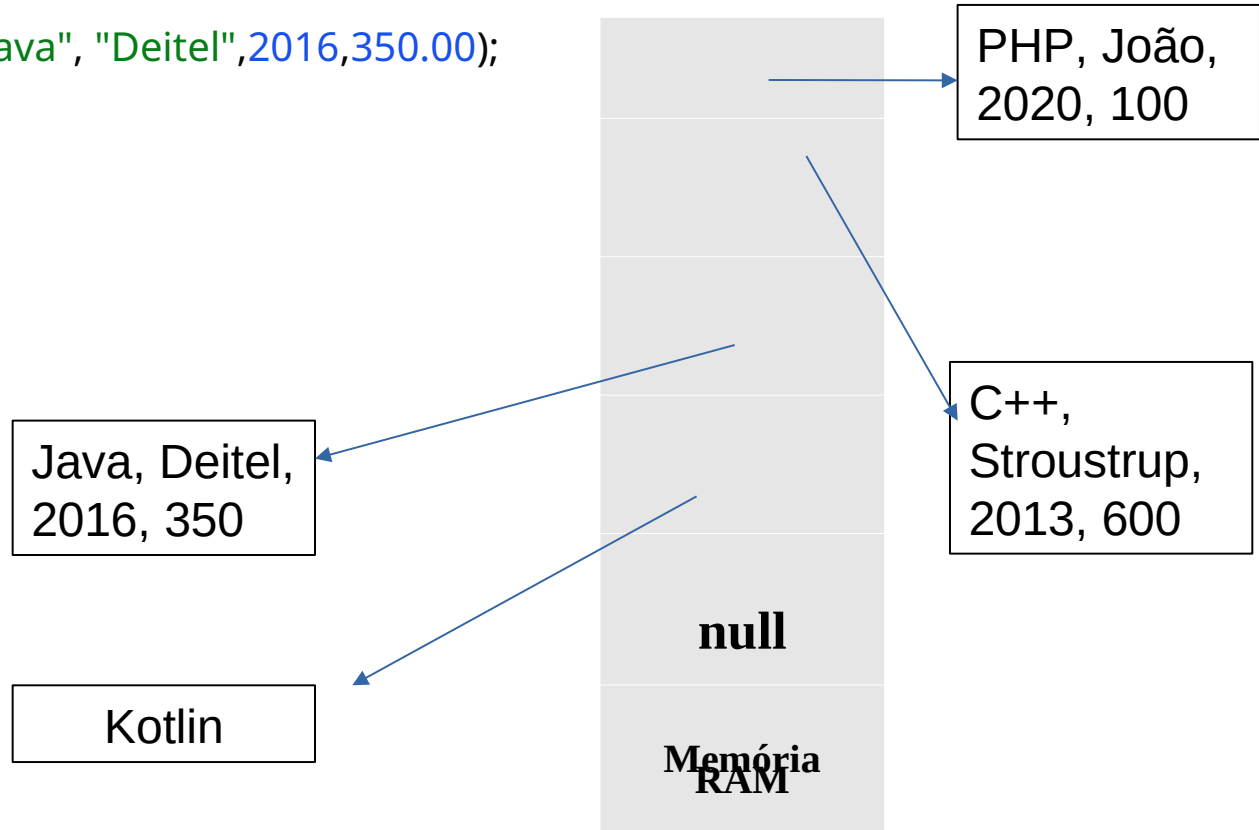


Arrays de Objetos

```
Livro[] livros = new Livro[5];  
livros[0] = new Livro("PHP", "João", 2020, 100);  
livros[1] = new Livro("C++", "Stroustrup", 2013, 600);
```

```
Livro livroJava = new Livro("Java", "Deitel", 2016, 350.00);  
livros[2] = livroJava;
```

```
livros[3] = new Livro();  
livros[3].titulo = "Kotlin";
```



Paradigma de Programação

Pilares da Programação Orientada a Objetos:

Abstração

Encapsulamento

Herança

Polimorfismo

Encapsulamento

Mecanismo que permite ocultar detalhes da implementação da classe.

Por meio dele, é possível esconder os atributos e métodos da nossa classe que não devem ser acessados por outras classes.

Desta forma, evita-se o acesso indevido ao estado do objeto.

Encapsulamento

Modificadores de acesso viabilizam o encapsulamento de métodos e atributos de uma classe:

public

Acessados de qualquer lugar do código.

private

Acessados somente dentro do corpo da mesma classe.

protected

Acessados pelas classes do mesmo pacote ou classes herdadas.

