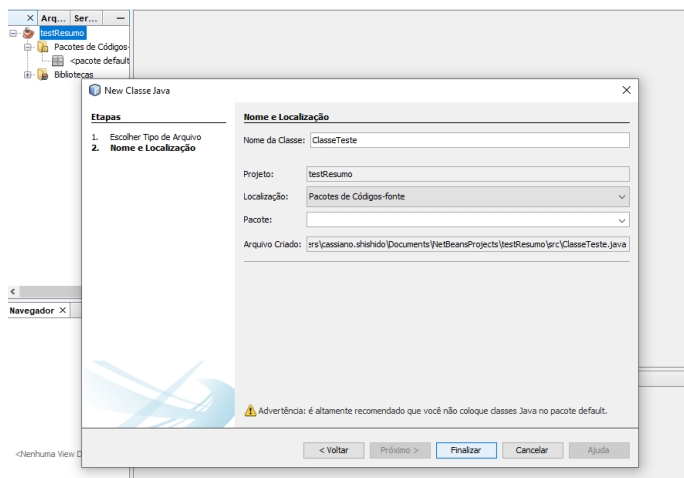


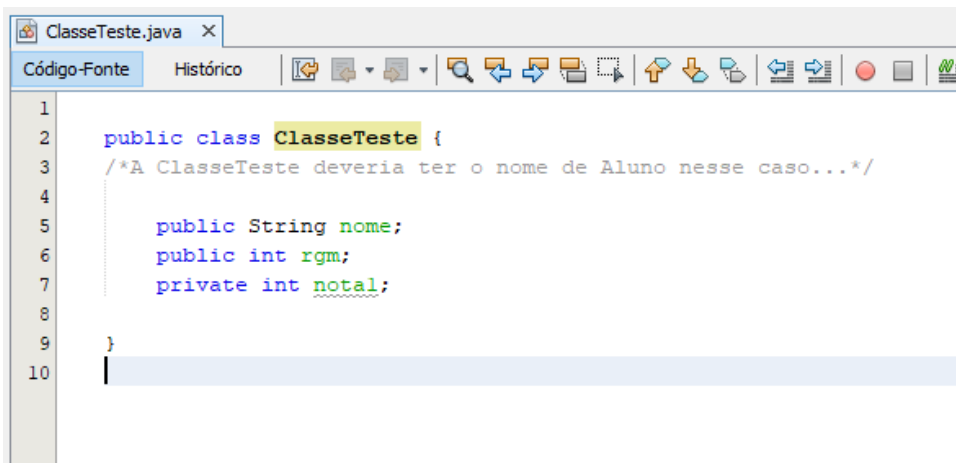
Resumo – Revisão**• POO****1) Classe**

Uma classe é um conjunto de características (atributos, métodos...) que definem o conjunto de objetos pertencentes a ela. Uma classe geralmente representa um substantivo, exemplo: Aluno, Funcionário, Móvel, Escola... entre outros. Deve-se atentar para sempre escrever o nome da classe no singular e com a inicial maiúscula.

**2) Atributos**

São as características de um objeto, tendo como por exemplo o objeto sendo Aluno, os atributos poderiam ser: nome, rgm, idade, curso, nota, nota2, media, situacao...

