

Guia Programação JAVA



Tópicos

- Construtores
- Método recursivo
- Conversão de tipos
- Especificadores de acesso
- Encapsulamento
- Polimorfismo
- Herança e Interface
- Sobreposição e sobrecarga

Tópicos

- Thread
- Java.lang
- Java.util
- Tipos brutos e código legado
- Expressões lambda e referências de método
- Trabalhando com datas (Calendar)
- Expressões regulares
- Anotações

Fóruns

Fórum - Stackoverflow (Português)

Devmedia (Inglês)

JavaProgressivo (Inglês)

JavaFree (Inglês)

Fórum - GUJ (Português)

Teclas de atalho

Ctrl + Shift + i -> importação automática

Ctrl + espaço -> auto preenchimento

Ctrl + Shift + seta para baixo -> cópia da linha

Ctrl + Shift + c -> comentar linha

1. Construtor

- É um método especial, que é chamado automaticamente assim que instancia o objeto.
- Para criar um Construtor, usa-se a tecla de atalho -
> Alt + Insert -> Construtor

1. Construtor

- Outra maneira para criar o Construtor é digitando o código:

- ```
public nome_da_classe () {

}
```

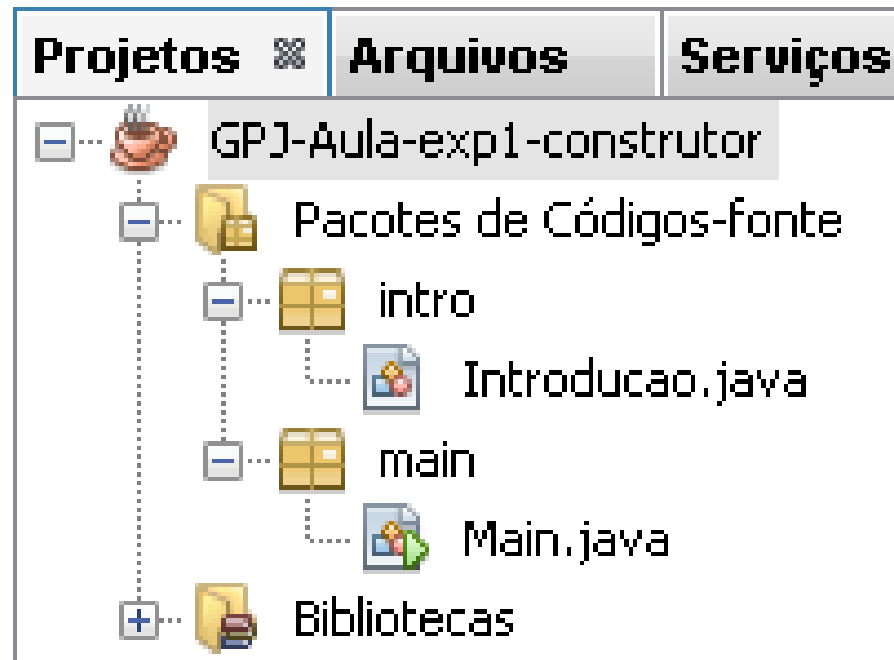
- Pode ser chamado sem Instanciar o objeto.

- Exemplo:

```
new nome_da_classe ();
```

# 1. Construtor

- GPJ – Aula 1 – Ex 1 – Exemplo Construtor
- Ver diferença entre método normal X construtor.
- Criar pacote **intro** e classe **Introducao** e Criar pacote **main** e classe **Main**.

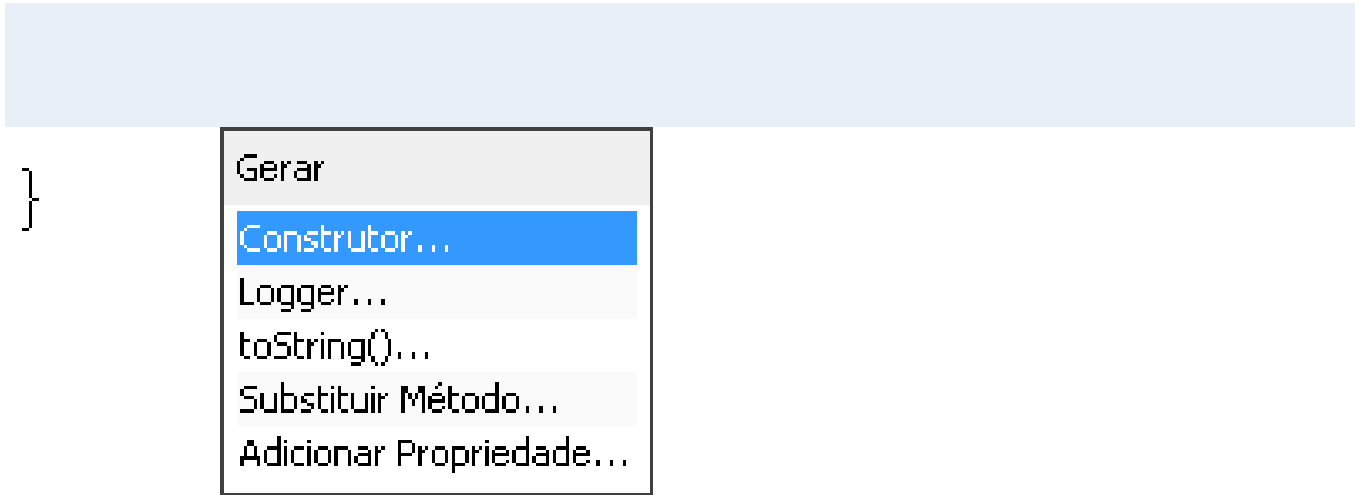




# 1. Construtor

- Na classe **Introducao** criar o construtor com Alt+Insert

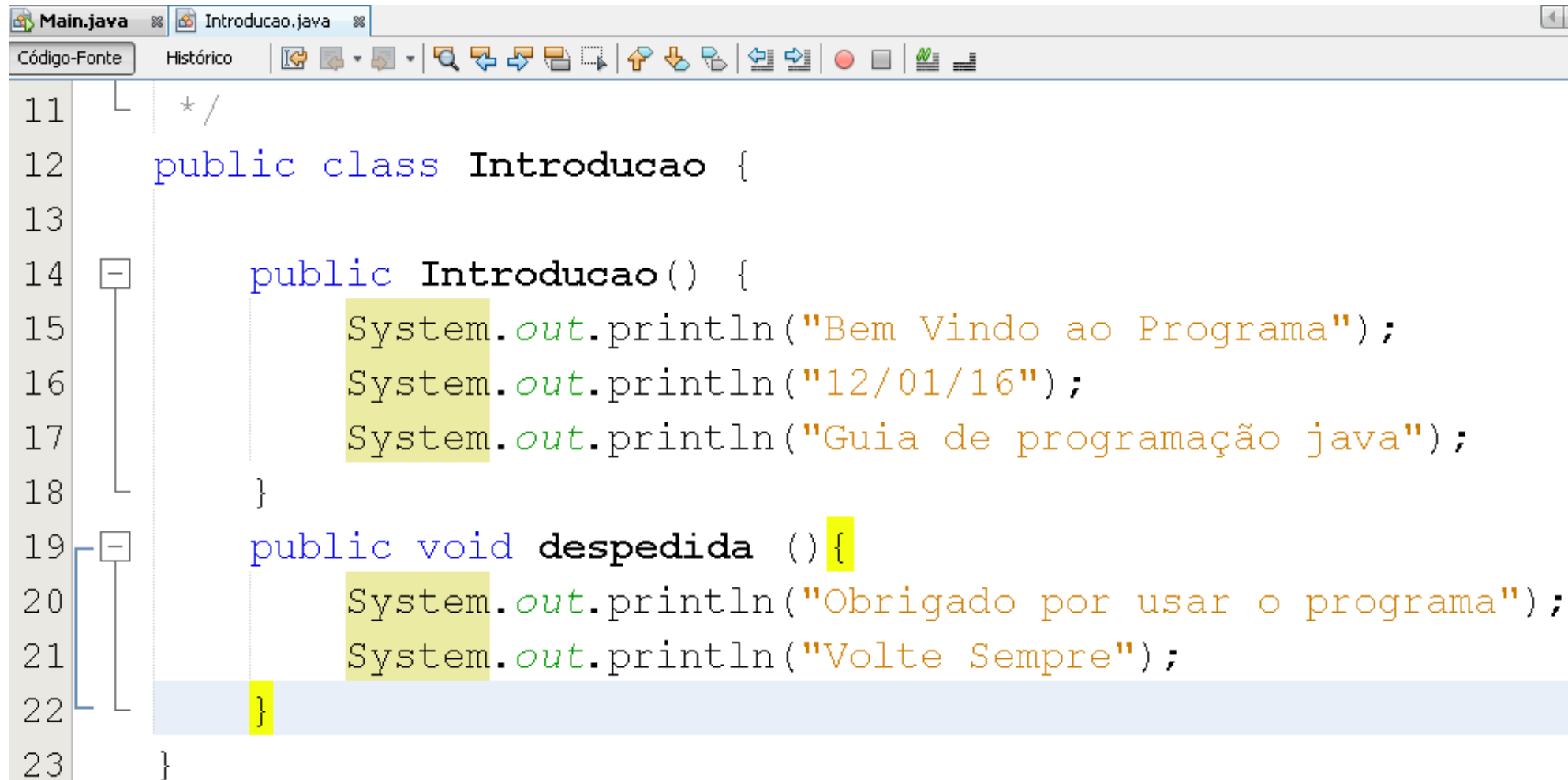
```
12 public class Introducao {
13
14
15 }
16
```



Gerar  
Construtor...  
Logger...  
toString()...  
Substituir Método...  
Adicionar Propriedade...

# 1. Construtor

- Após criar o construtor, criar o método despedida.

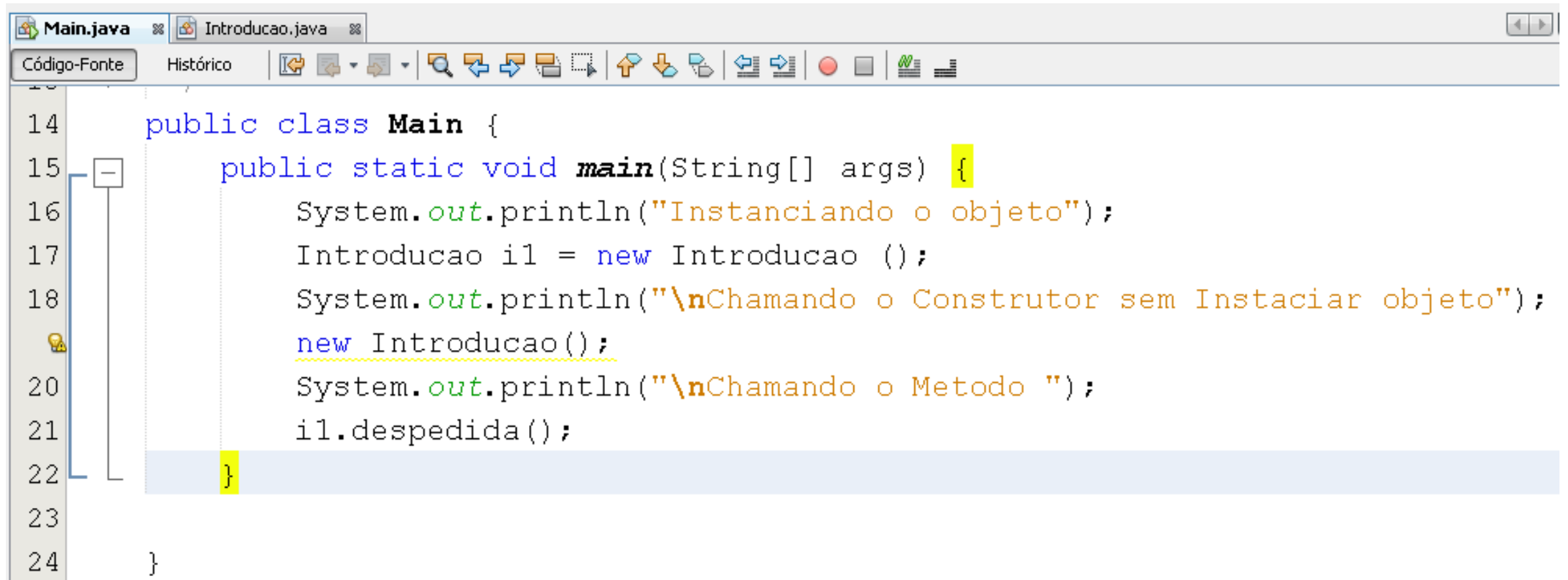


The screenshot shows an IDE window with two tabs: 'Main.java' and 'Introducao.java'. The 'Introducao.java' tab is active, displaying the source code of a Java class. The code defines a public class 'Introducao' with two methods: a constructor 'Introducao()' and a method 'despedida()'. The constructor prints three lines of text: 'Bem Vindo ao Programa', '12/01/16', and 'Guia de programação java'. The 'despedida()' method prints two lines: 'Obrigado por usar o programa' and 'Volte Sempre'. The code is color-coded, with keywords in blue, strings in orange, and the 'System.out.println' calls in green. The line numbers 11 through 23 are visible on the left side of the editor.

```
11 */
12 public class Introducao {
13
14 public Introducao() {
15 System.out.println("Bem Vindo ao Programa");
16 System.out.println("12/01/16");
17 System.out.println("Guia de programação java");
18 }
19
20 public void despedida () {
21 System.out.println("Obrigado por usar o programa");
22 System.out.println("Volte Sempre");
23 }
24 }
```

# 1. Construtor

- Na classe **Main** instanciar o objeto e também chamar o método sem instanciar.