Sem modificador

Acessados pelas classes do mesmo pacote

Encapsulamento

No exemplo anterior, os atributos da classe Livro estão sem modificador de acesso:

```
public class Livro {  
    String titulo;  
    String autor;  
    int anoPublicacao;  
    double preco;  
    ....  
}
```

Por isso, no arquivo Principal.java conseguimos acessar os atributos diretamente. **Entretanto, acessar os atributos diretamente não é boa prática de programação:**

```
public class Principal {  
    public static void main(String[] args) {  
        Livro livro2 = new Livro("Java", "Deitel", 2016, 350.00);
```

```
        System.out.println(livro2.preco);
```

```
        livro2.titulo = "Java: Como Programar";
```

```
    }
```

```
}
```

Encapsulamento

Boa prática de programação: utilize sempre o modificador **private**, a menos que tenha uma boa razão para não fazer isso.

```
public class Livro {  
    private String titulo;  
    private String autor;  
    private int anoPublicacao;  
    private double preco;  
    ....  
}
```

Os atributos definidos como **private**, deverão ser acessados por meio de métodos

Consulta (get+nome do atributo): utilizado para obter o valor de um atributo

Alteração (set+nome do atributo): utilizado para atribuir um valor ao atributo

```

public class Livro {
    private String titulo;
    private String autor;
    private int anoPublicacao;
    private double preco;

    public String getTitulo() {
        return titulo;
    }

    public void setTitulo(String titulo) {
        this.titulo = titulo;
    }

    public String getAutor() {
        return autor;
    }

    public void setAutor(String autor) {
        this.autor = autor;
    }

    public int getAnoPublicacao() {
        return anoPublicacao;
    }

    public void setAnoPublicacao(int anoPublicacao) {
        this.anoPublicacao = anoPublicacao;
    }
}

```

```

    public double getPreco() {
        return preco;
    }

    public void setPreco(double preco) {
        this.preco = preco;
    }

    public Livro(){}

    public Livro(String titulo, String autor, int
anoPublicacao, double preco) {
        this.titulo = titulo;
        this.autor = autor;
        this.anoPublicacao = anoPublicacao;
        this.preco = preco;
    }

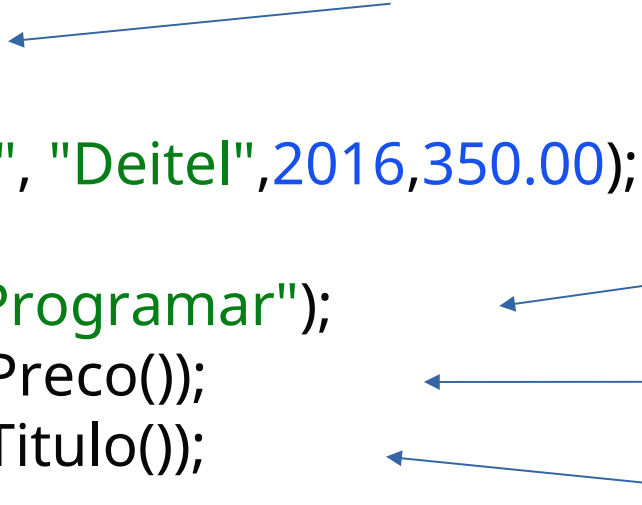
    public void atualizarPreco(int reajuste) {
        preco = preco + (preco * reajuste/100);
    }
}

```

**Classe Livro com atributos
*private***

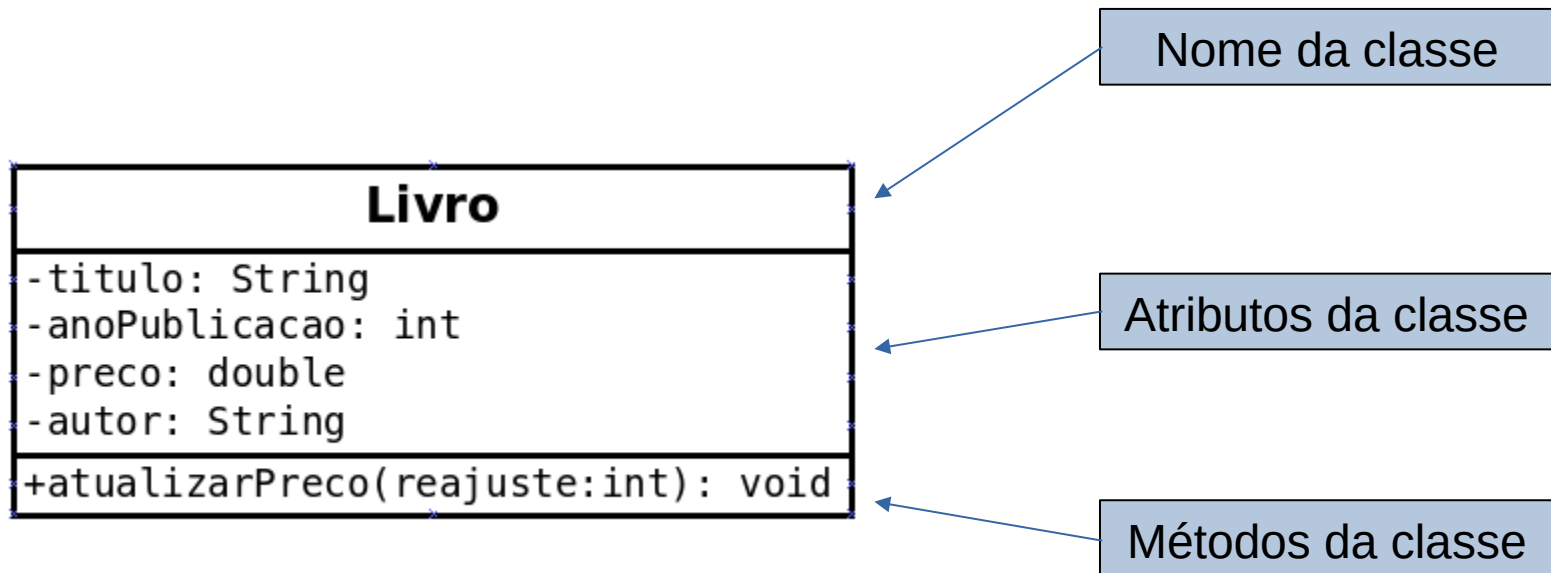
Classe Principal.java

```
public class Principal {  
    public static void main(String[] args) {  
        Livro livro1 = new Livro();  
        livro1.setTitulo("POO");  
  
        Livro livro2 = new Livro("Java", "Deitel", 2016, 350.00);  
        livro2.atualizarPreco(10);  
        livro2.setTitulo("Java: Como Programar");  
        System.out.println(livro2.getPreco());  
        System.out.println(livro2.getTitulo());  
    }  
}
```