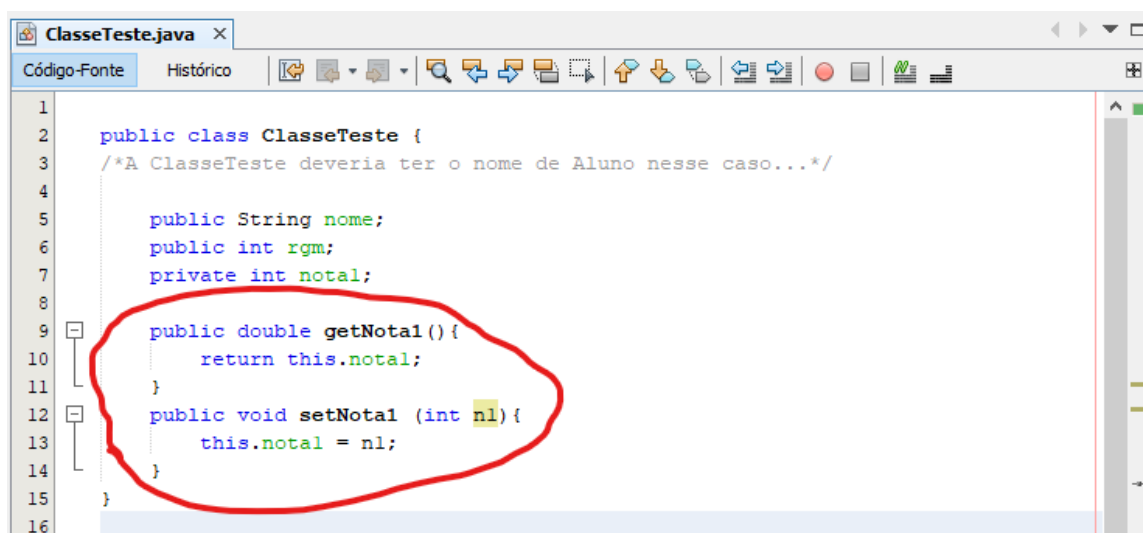
Os atributos podem ser públicos (visíveis para quaisquer classes) ou privados (só visível à própria classe) e, se privados, podem ser acessados para utilização em outra classe se feito o Encapsulamento.



```
1
2 public class ClasseTeste {
3     /*A ClasseTeste deveria ter o nome de Aluno nesse caso...*/
4
5     public String nome;
6     public int rgm;
7     private int notal;
8
9 }
10
```

3) Encapsulamento (atributos privados)

Se faz possível acessar e manipular um atributo privado através do Encapsulamento, utilizando o GET (busca o valor do atributo) e o SET (altera o seu conteúdo).

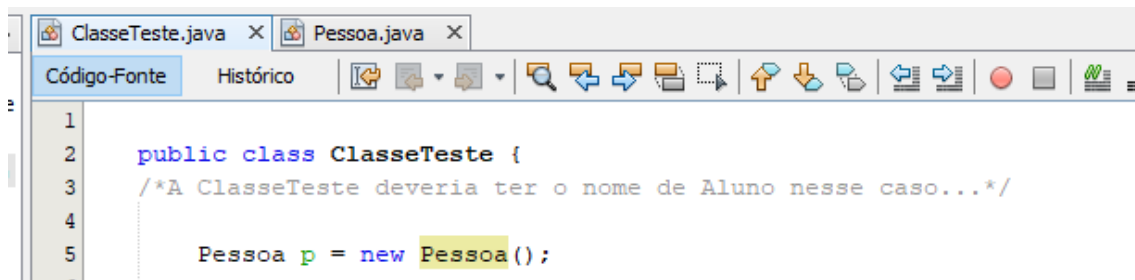


```
1
2 public class ClasseTeste {
3     /*A ClasseTeste deveria ter o nome de Aluno nesse caso...*/
4
5     public String nome;
6     public int rgm;
7     private int notal;
8
9     public double getNotal() {
10         return this.notal;
11     }
12     public void setNotal (int nl) {
13         this.notal = nl;
14     }
15 }
16
```

4) Objeto

É a instância de uma classe. Isso é, representa qualquer coisa, real ou abstrata, do mundo real, que será manipulado ou armazenado pelo sistema.