The screenshot shows an IDE window with two tabs: 'Main.java' and 'Introducao.java'. The 'Código-Fonte' (Source Code) tab is active. The code in 'Main.java' is as follows:

```
14 public class Main {
15 public static void main(String[] args) {
16 System.out.println("Instanciando o objeto");
17 Introducao i1 = new Introducao ();
18 System.out.println("\nChamando o Construtor sem Instaciar objeto");
19 new Introducao();
20 System.out.println("\nChamando o Metodo ");
21 i1.despedida();
22 }
23 }
24 }
```

The code demonstrates the creation of a `Main` class with a `main` method. Inside the `main` method, it prints a message, creates an instance of the `Introducao` class, prints another message, calls the constructor of `Introducao` without creating a new variable (highlighted with a yellow background), prints a third message, and calls the `despedida` method on the instance `i1`.

# 1. Construtor

- Teoricamente não deve aparecer nada.

Saída - GPJ-Aula-exp1-construtor (run)



run:

Instanciando o objeto

Bem Vindo ao Programa

12/01/16

Guia de programação java

Chamando o Construtor sem Instaciar objeto

Bem Vindo ao Programa

12/01/16

Guia de programação java

Chamando o Metodo

Obrigado por usar o programa

Volte Sempre

CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)

Note que o Método Despedida só aparece após chamá-lo. O mesmo não acontece com o construtor.

## 2. Herança

### O que é Herança?

- Todos os atributos e métodos são herdados da classe pai pelas classes filhas.
- Em Java, cada classe filha só pode ter uma classe pai.

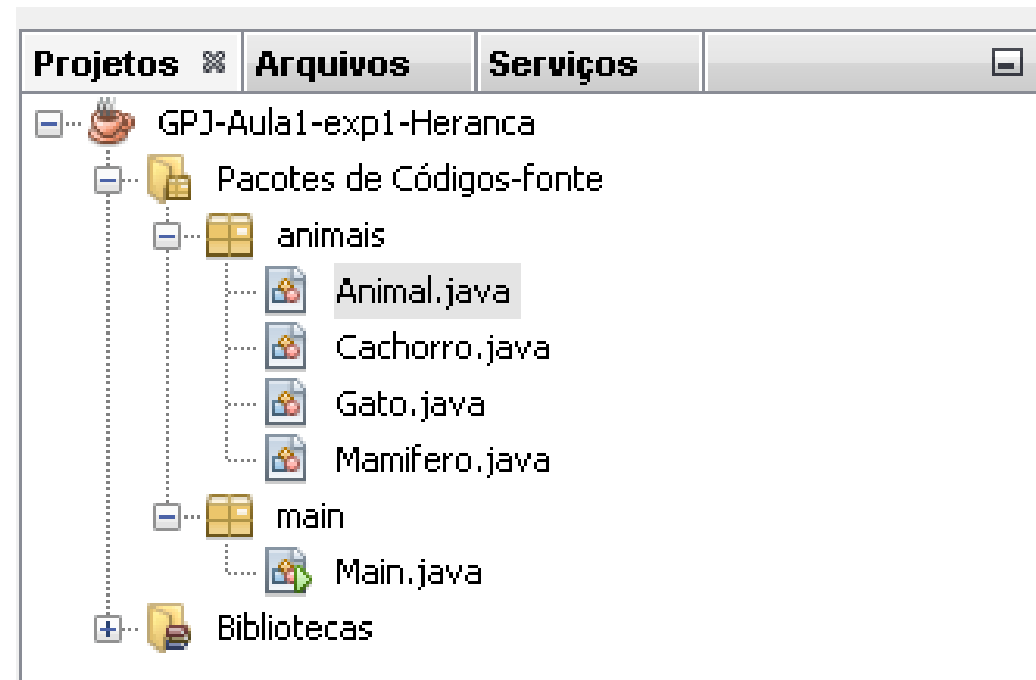
## 2. Herança

- A palavra chave de Herança é *extends*
- A vantagem de usar Herança é que a classe filha pode ter seus próprios métodos além daqueles herdados pela classe pai.

Ex.: `public class Classe_filha extends Classe_pai {`

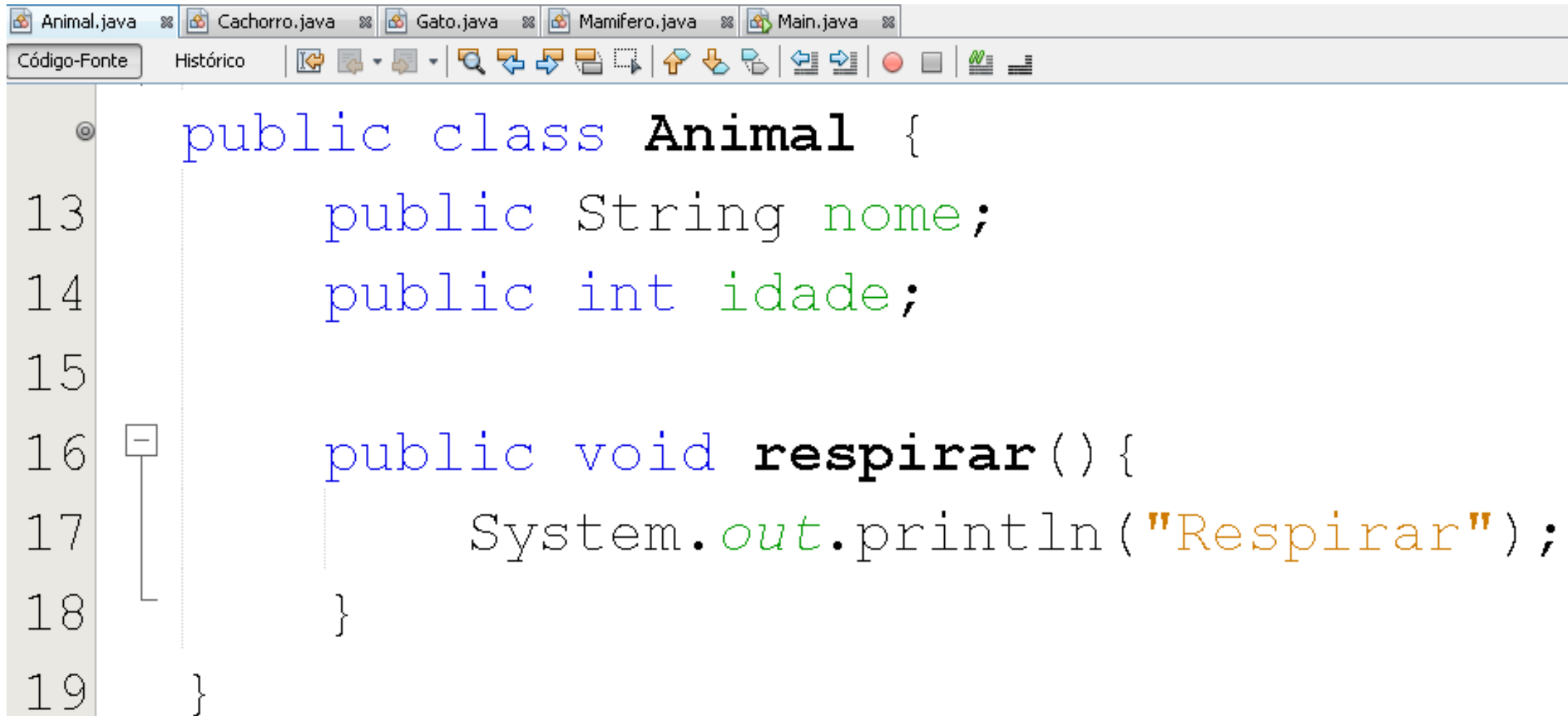
## 2. Herança

- GPJ – Aula 1 – Ex 2 – Exemplo Herança
- Ver exemplo de Herança.
- Criar pacote **animais** e as classes **Animal**, **Mamifero**, **Cachorro** e **Gato**.
- Criar pacote **main** e classe **Main**.



## 2. Herança

- Na classe **Animal** criar os Atributos e o Método



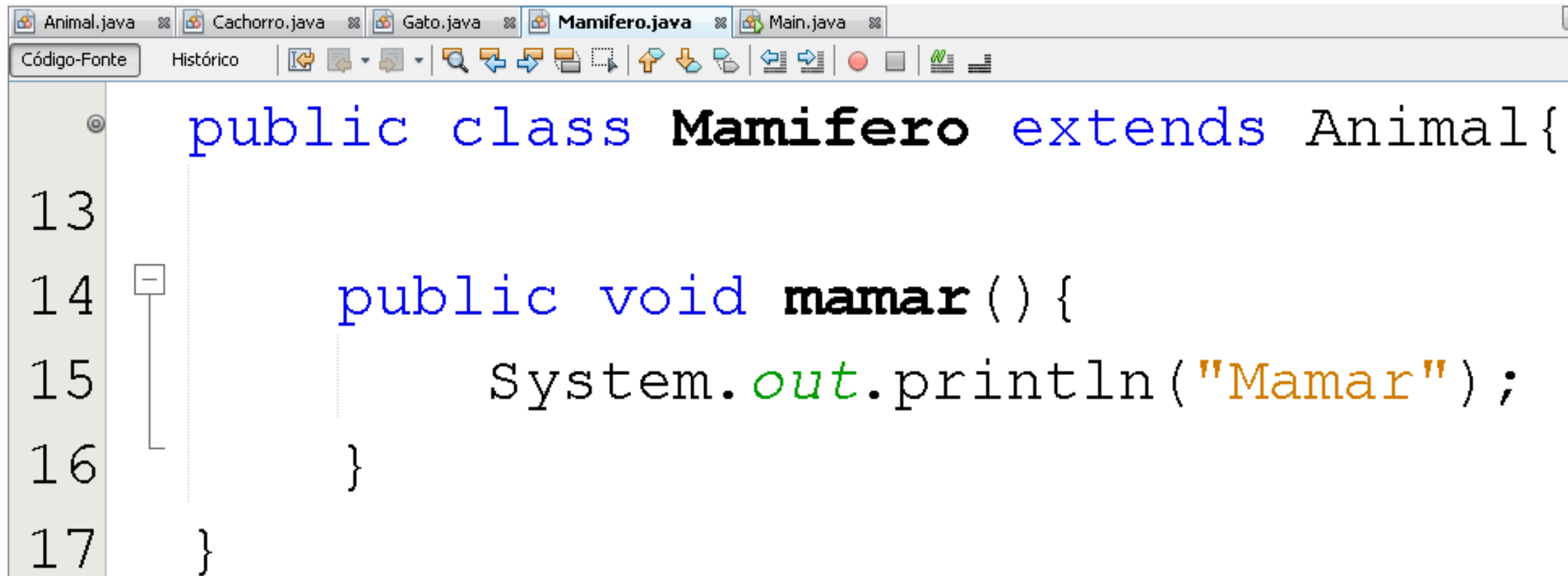
```
Animal.java Cachorro.java Gato.java Mamifero.java Main.java
Código-Fonte Histórico
public class Animal {
 public String nome;
 public int idade;

 public void respirar() {
 System.out.println("Respirar");
 }
}
```



## 2. Herança

- Na classe **Mamifero** colocar como filha da classe **Animal** e criar o Método **mamar**.



```
Animal.java Cachorro.java Gato.java Mamifero.java Main.java
Código-Fonte Histórico
public class Mamifero extends Animal{
13
14 public void mamar() {
15 System.out.println("Mamar");
16 }
17 }
```

## 2. Herança

- Na classe **Gato** e **Cachorro** colocar como filhas da classe **Mamiferos**.