Representação Diagramática

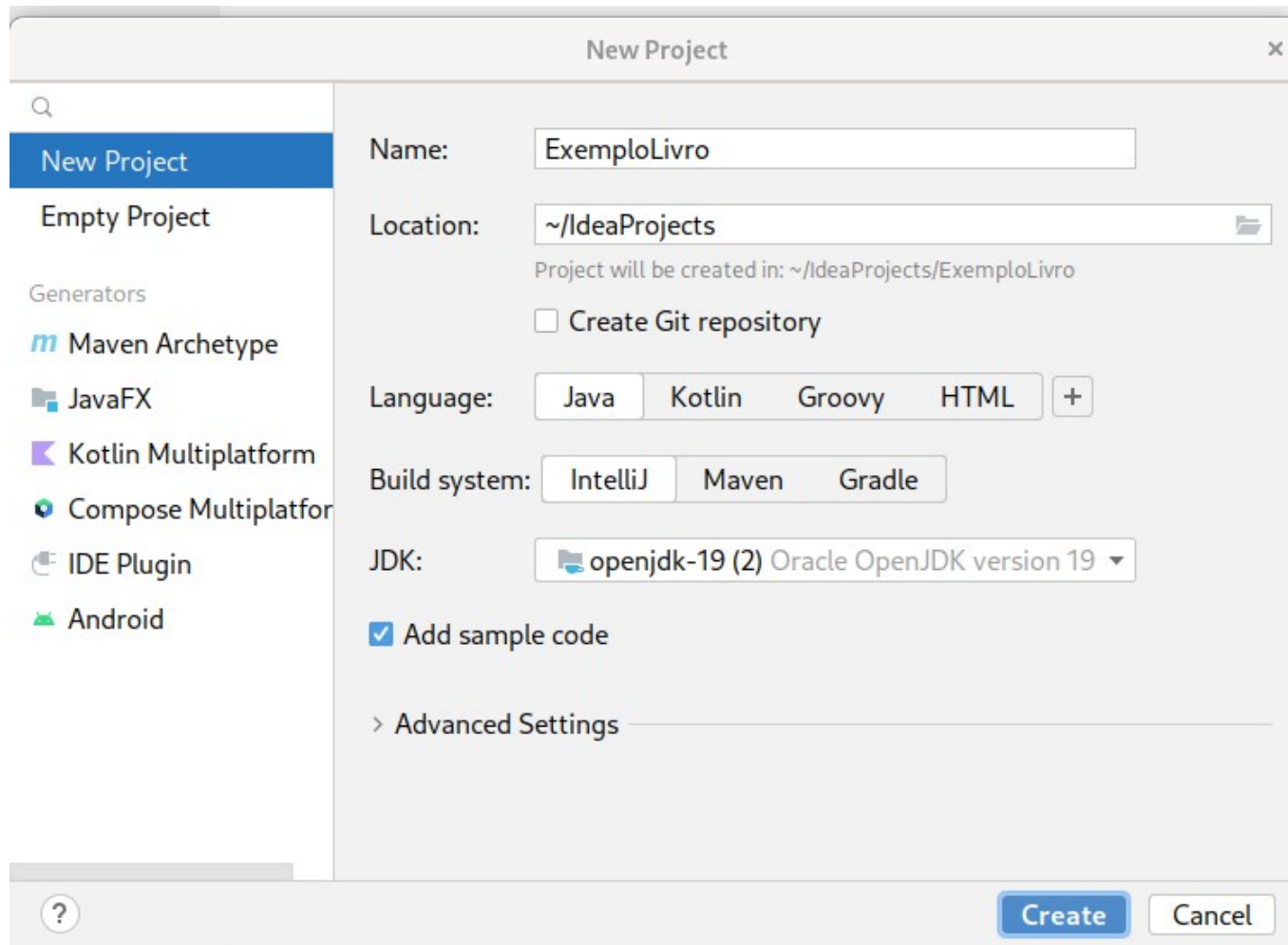
Representação diagramática da classe Livro