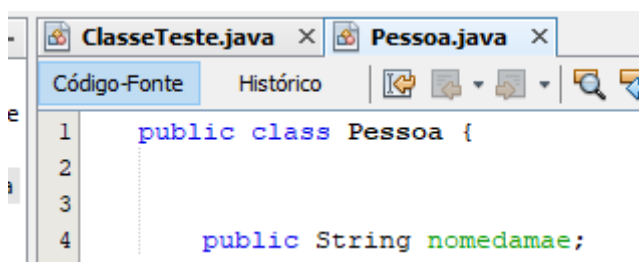
Exemplo: Classe Aluno deve instanciar a classe Pessoa



```
1
2 public class ClasseTeste {
3     /*A ClasseTeste deveria ter o nome de Aluno nesse caso...*/
4
5     Pessoa p = new Pessoa();
```

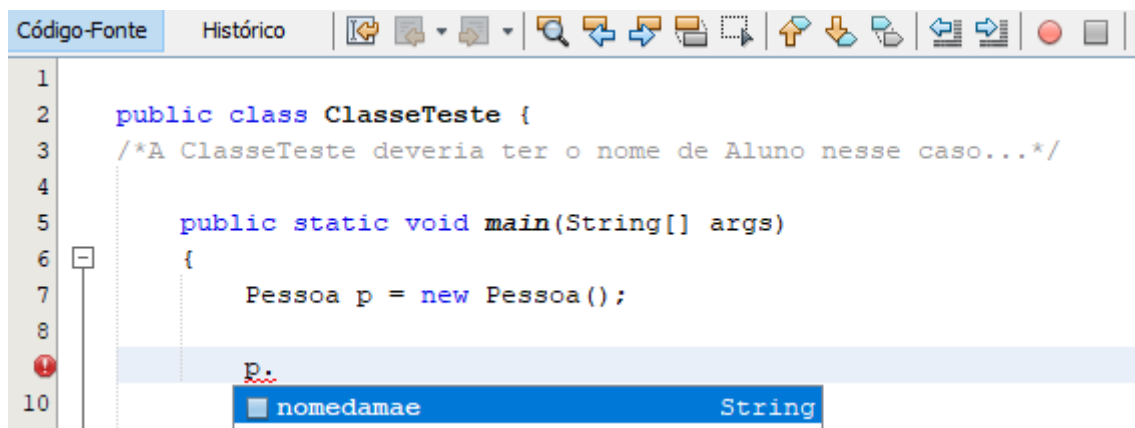
E então, agora a classe Aluno poderá acessar atributos e métodos públicos que estão na classe Pessoa, uma vez que está instanciada.

Sendo assim, ao criar um atributo, por exemplo:



```
1 public class Pessoa {
2
3
4     public String nomedamae;
```

E para acessar, podemos fazer dessa forma:



```
1
2 public class ClasseTeste {
3     /*A ClasseTeste deveria ter o nome de Aluno nesse caso...*/
4
5     public static void main(String[] args)
6     {
7         Pessoa p = new Pessoa();
8
9         p.
10        nomedamae String
```

5) Métodos

Ao pensarmos em métodos, devemos pensar na ação do objeto. Quando solicitados, devem trazer ao sistema, funções, operações ou comportamentos. Nos métodos, devemos nos atentar à visibilidade dos atributos, se há passagem ou não de parâmetros, se retorna ou não dados (void, ou apresentar o tipo do dado a ser retornado), além de detalhes para seguir a boa prática de programação, como por exemplo sempre escrever o método em letra maiúscula.

```

23
24 public int CalculaMedia(){
25     int media = ( this.notas1 + this.notas2 ) / 2;
26     return media;
27 }
28

```

Um exemplo simples de um método, elaborado para calcular a média da nota de um aluno, por exemplo.

6) Herança entre Classes

Trata-se do conceito de ter uma Superclasse, que possibilita herdar atributos e métodos às subclasses.

Uma das principais vantagens de se trabalhar com herança entre classes é diminuir a redundância de dados, evitando escrever o mesmo atributo em múltiplas classes.

Para declarar herança no código, devemos usar a palavra “extends” na declaração da classe, como na imagem:

The image shows a screenshot of a Java IDE. The top bar displays two open files: 'ClasseTeste.java' and 'Pessoa.java'. Below the bar, there are tabs for 'Código-Fonte' (Source Code) and 'Histórico' (History). The 'Código-Fonte' tab is active, showing the following code on line 1:

```
public class ClasseTeste extends Pessoa {
```

Um objeto instanciado para a ClasseTeste tem acesso aos atributos públicos da classe ClasseTeste e também da classe Pessoa.