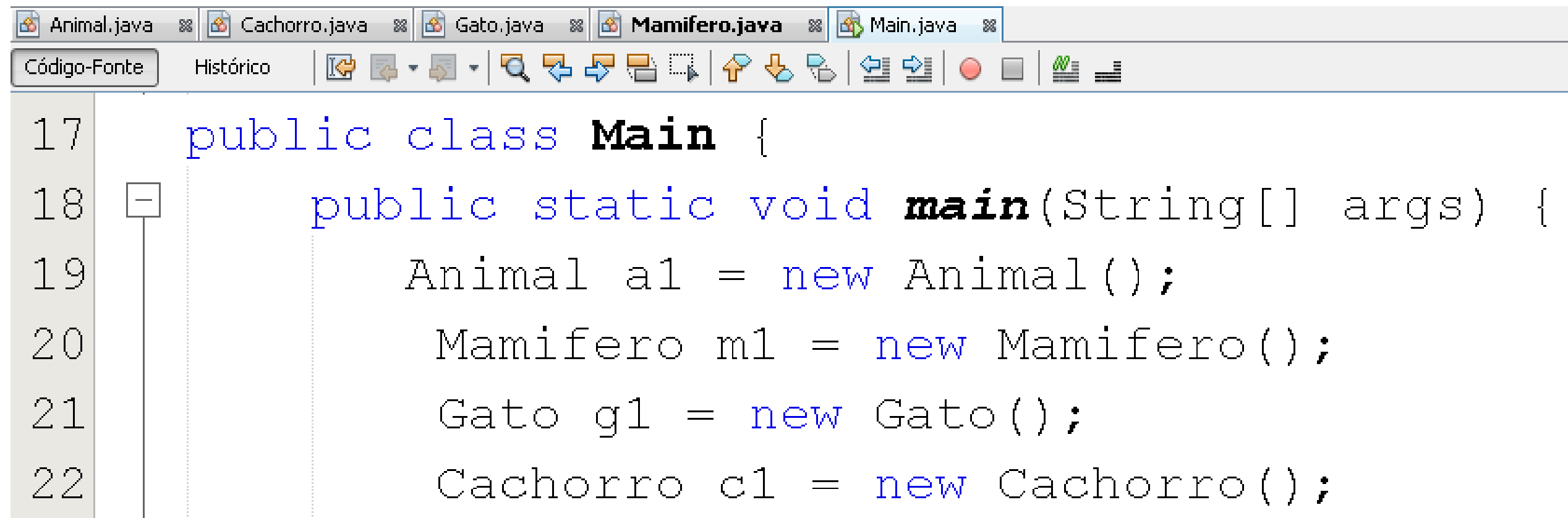
Exemplo:

```
public class Gato extends Mamifero{
```

```
public class Cachorro extends Mamifero{
```

## 2. Herança

- Na classe **Main** instanciar os objetos.

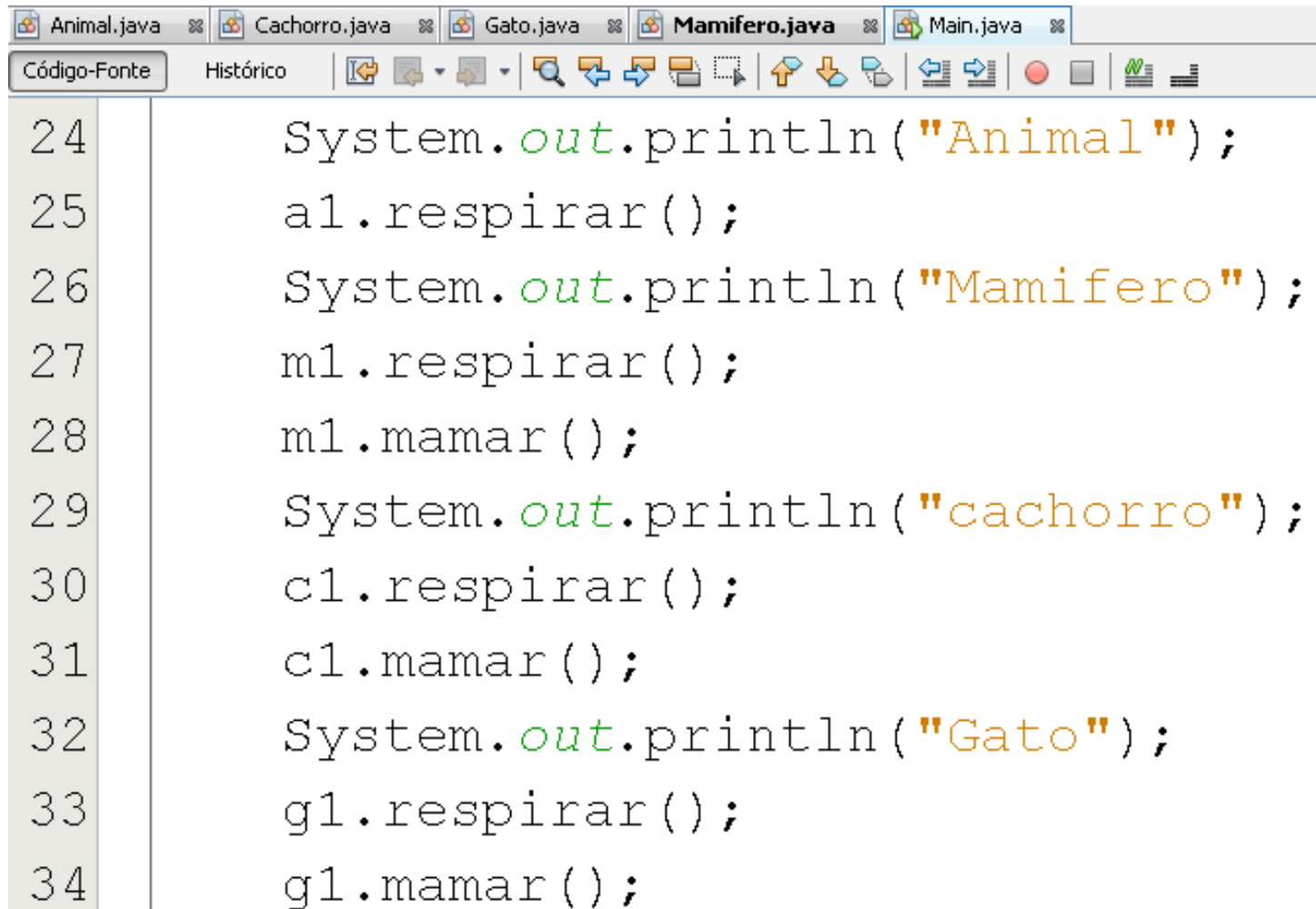


The screenshot shows an IDE window with several tabs: Animal.java, Cachorro.java, Gato.java, Mamifero.java, and Main.java. The 'Main.java' tab is active. The code editor displays the following Java code:

```
17 public class Main {
18 public static void main(String[] args) {
19 Animal a1 = new Animal();
20 Mamifero m1 = new Mamifero();
21 Gato g1 = new Gato();
22 Cachorro c1 = new Cachorro();
 }
```

## 2. Herança

- Chamar os Métodos das Classes.



```
Animal.java Cachorro.java Gato.java Mamifero.java Main.java
Código-Fonte Histórico
24 System.out.println("Animal");
25 a1.respirar();
26 System.out.println("Mamifero");
27 m1.respirar();
28 m1.mamar();
29 System.out.println("cachorro");
30 c1.respirar();
31 c1.mamar();
32 System.out.println("Gato");
33 g1.respirar();
34 g1.mamar();
```

## 2. Herança

- Veja como as classes herdaram os métodos.

:: Saída - GPJ-Aula1-exp1-Heranca (run)



run:



Animal

Respirar



Mamifero



Respirar

Mamar

cachorro

Respirar

Mamar

Gato

Respirar

Mamar

CONSTRUÍDO COM SUCESSO (tempo total: 3 segundos)

## 2. Herança

**Exercício:** Criar um pacote veículos com as classe pai **Veiculos**, as classes filhas **Terrestre**, **Aquatico** e **Aereo** e outro pacote **main** com Classe **Main**. Na classe **Veiculos** criar os atributos **marca**, **cor** e **preco**. Na classe **Terrestre** criar o atributo **rodas** e Método **correr**, na **Aquatico** atributo **peso** e método **nadar** e na **Aereo** atributo **material** e método **voar**.

### 3. Sobreposição de Método:

- É quando a classe filha herda o método da classe pai, mas muda o conteúdo do método. É o famoso filho rebelde, que não segue os caminhos do pai.
- Sobreposição de método só é usado em herança.

### 3. Sobreposição de Método

- O método sobreposto deve conter o mesmo tipo, nome, parâmetros e argumentos do método herdado, apenas o conteúdo muda.

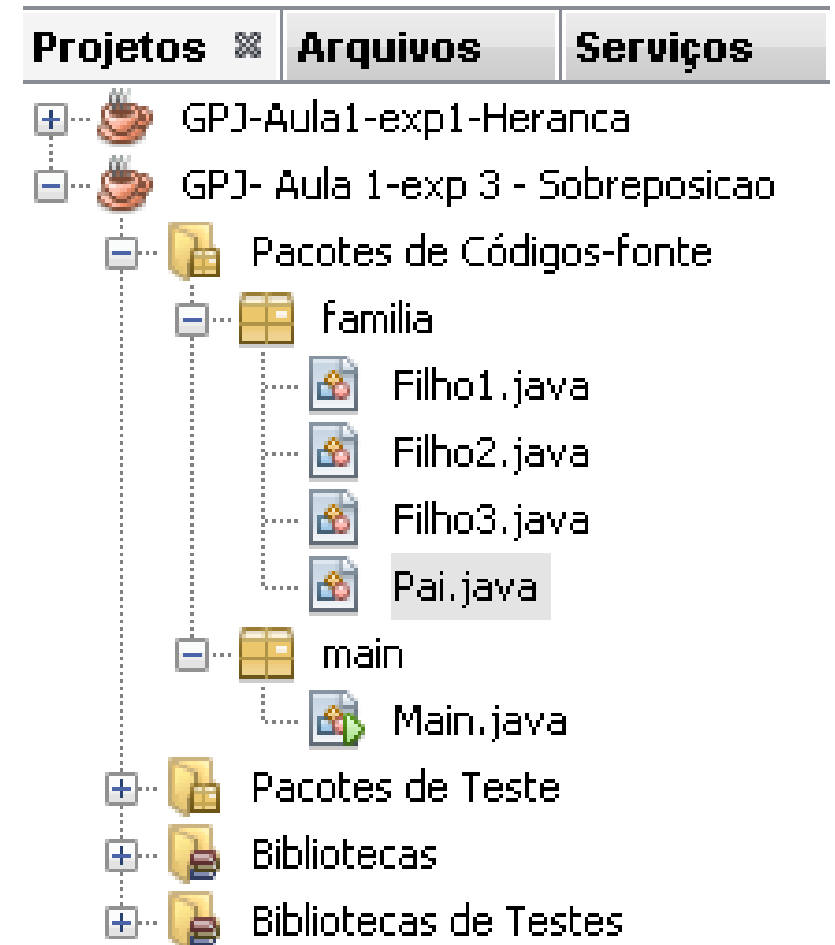
Métodos sobrescritos possuem a anotação:

@Override



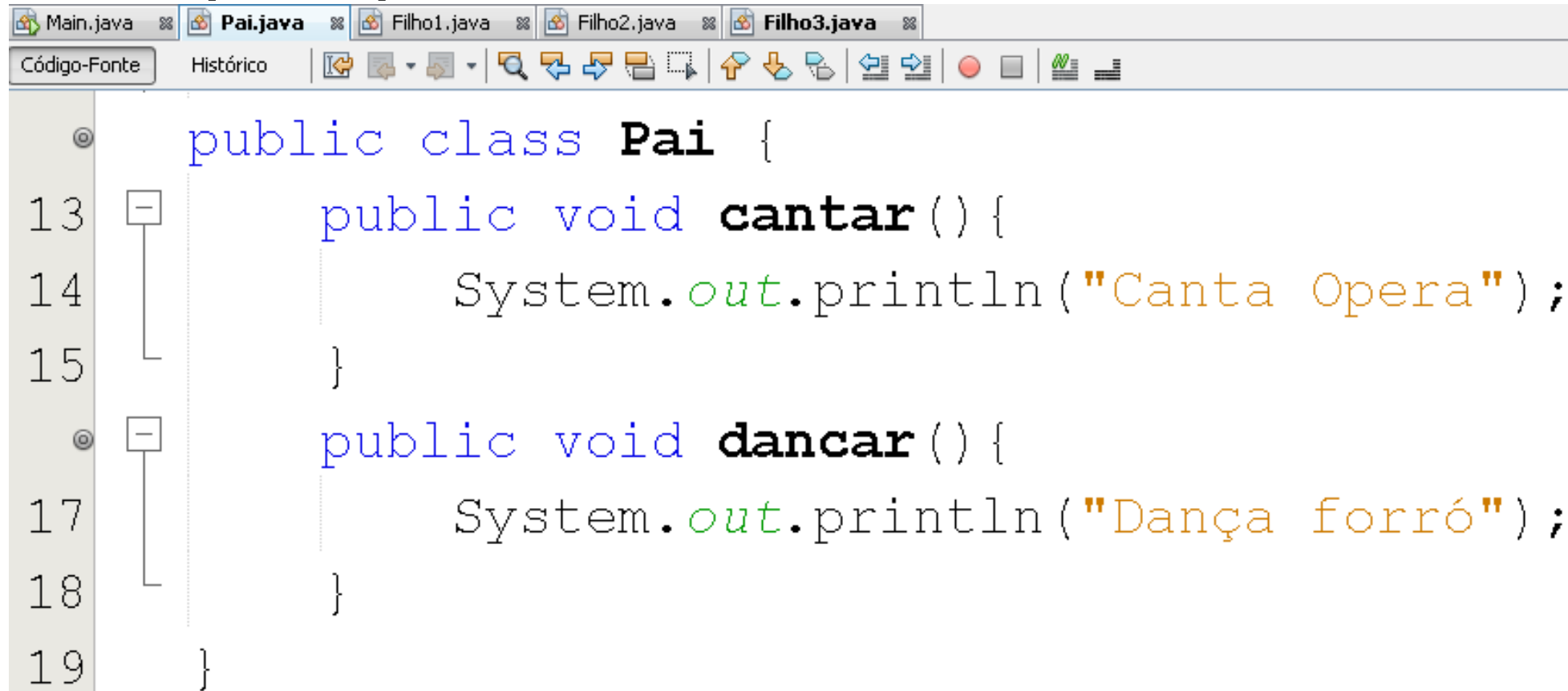
### 3. Sobreposição de Método

- GPJ – Aula 1 – Ex 3 – Exemplo Sobreposição de Método
- Criar o pacote **main** com a classe **Main** e o pacote **família** com as classes **Pai**, **Filho1**, **Filho2** e **Filho3**.



### 3. Sobreposição de Método

- Na classe **Pai** criar os métodos **cantar(operas)** e **dancar(forró)**.

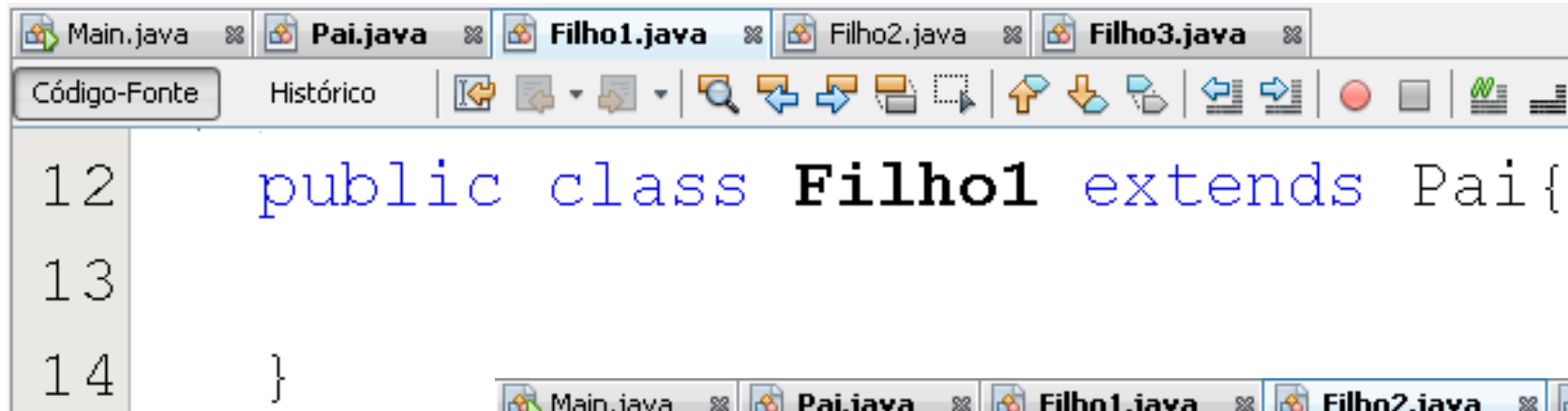


The screenshot shows a Java IDE with several tabs: Main.java, Pai.java (selected), Filho1.java, Filho2.java, and Filho3.java. The code in the Pai.java tab is as follows:

```
public class Pai {
 13 public void cantar() {
 14 System.out.println("Canta Opera");
 15 }
 16 public void dancar() {
 17 System.out.println("Dança forró");
 18 }
 19 }
```

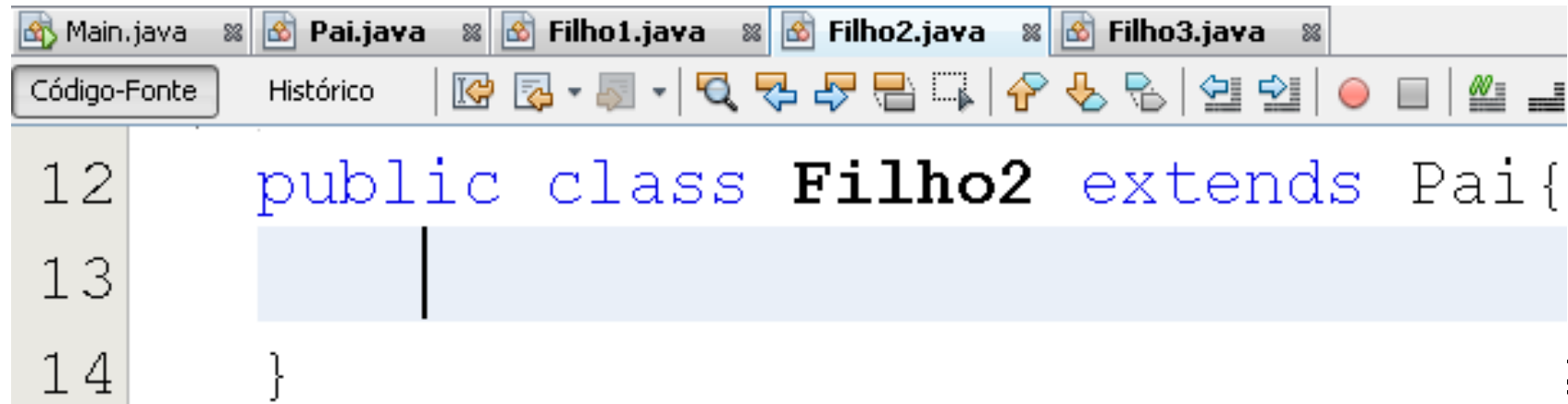
### 3. Sobreposição de Método

- Deixar as classes **Filho1** e **Filho2** como herdeiras da classe **Pai** (*extends*).



The screenshot shows an IDE window with several tabs: Main.java, Pai.java, Filho1.java (active), Filho2.java, and Filho3.java. The active tab displays the following code:

```
12 public class Filho1 extends Pai{
13
14 }
```



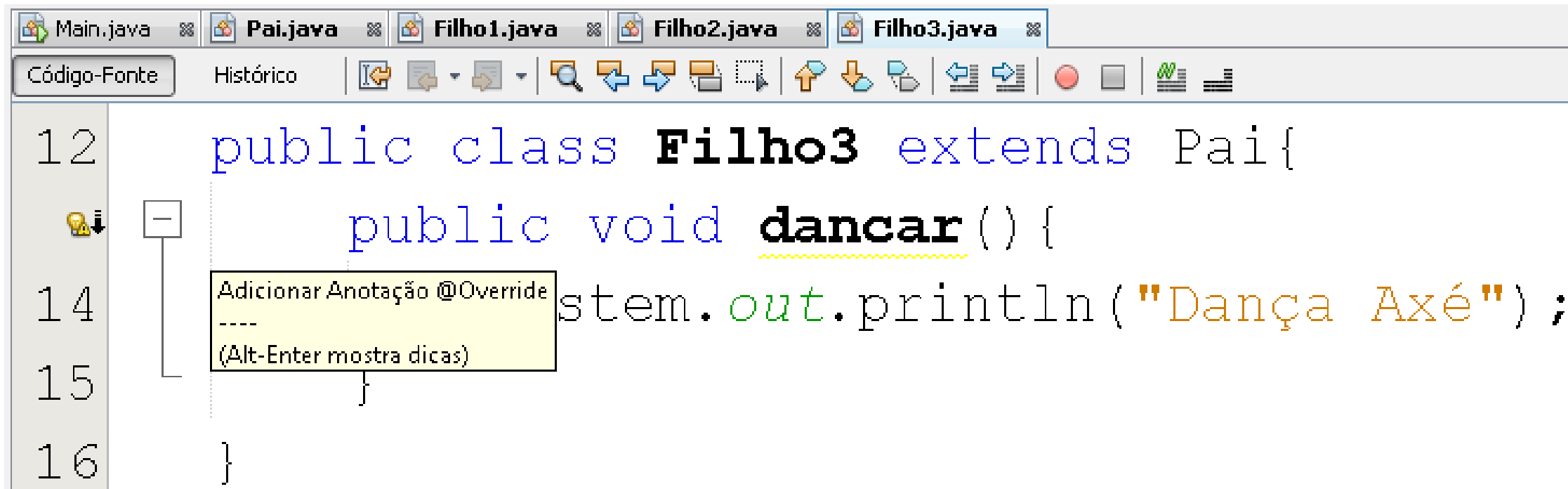
The screenshot shows an IDE window with several tabs: Main.java, Pai.java, Filho1.java, Filho2.java (active), and Filho3.java. The active tab displays the following code:

```
12 public class Filho2 extends Pai{
13 |
14 }
```

### 3. Sobreposição de Método

- Colocar a classe **Filho3** como herdeira da classe **Pai**, criar o método **dancar** (igual ao pai) e alterar para **Axé**.

Repare que do lado esquerdo há uma lâmpada amarela.



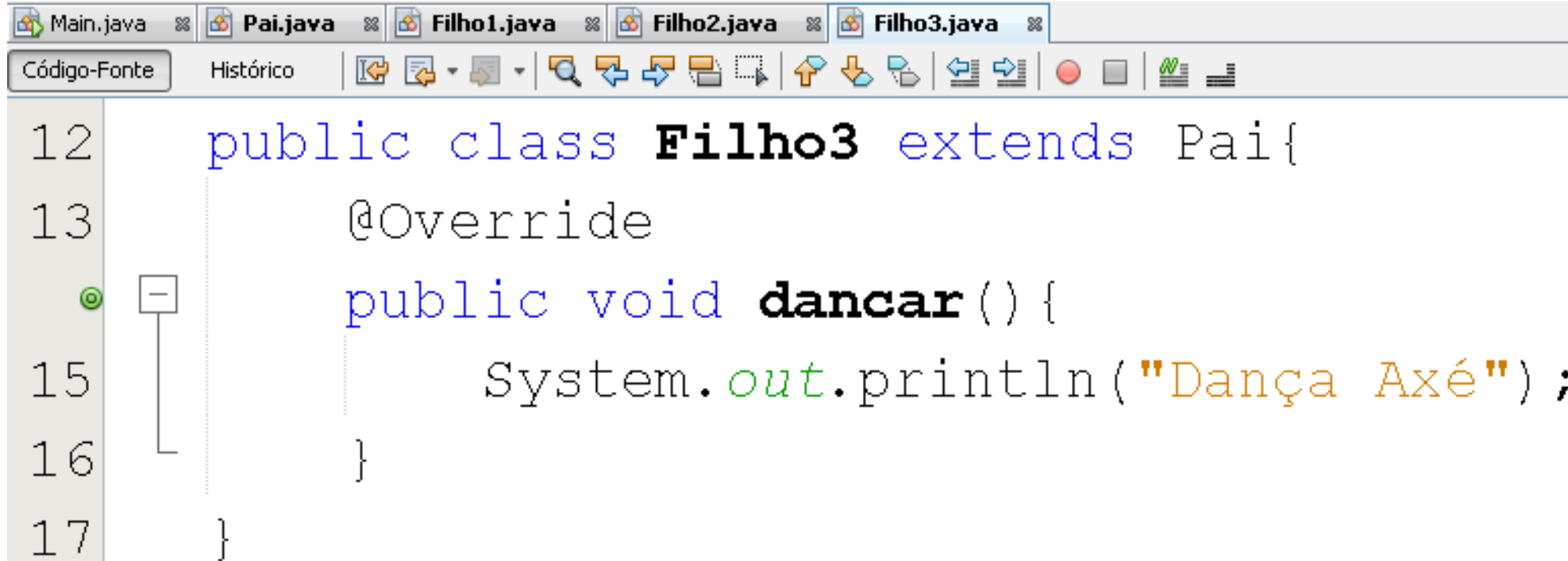
```
12 public class Filho3 extends Pai{
13 public void dancar() {
14 System.out.println("Dança Axé");
15 }
16 }
```

Adicionar Anotação @Override  
----  
(Alt-Enter mostra dicas)

### 3. Sobreposição de Método

- Clicar na lâmpada amarela para ativar a sobreposição.