Modificadores de acesso aos membros da classe:
Público (+), Privado (-), Protegido (#) e Pacote (~)

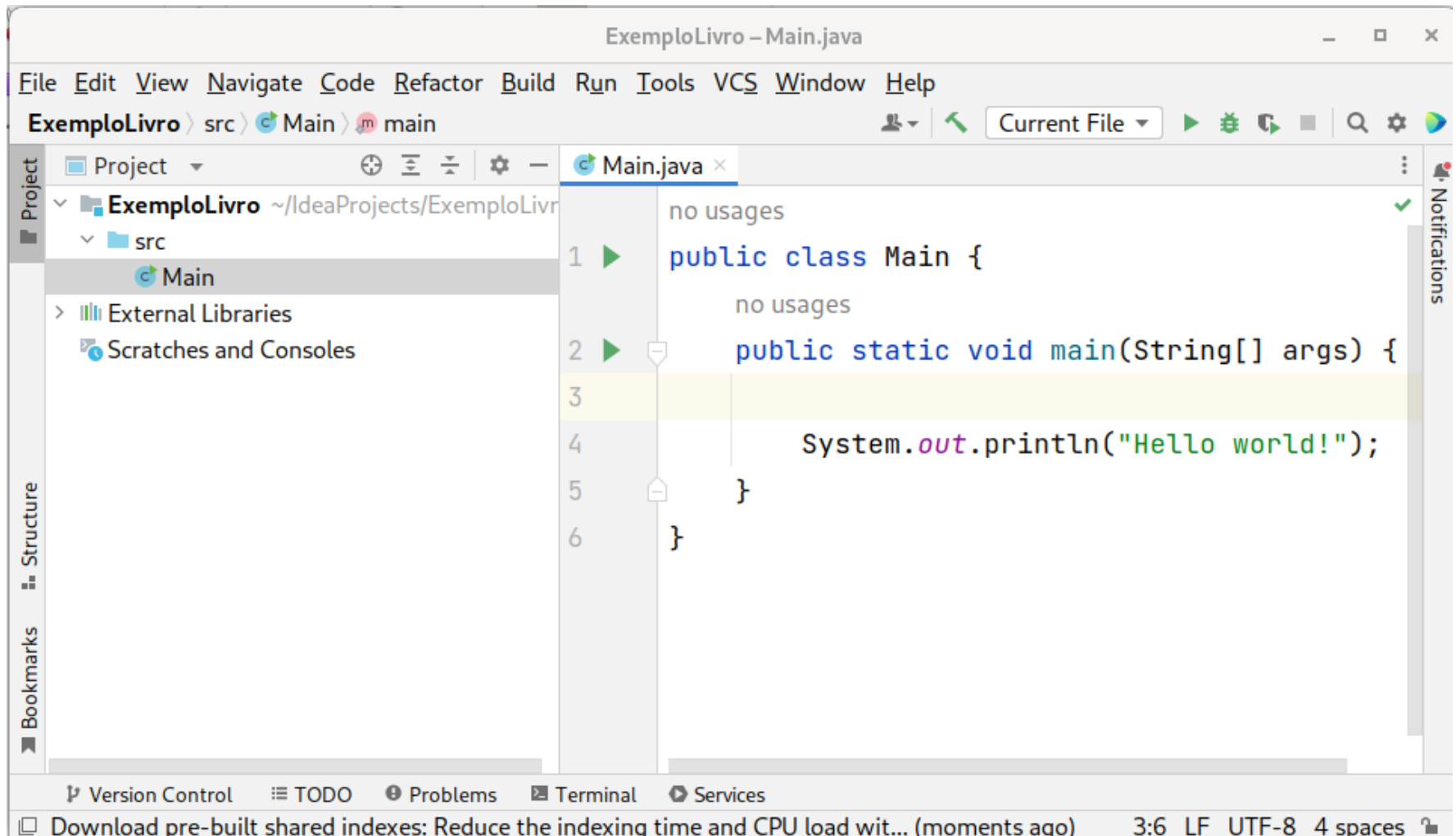
No IntelliJ

Crie um novo projeto no IntelliJ



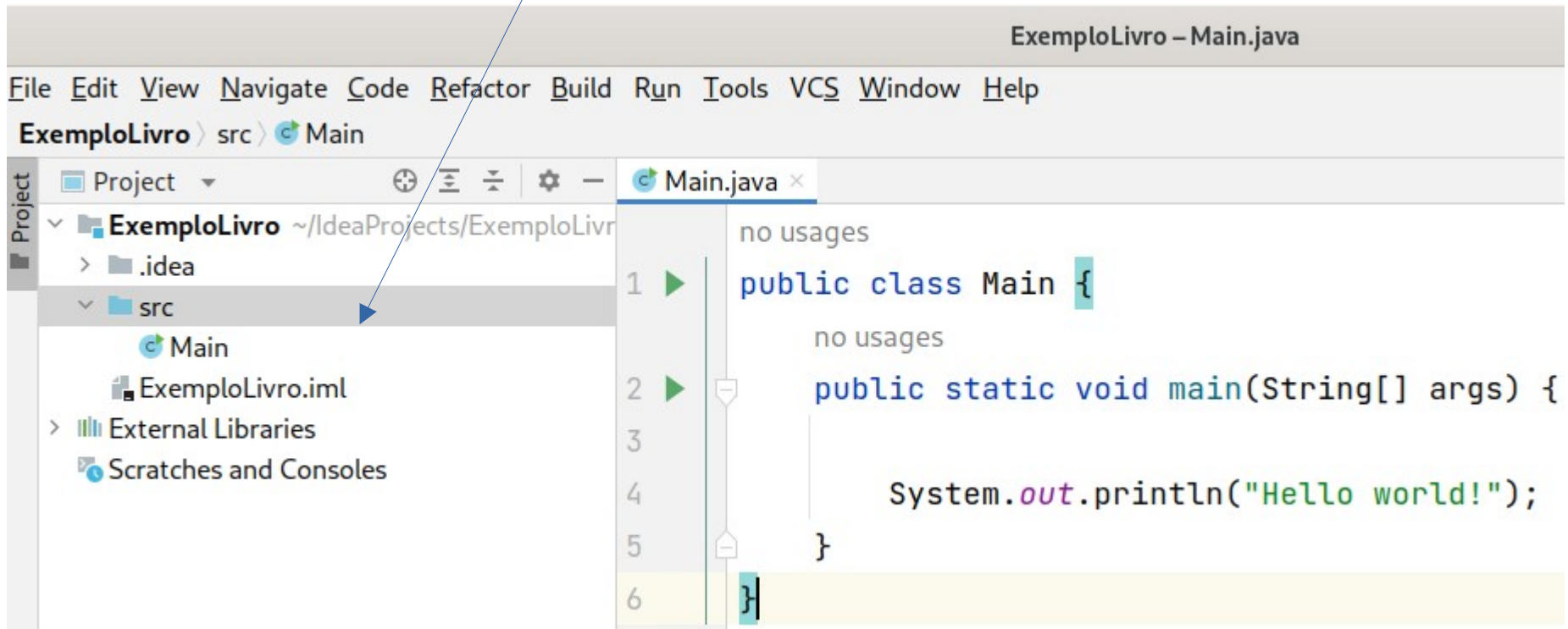
No IntelliJ

Projeto criado. No próximo slide, vamos adicionar a classe Livro



No IntelliJ

No **Project Explorer** (lado esquerdo), clique com botão direito do mouse sobre a pasta **src**