7) Métodos Construtores

Neles, podem-se atribuir valores aos atributos ao carregar o programa. Para declará-los, existem três formas, onde os métodos devem ter o mesmo nome da classe, sendo elas:

a. Método Construtor Padrão:

Essa é a forma mais comum e mais utilizada para se declarar métodos construtores, utilizando dois construtores.

Ele é feito da seguinte forma:

```
1
2 public class ClasseTeste extends Pessoa {
3
4     public String nome;
5     public int rgm;
6     private int notal;
7     private int nota2;
8
9     public ClasseTeste() {
10         this("João", 123456);
11     }
12
13     public ClasseTeste(String nm, int rgm) {
14         this.nome = nm;
15         this.rgm = rgm;
16     }
17
18 }
```

No primeiro momento, declaramos os atributos. E então, começa-se o primeiro construtor, já atribuindo os valores na sequência dos atributos desejados. Então, no segundo construtor, declaramos quais serão os atributos a receber os valores.

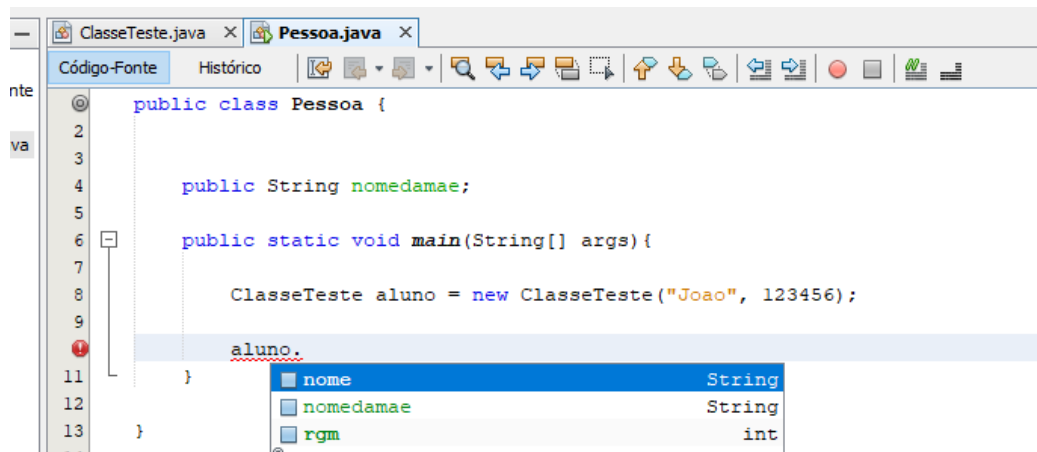
b. Método – passando os valores pela declaração do objeto:

Dessa forma, ao criar o objeto na classe, dentro dos próprios parâmetros do objeto, já se declaram os valores a serem atribuídos.

Aqui, criamos o primeiro e único construtor, chamando o método com o nome e declarando os atributos a receberem os valores:

```
1
2 public class ClasseTeste extends Pessoa {
3
4     public String nome;
5     public int rgm;
6
7     public ClasseTeste(String nm, int rgm) {
8         this.nome = nm;
9         this.rgm = rgm;
10     }
11
12 }
```

E então, ao declarar o objeto a ser instanciado, já atribui-se o valor, como no print abaixo:



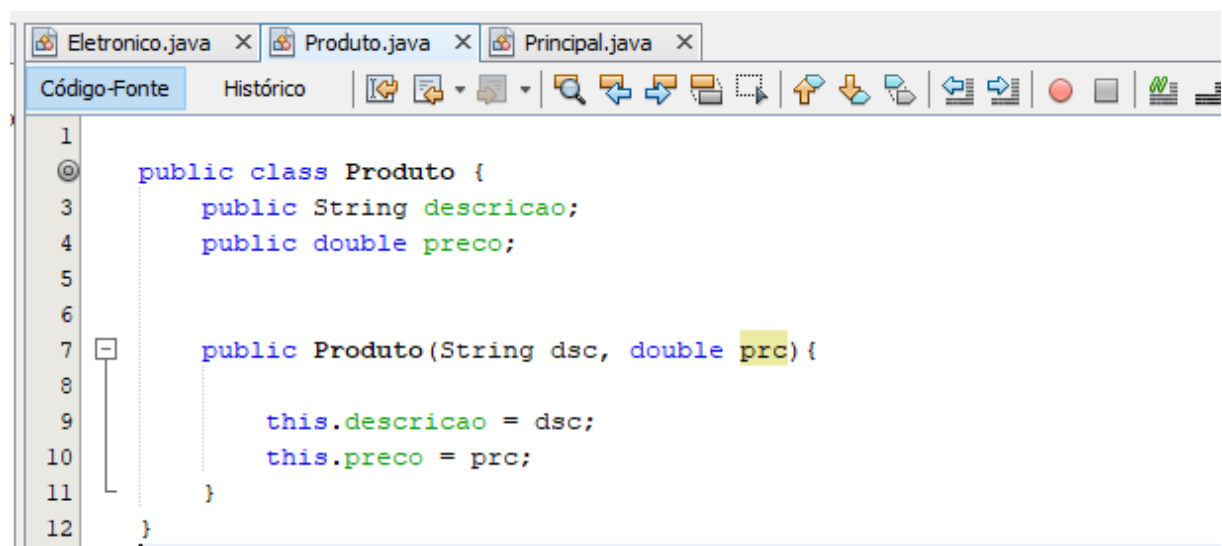
```
public class Pessoa {  
    public String nomedamae;  
  
    public static void main(String[] args) {  
        ClasseTeste aluno = new ClasseTeste("Joao", 123456);  
        aluno.  
    }  
}
```

nome	String
nomedamae	String
rgm	int

c. Método – Iniciando atributos por sub e superclasses:

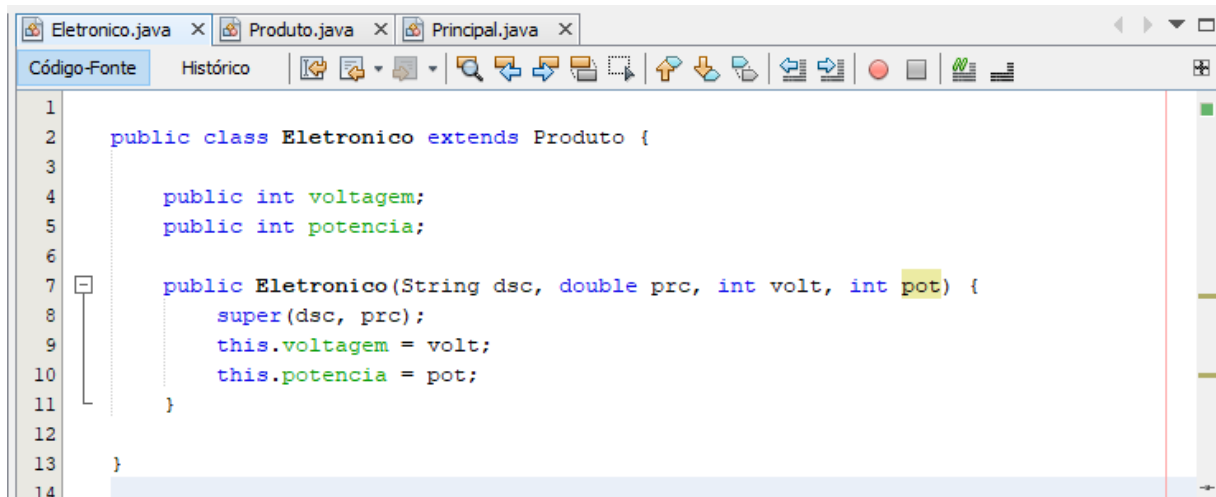
Dessa forma, utiliza-se o conceito de herança para auxiliar no método construtor, sendo assim feita a atribuição de valores aos atributos.