Note que a palavra `@Override` irá aparecer e a lâmpada sumirá.



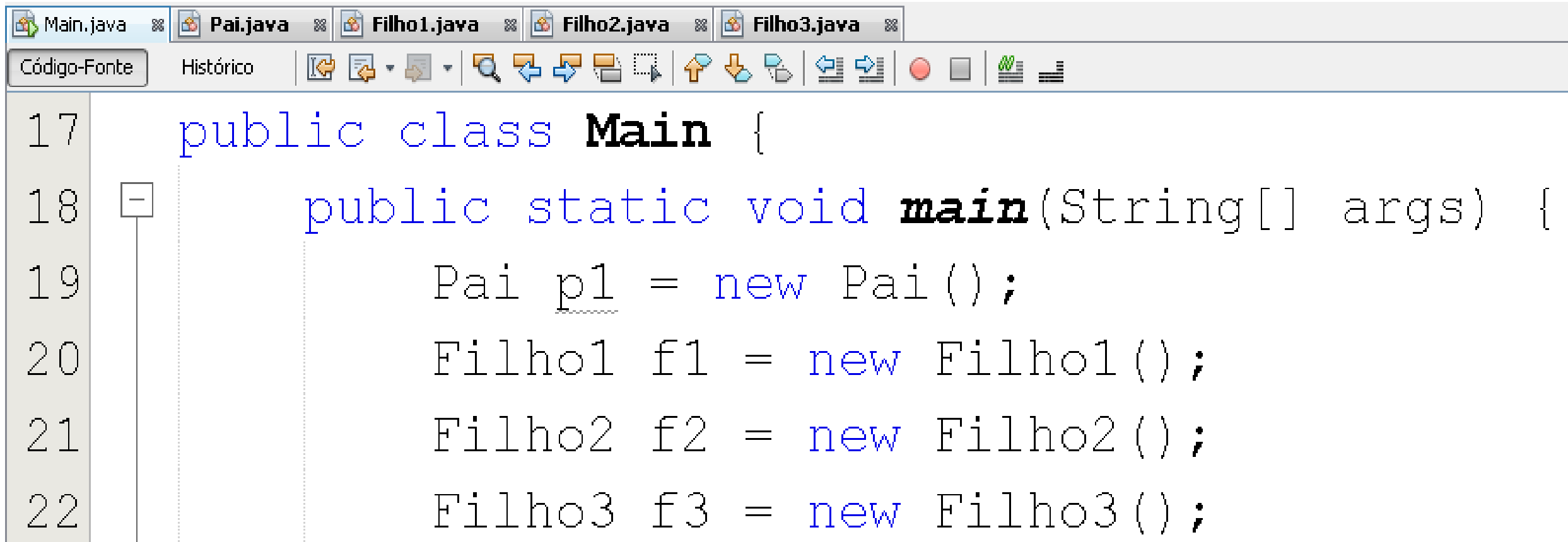
The screenshot shows an IDE window with several tabs: Main.java, Pai.java, Filho1.java, Filho2.java, and Filho3.java. The 'Filho3.java' tab is active. The code editor displays the following code:

```
12 public class Filho3 extends Pai{
13 @Override
14 public void dancar() {
15 System.out.println("Dança Axé");
16 }
17 }
```

A green circle icon is visible next to line 14, indicating that the IDE has recognized the method as an override.

### 3. Sobreposição de Método

- Na classe **Main** instanciar os objetos.

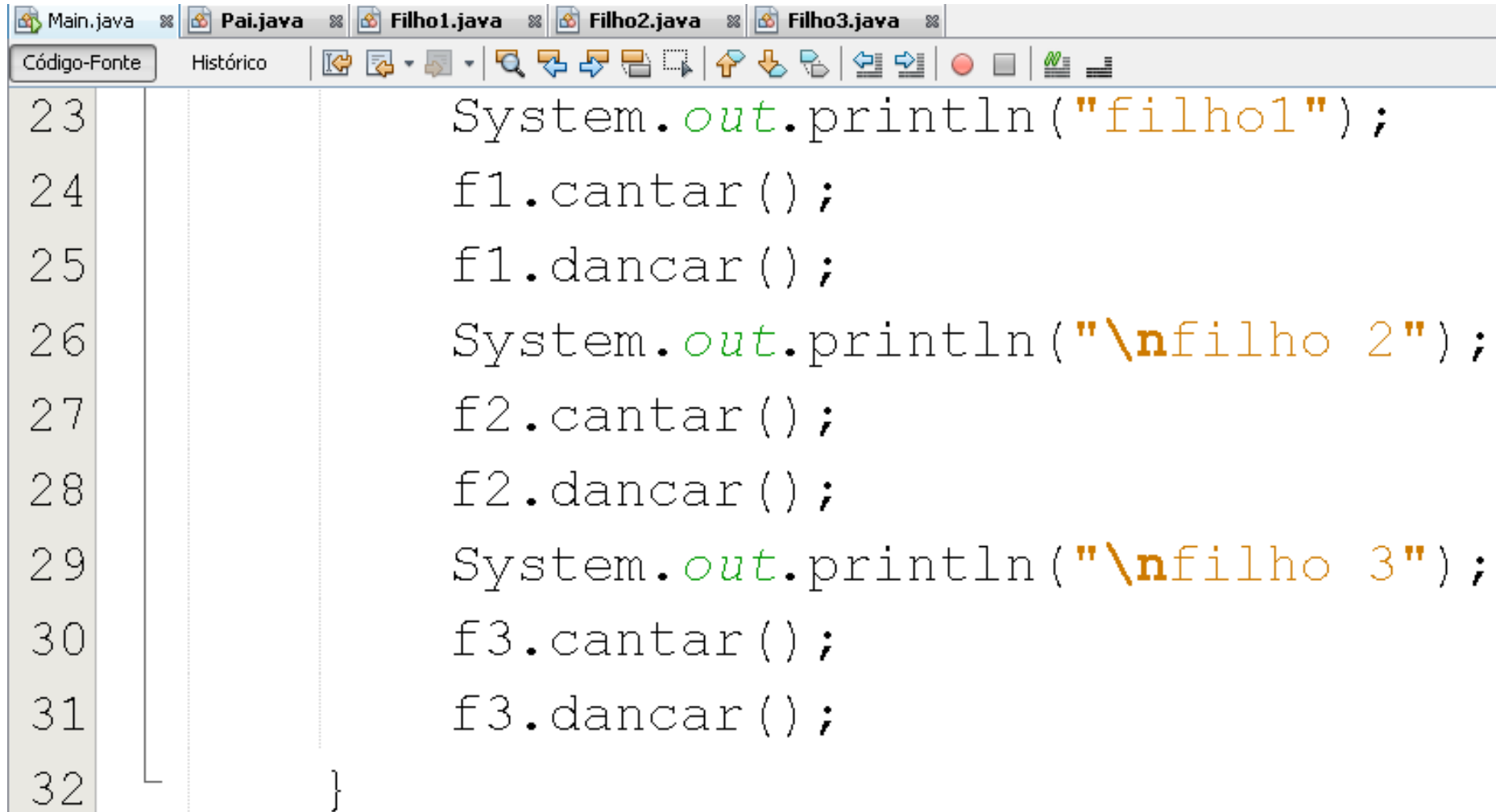
A screenshot of a Java IDE window. The title bar shows five open files: Main.java, Pai.java, Filho1.java, Filho2.java, and Filho3.java. The 'Código-Fonte' (Source Code) tab is active. The code editor displays the following Java code:

```
17 public class Main {
18 public static void main(String[] args) {
19 Pai p1 = new Pai();
20 Filho1 f1 = new Filho1();
21 Filho2 f2 = new Filho2();
22 Filho3 f3 = new Filho3();
```

The code is color-coded: keywords like 'public', 'class', 'static', 'void', 'new', and 'String' are in blue; the class name 'Main' is in bold black; the method name 'main' is in bold italic black; and variable names and literals are in black. The IDE interface includes a toolbar with various icons for navigation and editing.

# 3. Sobreposição de Método

- Chamar os Métodos das Classes.

A screenshot of a Java IDE window. The title bar shows five open files: Main.java, Pai.java, Filho1.java, Filho2.java, and Filho3.java. The 'Código-Fonte' (Source Code) tab is active. The code editor displays the following Java code:

```
23 System.out.println("filho1");
24 f1.cantar();
25 f1.dancar();
26 System.out.println("\nfilho 2");
27 f2.cantar();
28 f2.dancar();
29 System.out.println("\nfilho 3");
30 f3.cantar();
31 f3.dancar();
32 }
```

### 3. Sobreposição de Método

- Veja como as classes herdaram os métodos e a **filho3** mudou o método **dança**.

⋮ Saída - GPJ- Aula 1-exp 3 - Sobreposicao (run)



run:



filhol

Canta Opera



Dança forró



filho 2

Canta Opera

Dança forró

filho 3

Canta Opera

Dança Axé

CONSTRUÍDO COM SUCESSO (tempo total: 3 segundos)



### 3. Sobreposição de Método

**Exercício:** Taxa de Importação

Criar pacote **main** com classe **Main** e pacote **importação** com a classe pai **Mundo** e as classes filhas **Brasil**, **China**, **EUA** e **Franca**. Na classe **Mundo** criar um método **taxa de importação (+200)** e outro **taxa de exportação (+10%)**. Na China a taxa de exportação é de (+5%) e nos EUA a taxa de importação é de (+100).

## 4. Classe Abstrata

Classe Abstrata (*abstract*)

- Só é usado em herança
- Em classe Abstrata não é possível instanciar o objeto da Classe.
- A palavra chave da classe abstrata é *abstract*

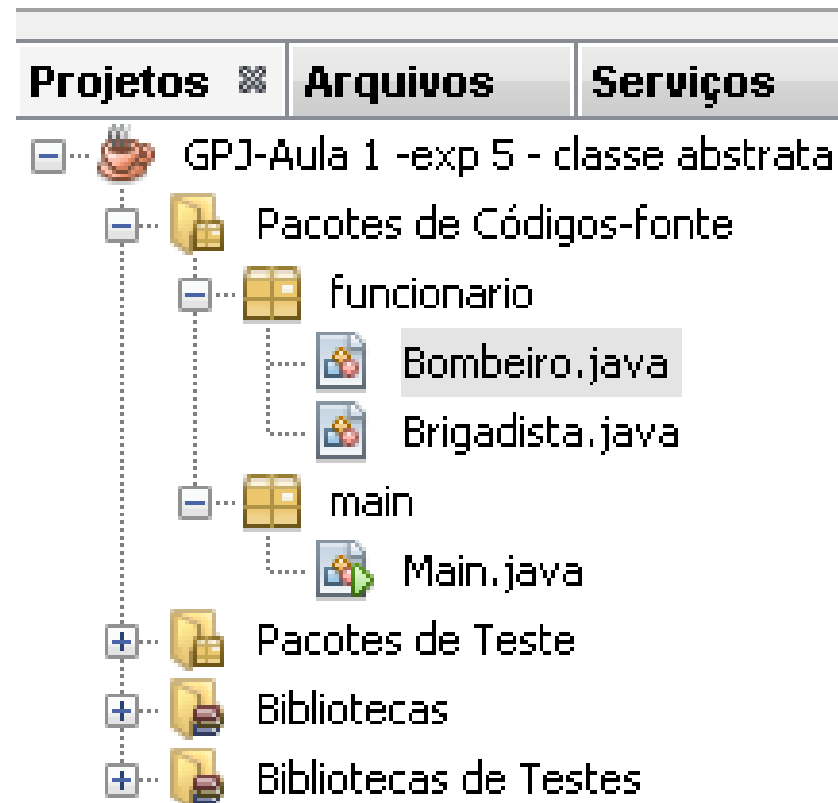
Exemplo: `public abstract class Nome_da_classe`

## 4. Classe Abstrata

- Por que criar uma Classe para não instanciar o objeto?
- Porque ela pode ser usada como modelo (classe pai) para as outras classes (classes filhas).
- Pode-se instanciar os objetos das classes filhas normalmente, porém não se instancia os objetos da classe pai.