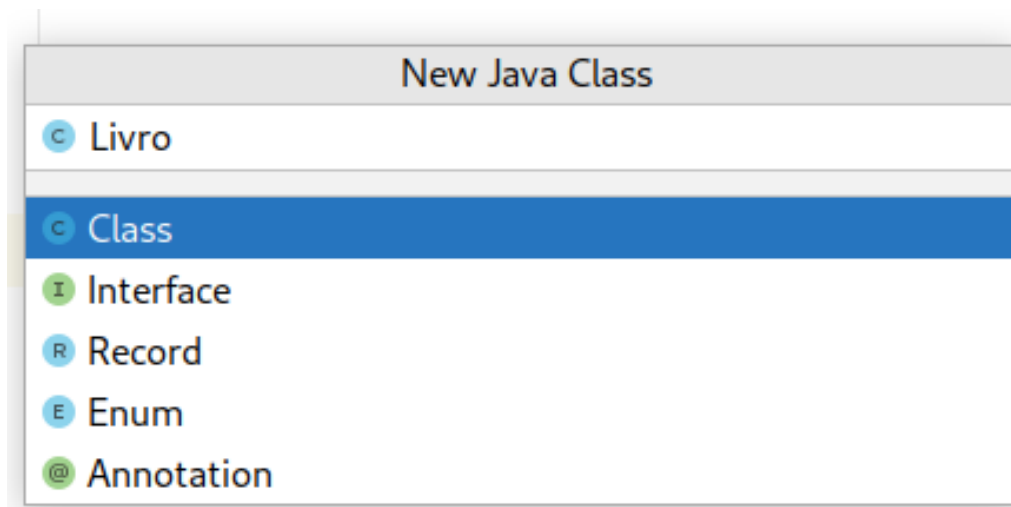


No menu que irá se abrir, escolha **New** → **Java Class**

No IntelliJ

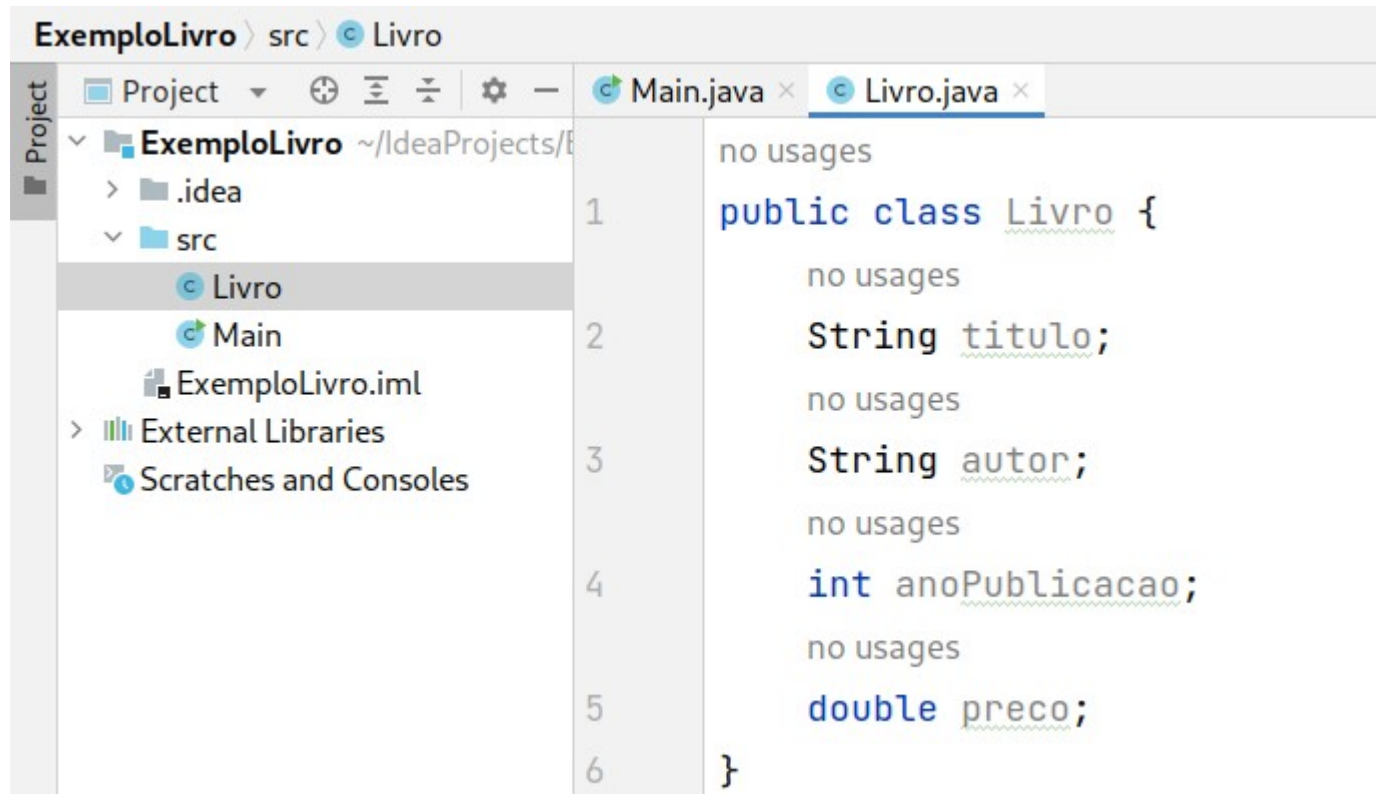
Na janela que irá se abrir, digite o nome da nova classe. No nosso exemplo, **Livro**

Tecla «enter»



No IntelliJ

No editor (lado direito) da classe Livro, adicione primeiramente os atributos



No IntelliJ

Agora, vamos encapsular os atributos e gerar o métodos get e set.

Clique no Menu **Refactor** → **Encapsulate Fields...**

Encapsulate Fields - Livro

Fields to Encapsulate

Field	Getter	Setter
<input checked="" type="checkbox"/> f 🔒 titulo:String	<input checked="" type="checkbox"/> m 📄 getTitulo	<input checked="" type="checkbox"/> m 📄 setTitulo
<input checked="" type="checkbox"/> f 🔒 autor:String	<input checked="" type="checkbox"/> m 📄 getAutor	<input checked="" type="checkbox"/> m 📄 setAutor
<input checked="" type="checkbox"/> f 🔒 anoPublicacao:int	<input checked="" type="checkbox"/> m 📄 getAnoPublicacao	<input checked="" type="checkbox"/> m 📄 setAnoPublicacao
<input checked="" type="checkbox"/> f 🔒 preco:double	<input checked="" type="checkbox"/> m 📄 getPreco	<input checked="" type="checkbox"/> m 📄 setPreco