Aqui está a superclasse. Nela, estão os principais atributos – que serão referenciados na subclasse. É onde se colocará o primeiro método construtor:



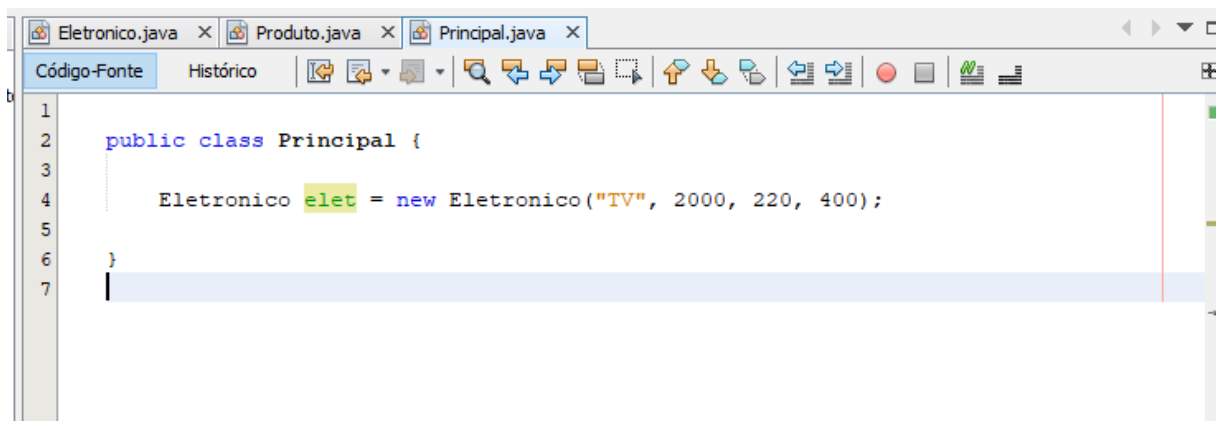
```
public class Produto {  
    public String descricao;  
    public double preco;  
  
    public Produto(String dsc, double prc) {  
        this.descricao = dsc;  
        this.preco = prc;  
    }  
}
```

E então, a subclasse, herdando os atributos da superclasse, e ainda complementando o primeiro construtor com os atributos da sua própria classe (Eletronico):



```
1
2 public class Elettronico extends Produto {
3
4     public int voltagem;
5     public int potencia;
6
7     public Elettronico(String dsc, double prc, int volt, int pot) {
8         super(dsc, prc);
9         this.voltagem = volt;
10        this.potencia = pot;
11    }
12
13 }
14
```

Aqui temos a classe Principal, onde está instanciado o objeto Elettronico, e nele, enfim são atribuídos os valores desejados para o início do programa:

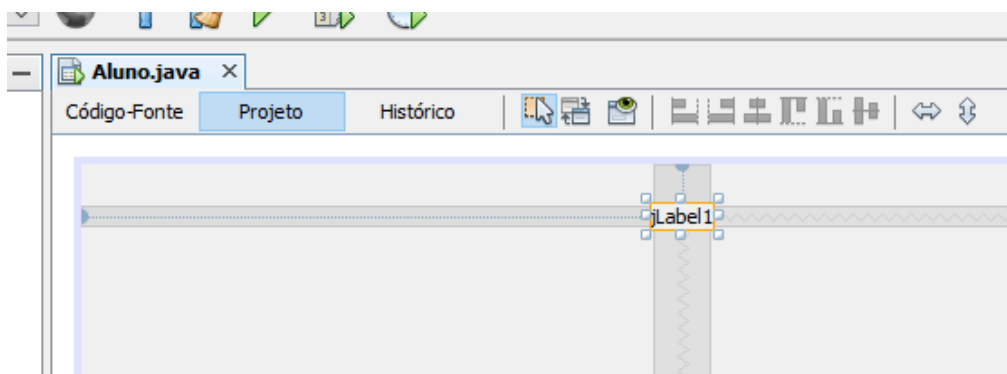


```
1
2 public class Principal {
3
4     Elettronico elet = new Elettronico("TV", 2000, 220, 400);
5
6 }
7
```