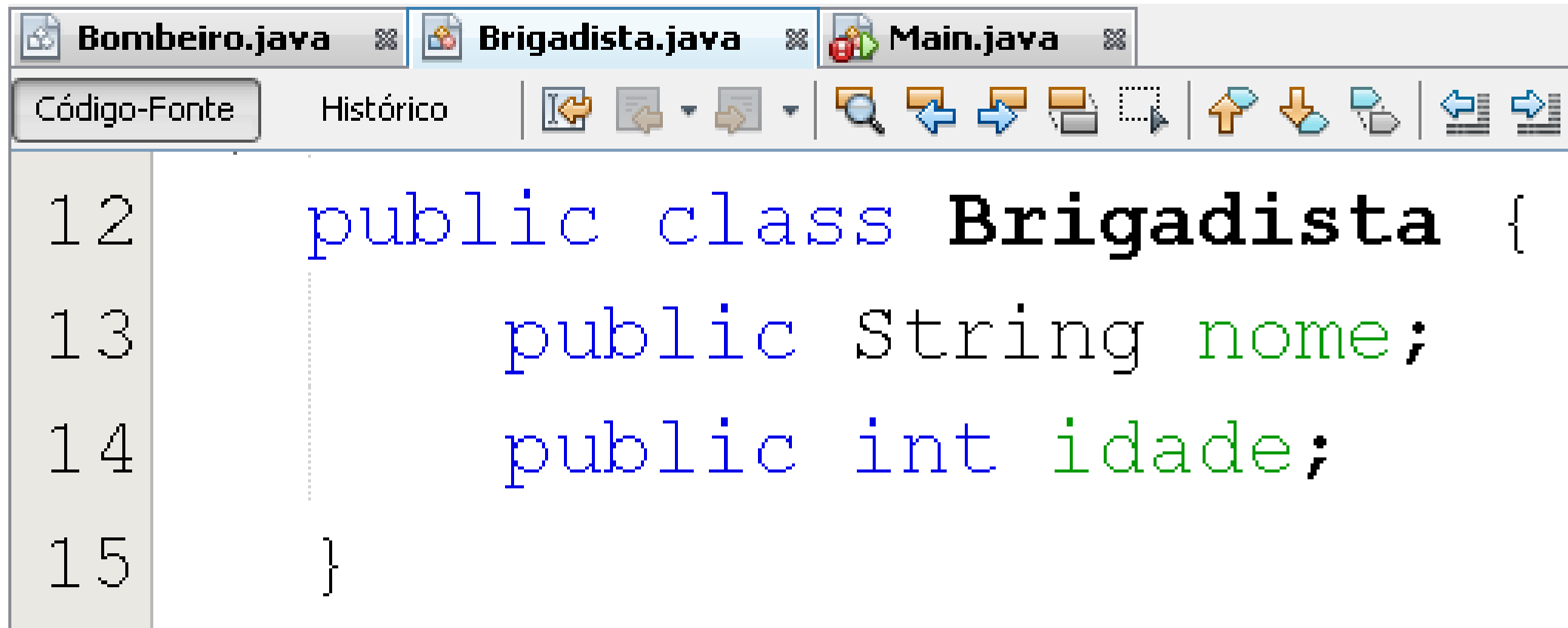
## 4. Classe Abstrata

- Exemplo classe Abstrata: Criar Pacote **funcionário** com as classes **Bombeiro** e **Brigadista**. Criar pacote **main** e classe **Main**.



## 4. Classe Abstrata

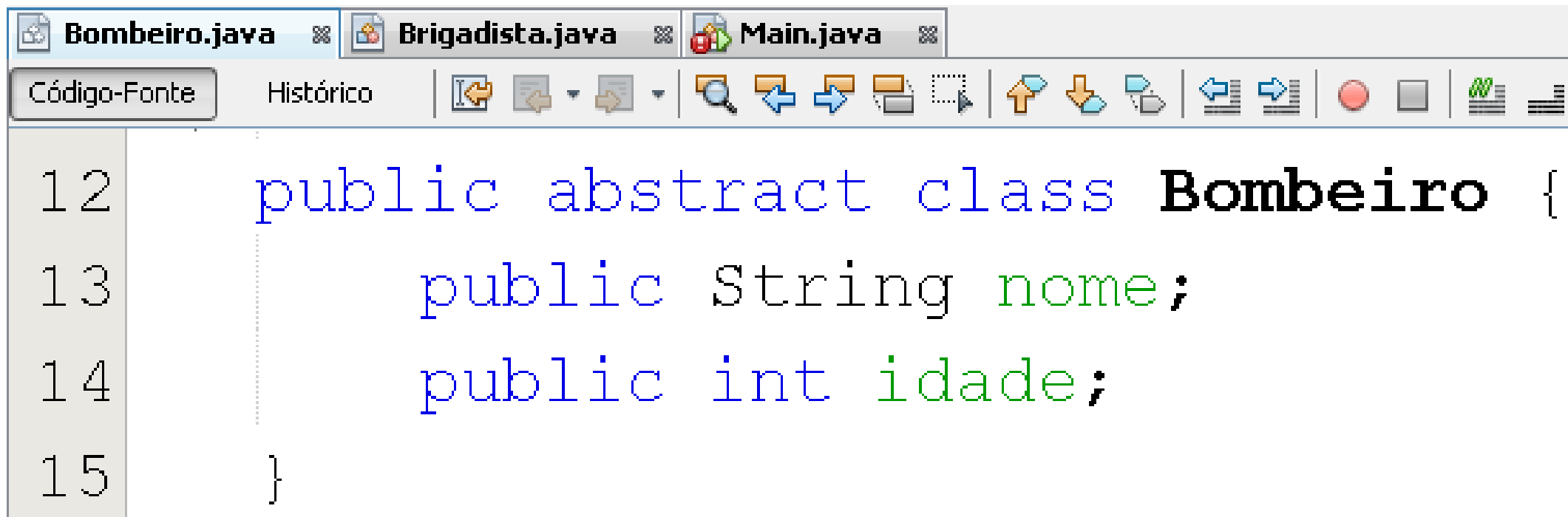
- Na classe **Brigadista** criar os atributos **nome** e **idade**



```
Bombeiro.java Brigadista.java Main.java
Código-Fonte Histórico
1 2 public class Brigadista {
1 3 public String nome;
1 4 public int idade;
1 5 }
```

## 4. Classe Abstrata

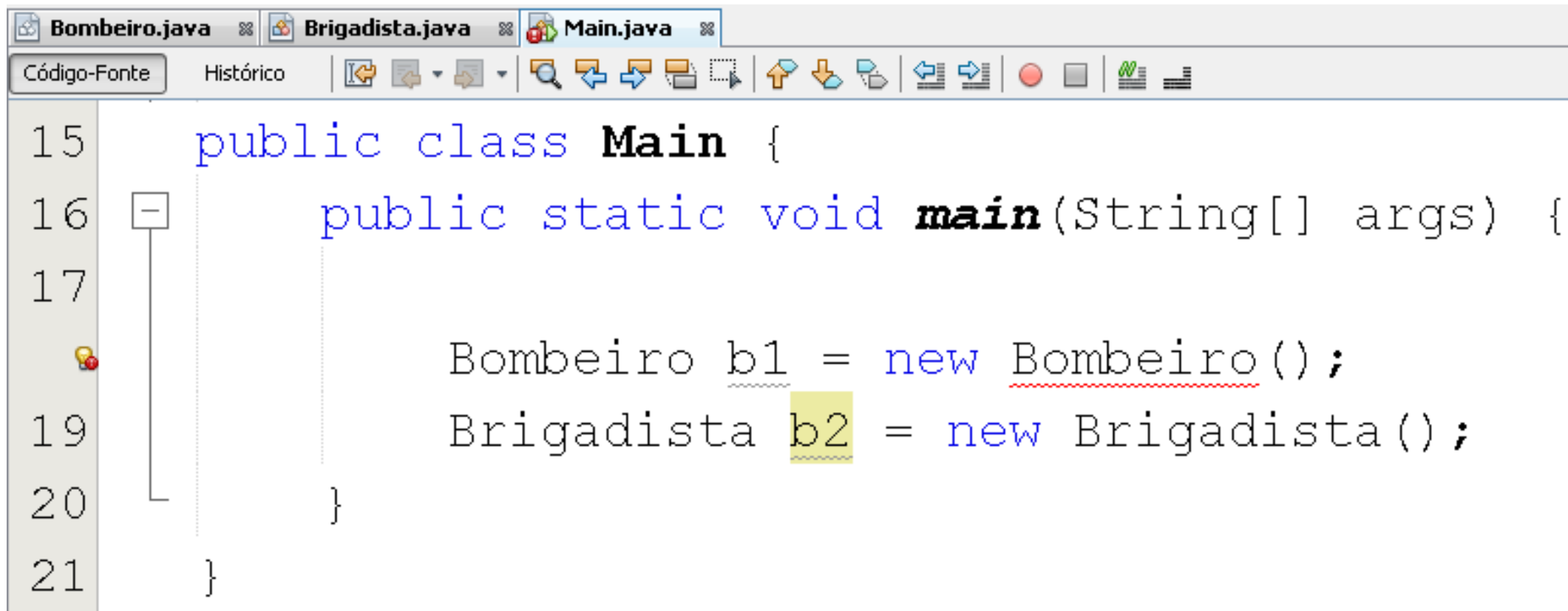
- Na classe **Bombeiro** colocar como classe abstrata e criar os atributos **nome** e **idade**.



```
Bombeiro.java Brigadista.java Main.java
Código-Fonte Histórico
12 public abstract class Bombeiro {
13 public String nome;
14 public int idade;
15 }
```

## 4. Classe Abstrata

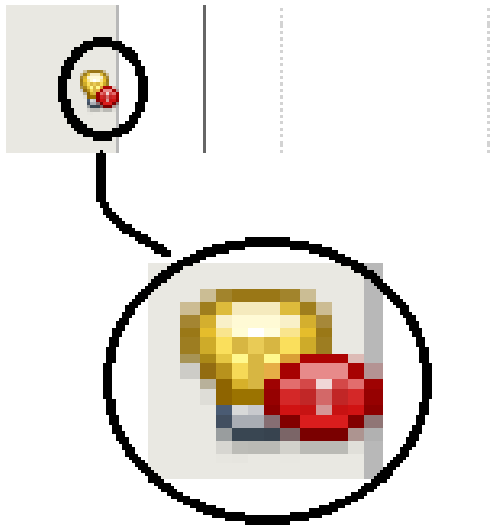
- Na classe **Main** instanciar os objetos das classes **Bombeiro** e **Brigadista**.



```
Bombeiro.java Brigadista.java Main.java
Código-Fonte Histórico
15 public class Main {
16 public static void main(String[] args) {
17
18 Bombeiro b1 = new Bombeiro();
19 Brigadista b2 = new Brigadista();
20 }
21 }
```

## 4. Classe Abstrata

- Note que na classe **Bombeiro** aparece um erro, esse erro é devido a classe ser abstrata, então não é possível instanciar o objeto da classe.



```
Bombeiro b1 = new Bombeiro();
```

Salvar o programa e deixar o erro para ficar como exemplo.



## 5. Método Abstrato

- Método abstrato é uma função do Java que dificilmente você cria, porém alguns métodos prontos já vem com essa função.
- No Java, só é permitido usar esse método em uma classe abstrata.
- Cada classe filha determina o que o método faz, porém os métodos são herdados da classe pai.