Encapsulate

- ☒ Get access
- ☒ Set access
- ☒ Use accessors even when field is accessible

Encapsulated Fields' Visibility

- ☒ Private
- ☐ Package local
- ☐ Protected
- ☐ As is

Accessors' Visibility

- ☒ Public
- ☐ Protected
- ☐ Package local
- ☐ Private

Buttons: **Refactor** Preview Cancel

Marque todos os atributos

Clique em Refactor

Pronto, a classe Livro já está com todos os atributos encapsulados e com os métodos get e set criados.

```
1  public class Livro {  
2      2 usages  
3      private String titulo;  
4      2 usages  
5      private String autor;  
6      2 usages  
7      private int anoPublicacao;  
8      2 usages  
9      private double preco;  
10     no usages  
11     public String getTitulo() {  
12         return titulo;  
13     }  
14     no usages  
15     public void setTitulo(String titulo) {  
16         this.titulo = titulo;  
17     }  
18     no usages  
19     public String getAutor() {  
20         return autor;  
21     }  
22     no usages  
23     public void setAutor(String autor) {  
24         this.autor = autor;  
25     }  
26 }
```

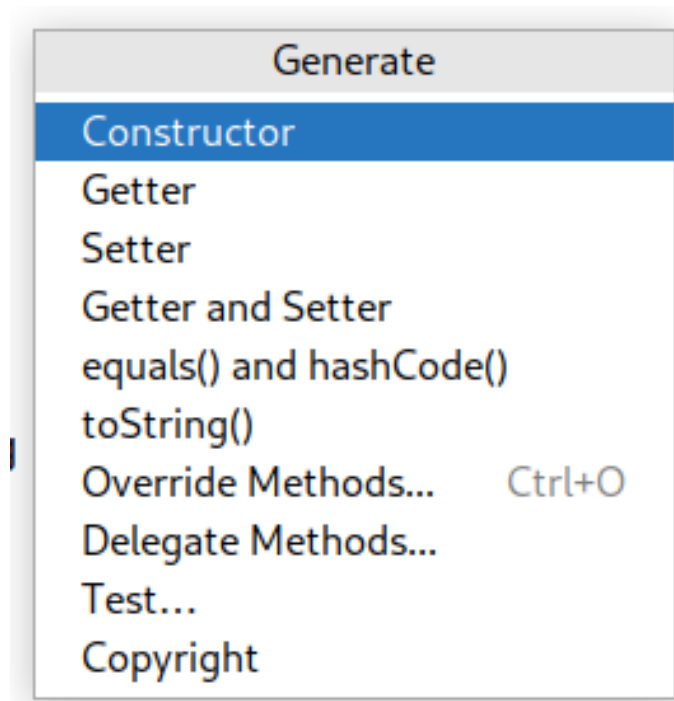
```
18     public int getAnoPublicacao() {  
19         return anoPublicacao;  
20     }  
21     no usages  
22     public void setAnoPublicacao(int anoPublicacao) {  
23         this.anoPublicacao = anoPublicacao;  
24     }  
25     no usages  
26     public double getPreco() {  
27         return preco;  
28     }  
29     no usages  
30     public void setPreco(double preco) {  
31         this.preco = preco;  
32     }  
33 }
```


No IntelliJ

Agora, vamos criar os métodos construtores.

Clique no menu **Code** → **Generate...**

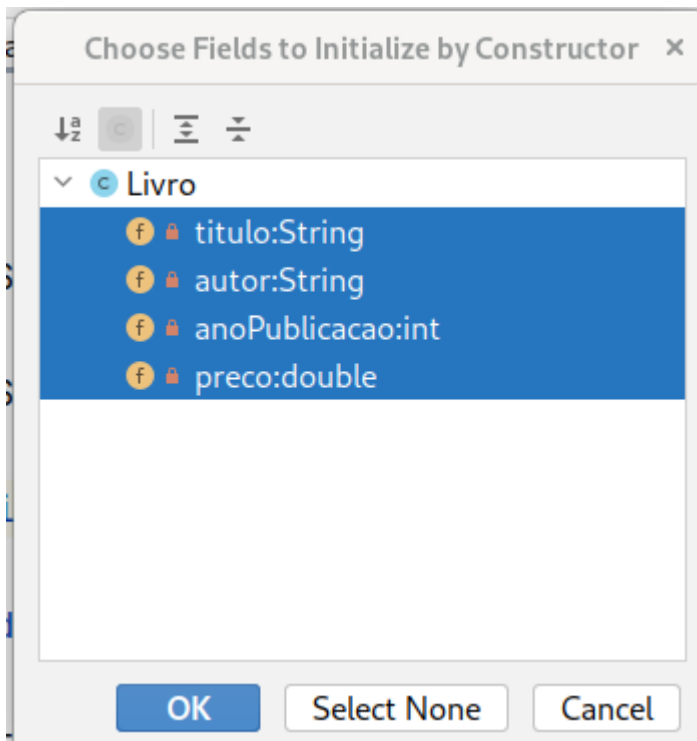
Selecione **Constructor** e tecle «enter»



No IntelliJ

Na janela que irá se abrir, selecione os atributos que deseja serem passados como parâmetro para o método construtor.