8) Interfaces

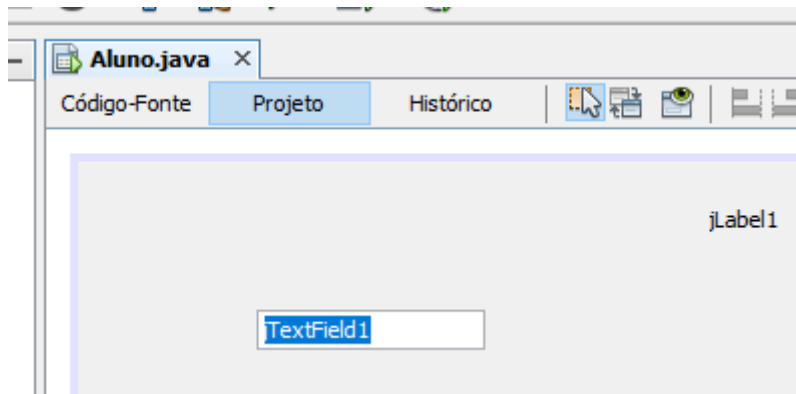
Interfaces servem apenas para receber e enviar dados. Nelas, não devem ser armazenadas operações, apenas métodos. Nelas, existem diversas funcionalidades a serem utilizadas, por exemplo:

- Label: Aqui podemos adicionar algum texto ou descrição à interface.



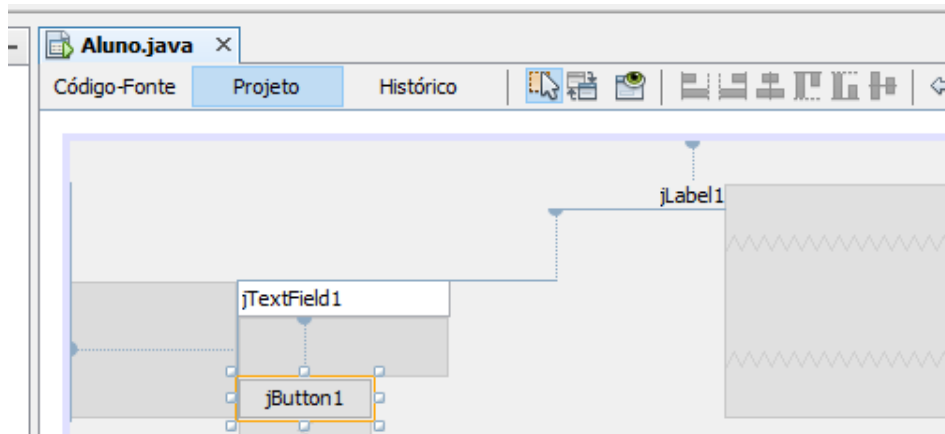
- TextField: Com essa seleção, podemos criar um campo de texto. Pode servir para inserir dados ou exibir dados/resultados.

Na edição de variáveis, seguimos o padrão utilizando “txt” + identificação do campo. Exemplo: Campo de texto para cadastrar a primeira nota do bimestre de um aluno – txtnota1.



- Buttons: Neles são armazenados MÉTODOS, utilizados para operar e manipular os dados inseridos na interface.

Na edição de variáveis, deve-se seguir o padrão: btn + nome do método (utilizando inicial maiúscula para o nome do método. (“btnNomeDoMetodo”))



Ao utilizar a interface, devemos nos atentar para renomear os campos e, também as variáveis. Assim, é possível manipular de forma mais prática as funcionalidades da interface pelo código-fonte.