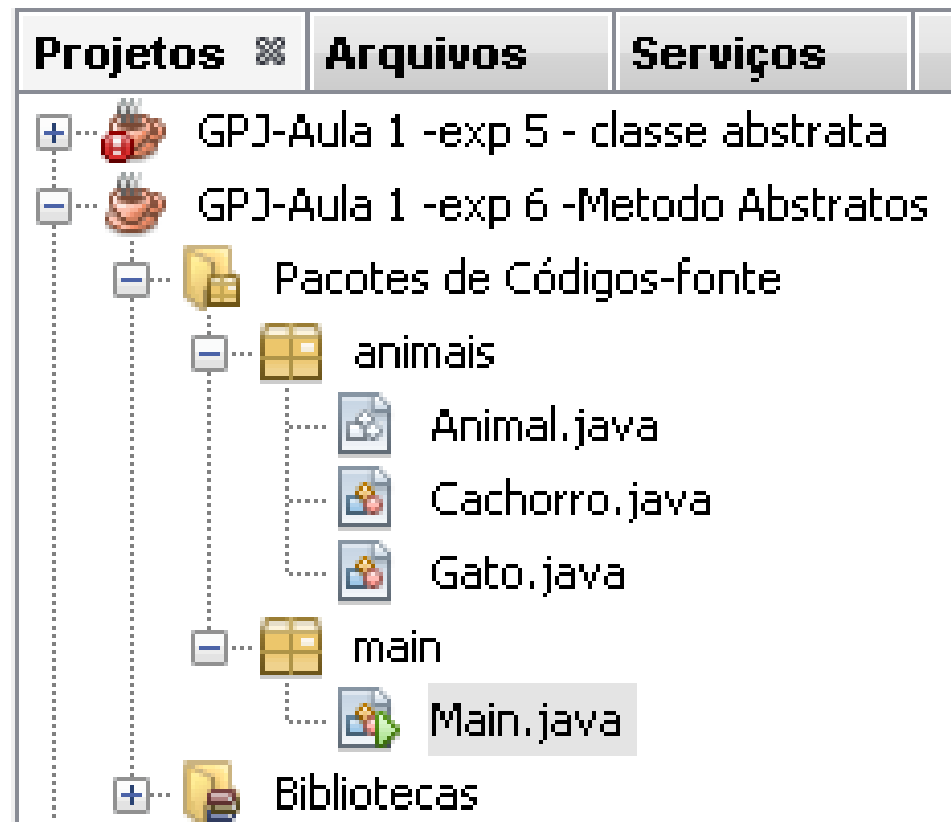
## 5. Método Abstrato

- A palavra chave para o método abstrato, assim como na classe, também é *abstract*.
- O método abstrato na classe pai termina com “;”

Exemplo: `public abstract void Nome_do_método ( );`

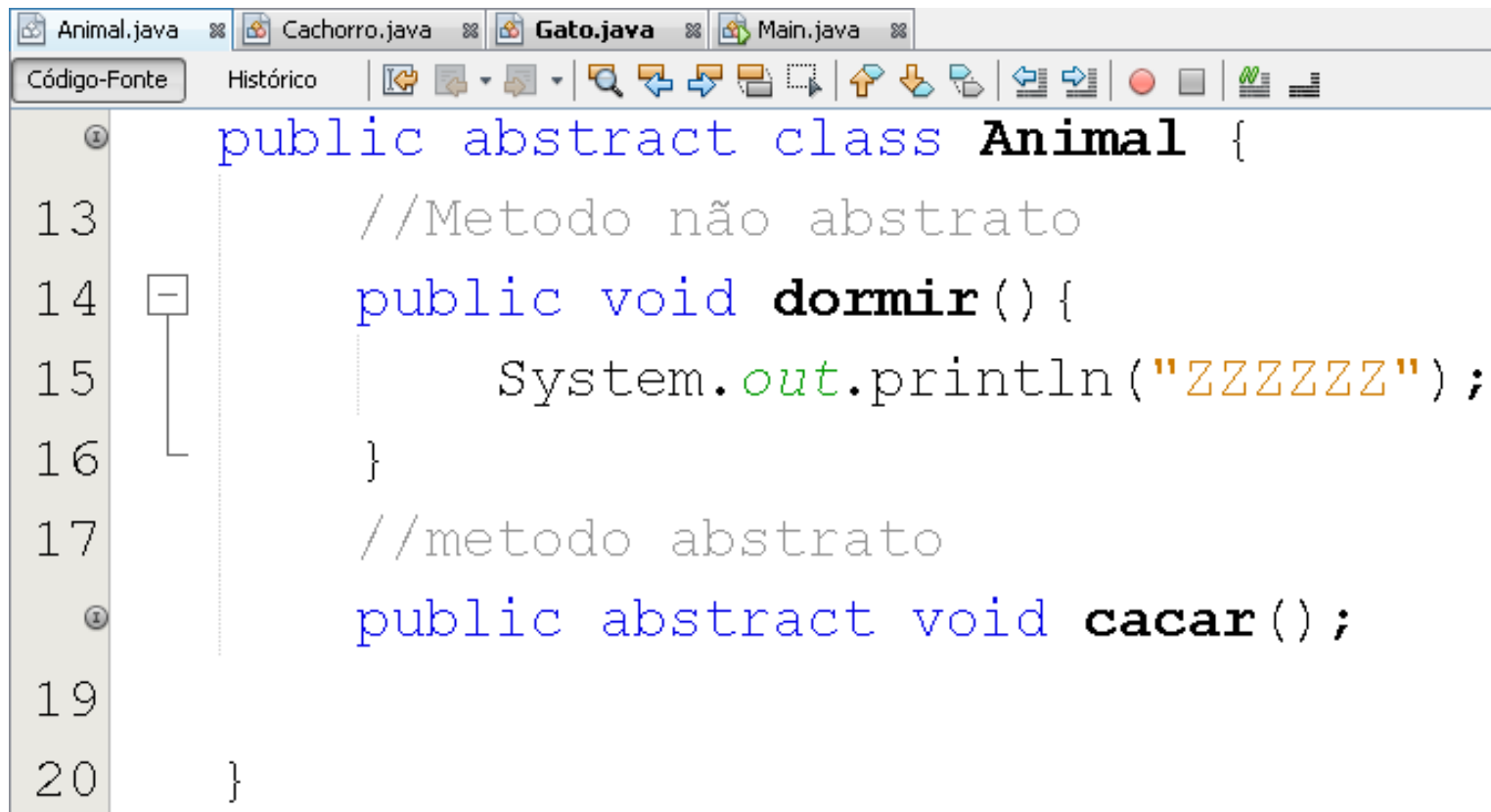
## 5. Método Abstrato

**Exemplo Método Abstrato:** Criar pacote **animais** com as classes **Animais**, **Cachorro** e **Gato**. Criar pacote **main** e classe **Main**.



## 5. Método Abstrato

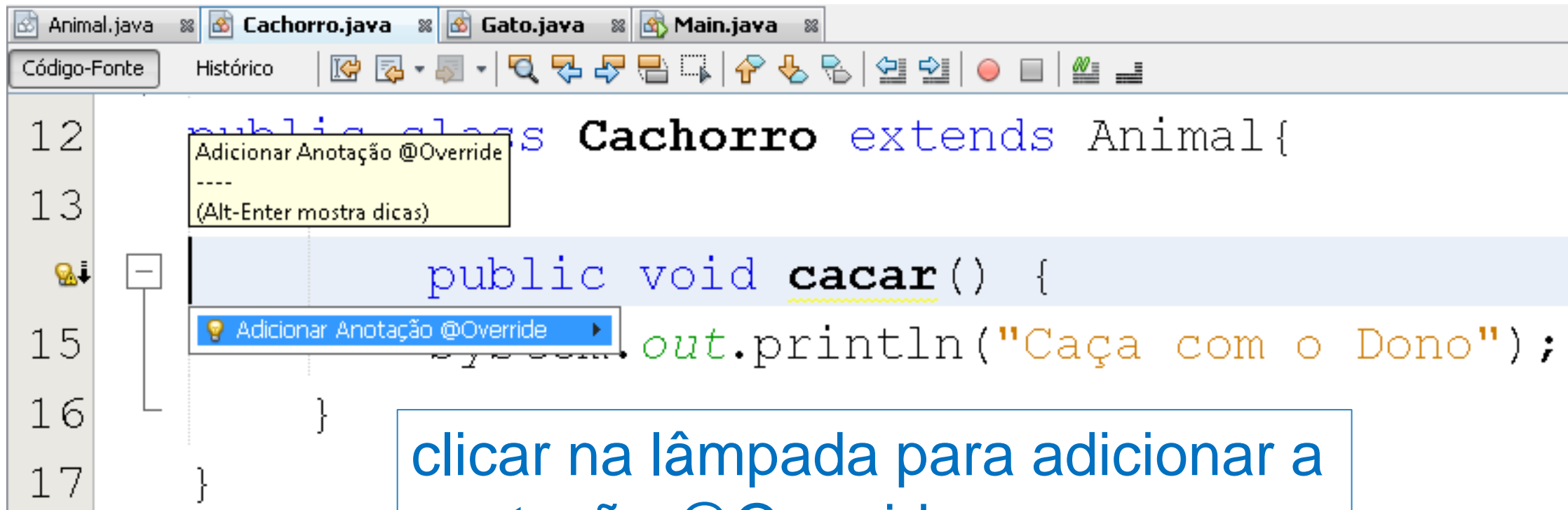
Na classe **Animal**, deixa-la como abstrata, criar o método **dormir** e o método abstrato **caçar**.



```
Animal.java Cachorro.java Gato.java Main.java
Código-Fonte Histórico
13 //Metodo não abstrato
14 public void dormir() {
15 System.out.println("ZZZZZZ");
16 }
17 //metodo abstrato
18 public abstract void cacar();
19
20 }
```

## 5. Método Abstrato

Colocar a classe **Cachorro** como filha da classe **Animal** e criar o método **cacar**.



```
12 public class Cachorro extends Animal {
13
14 public void cacar() {
15 out.println("Caça com o Dono");
16 }
17 }
```

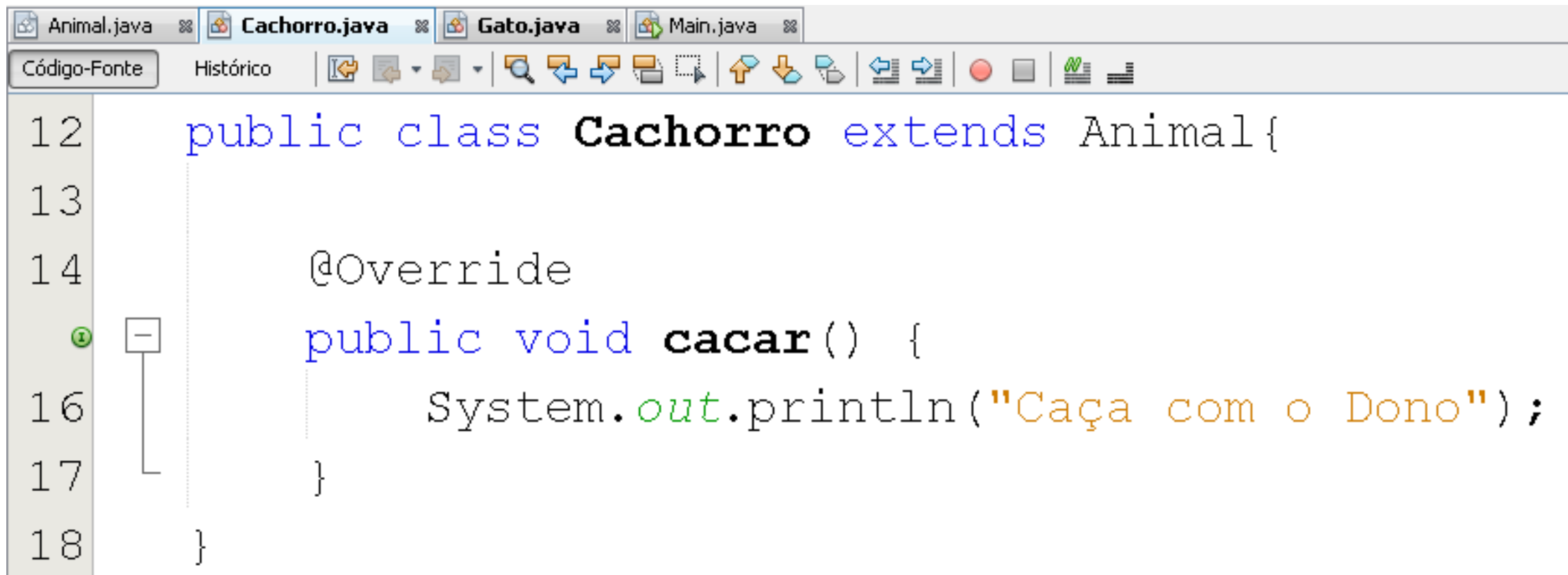
Adicionar Anotação @Override  
----  
(Alt-Enter mostra dicas)