Aqui, vamos passar todos os atributos. Selecione-os e clique em **OK**

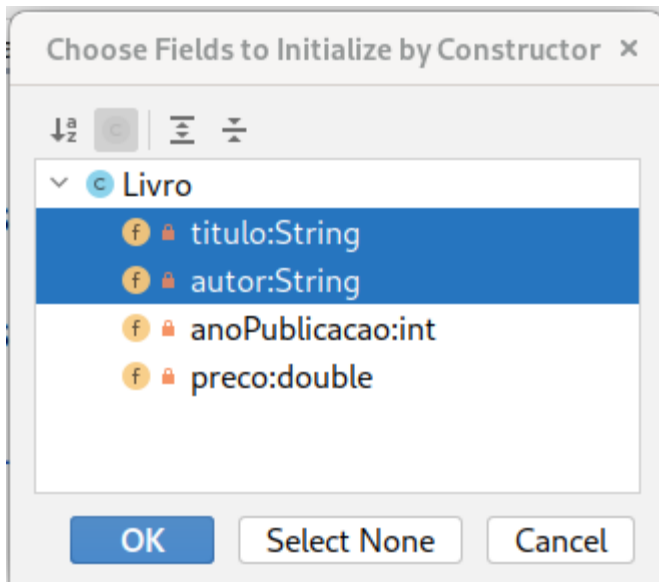


Código Gerado:

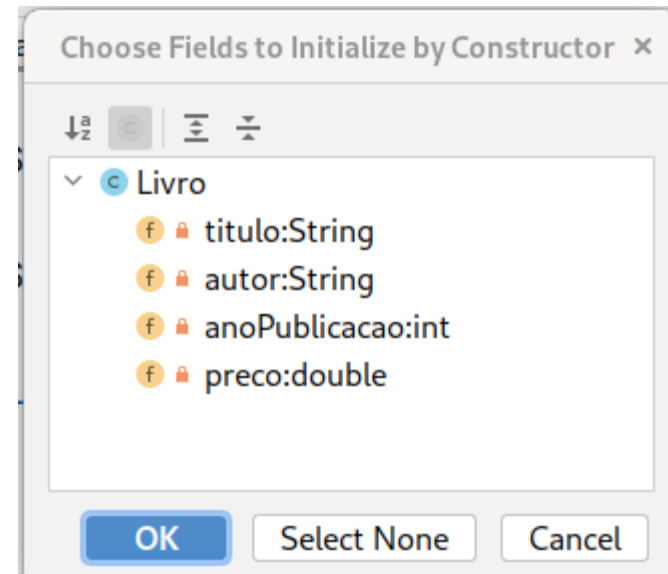
```
public Livro(String titulo, String autor,  
              int anoPublicacao, double preco) {  
    this.titulo = titulo;  
    this.autor = autor;  
    this.anoPublicacao = anoPublicacao;  
    this.preco = preco;  
}
```

No IntelliJ

Podemos repetir a operação alterando a seleção dos atributos, a fim de criarmos os métodos construtores necessários. Exemplos:



```
public Livro(String titulo, String autor) {  
    this.titulo = titulo;  
    this.autor = autor;  
}
```



```
public Livro() {  
}
```

No IntelliJ

Por fim, implementamos o método ***atualizarPreco(int reajuste)***

```
public void atualizarPreco(int reajuste) {  
    preco = preco + (preco * reajuste/100);  
}
```

No IntelliJ

Na classe Main, testamos a classe Livro

```
1 ▶ public class Main {  
    no usages  
2 ▶ ▶ public static void main(String[] args) {  
3     Livro livro1 = new Livro();  
4     livro1.setTitulo("P00");  
5  
6     Livro livro2 = new Livro( titulo: "Java para Iniciantes", autor: "Herbert Schildt");  
7     livro2.setAnoPublicacao(2015);  
8  
9     Livro livro3 = new Livro( titulo: "Java", autor: "Deitel",  
10         anoPublicacao: 2016, preco: 350.00);  
11     livro3.atualizarPreco( reajuste: 10);  
12     livro3.setTitulo("Java: Como Programar");  
13  
14     System.out.println(livro1.getTitulo());  
15     System.out.println(livro2.getAnoPublicacao());  
16     System.out.println(livro3.getPreco());  
17 }  
18 }
```

Exercício I

Implemente a classe Paciente, com os atributos nome, peso e altura. Crie o método calcularIMC(), para calcular o índice de massa corporal do paciente de acordo com a fórmula:

$$IMC = \frac{\text{peso}}{\text{altura}^2}$$

Peso é fornecido em quilos e altura em metro

IMC	Classificação
< 18,5	Abaixo do peso
18,6 a 24,9	Peso normal
25 a 29,9	Acima do peso
30 a 34,9	Obesidade I
35 a 39,9	Obesidade II
> 40	Obesidade III

Exercício II

Implemente a classe Funcionario com os atributos nome e salario.

Em seguida, faça um vetor de 5 posições para armazenar objetos da classe Funcionario, lidos via teclado.

Calcule a média dos salários. Mostre o nome dos funcionários que ganham acima desta média.

Por fim, reajuste o salário dos funcionários. Para aqueles que estão abaixo da média salarial, dê um reajuste de 10%. Para quem está acima ou igual a média, o reajuste é de apenas 5%.