Adicionar Anotação @Override

clicar na lâmpada para adicionar a anotação @Override

## 5. Método Abstrato

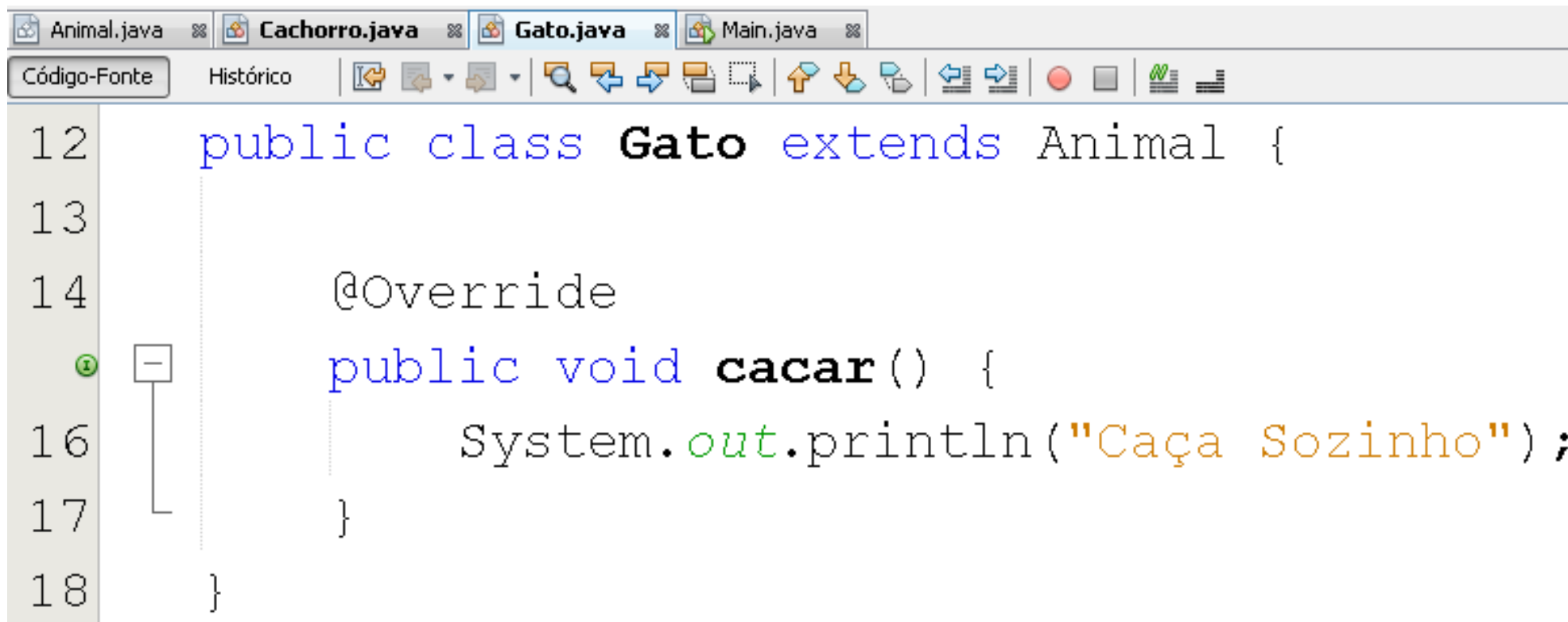
Após adicionar a anotação @Override.



```
Animal.java Cachorro.java Gato.java Main.java
Código-Fonte Histórico
12 public class Cachorro extends Animal{
13
14 @Override
15 public void cacar() {
16 System.out.println("Caça com o Dono");
17 }
18 }
```

## 5. Método Abstrato

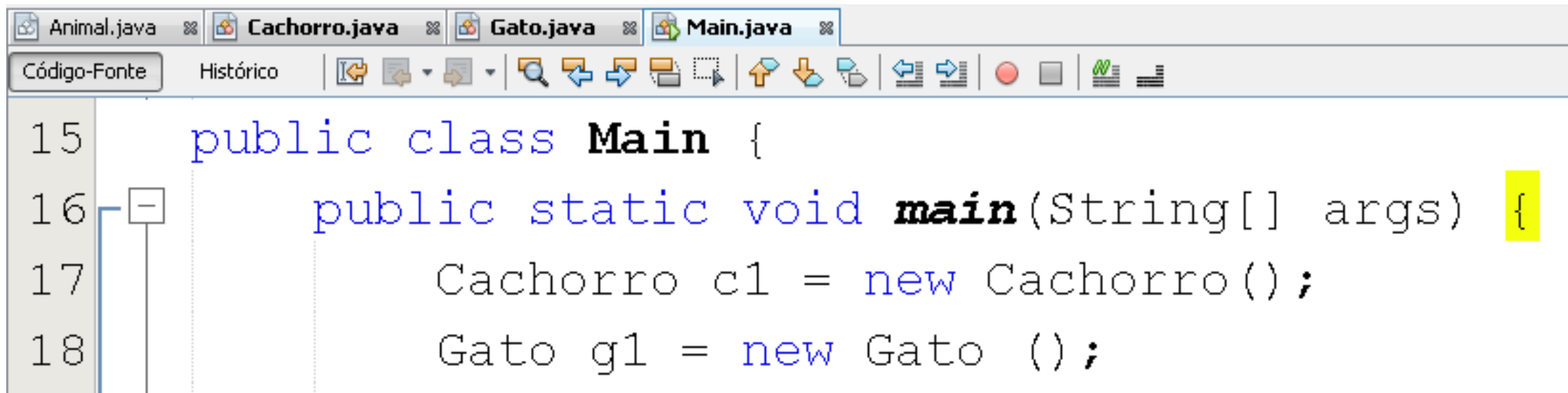
Faça o mesmo procedimento da classe **Cachorro** para a classe **Gato**.



```
Animal.java Cachorro.java Gato.java Main.java
Código-Fonte Histórico
12 public class Gato extends Animal {
13
14 @Override
15 public void cacar() {
16 System.out.println("Caça Sozinho");
17 }
18 }
```

## 5. Método Abstrato

Instanciar os objetos das classes **Cachorro** e **Gato** na classe **Main**.



```
Animal.java x Cachorro.java x Gato.java x Main.java x
Código-Fonte Histórico [Icons]
15 public class Main {
16 public static void main(String[] args) {
17 Cachorro c1 = new Cachorro();
18 Gato g1 = new Gato ();
```



## 5. Método Abstrato

Após instanciar os objetos chamar os métodos.

```
20 g1.cacar();
21 g1.dormir();
22
23 c1.cacar();
24 c1.dormir();
```

Note que os métodos podem ser chamados na classe principal

## 5. Método Abstrato

Veja como fica o resultado com os métodos abstrato

```

Saída - GPJ-Aula 1 -exp 6 -Metodo Abstratos (run)

run:
Caça Sozinho
ZZZZZZ
Caça com o Dono
ZZZZZZ
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)

```

Repare que as maneiras de caçar do cachorro é diferente da maneira do gato e que ambos adotaram o método de dormir.

## 5. Método Abstrato

**Exercício:** Criar 2 pacotes: 1) **main** que terá a classe **Main** e 2) **func** com a classe pai **Funcionario** e as classes filhas **Adm**, **Financeiro** e **Ti**. **Funcionario** é uma classe abstrata com os métodos **trabalha** (abstrata) e **horas\_trabalhadas**(não Abstrata). Todos trabalham das 09:00 às 17:00, porém **Ti** é das 09:00 às 20:00.

## 6. Interface

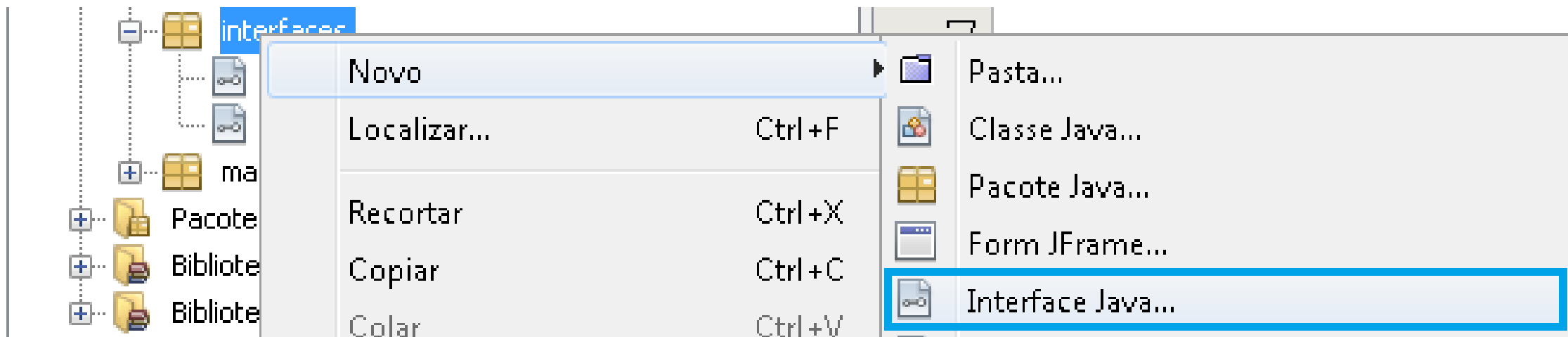
- Quantos pai uma pessoa pode ter?
- No Java é o mesmo, só pode existir uma classe pai para cada classe filha.
- Quantos contratos uma pessoa ou empresa pode ter?
- Interface pode ser comparada com um contrato, uma pessoa só pode ter um pai, porém ela pode assinar (contratar) vários contratos.

## 6. Interface

- Importante lembrar que uma interface não é uma classe.
- Uma interface só fornece métodos abstrato, porém **não** se escreve abstract.
- A classe que contratar uma interface é quem deve definir o que faz o método, idêntico ao método abstrato.

## 6. Interface

- Uma interface famosa é a Runner.
- Criar uma interface é bem parecido com criar uma classe, porém, ao invés de clicar em classe, clica-se em interface Java.



## 6. Interface

- Boas práticas de programação colocamos a letra “I” antes do nome da interface.

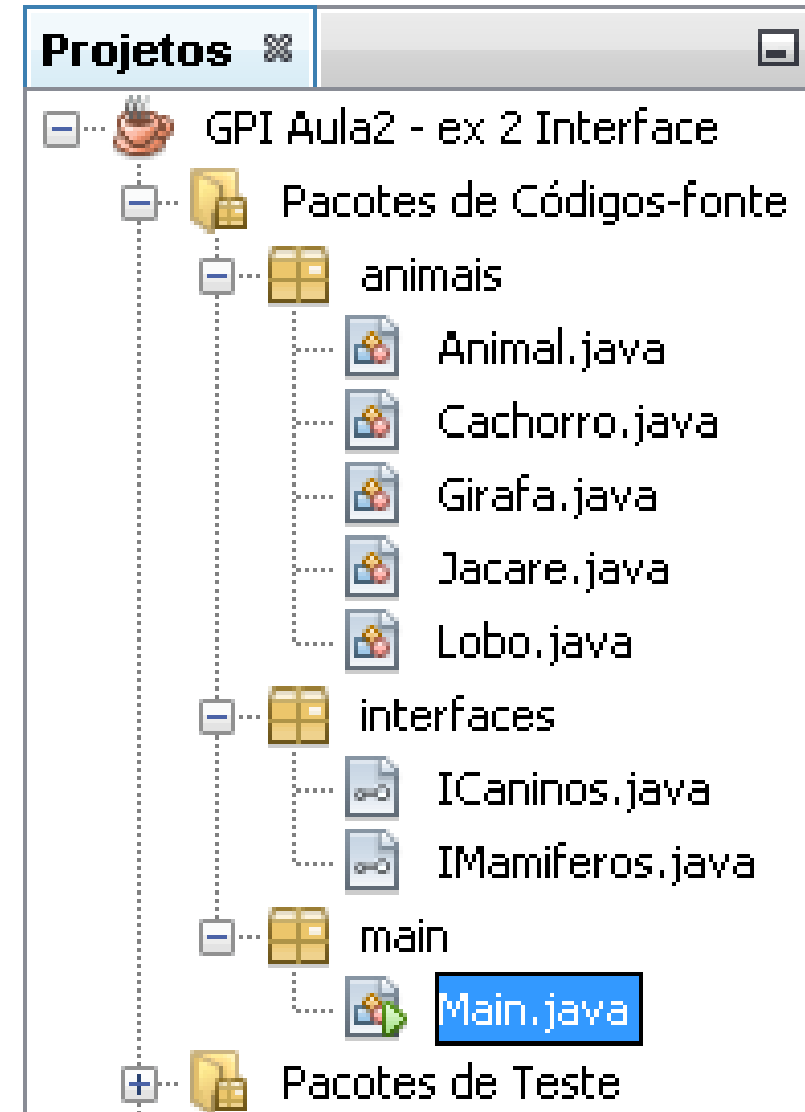
Exemplo: `public interface` Inome\_da\_interface

- Para contratar uma interface é muito parecido com herança, porém usa-se a palavra chave *implements*.

Exemplo: `public class` Nome\_da\_Classe `implements`  
Inome\_da\_Interface

## 6. Interface

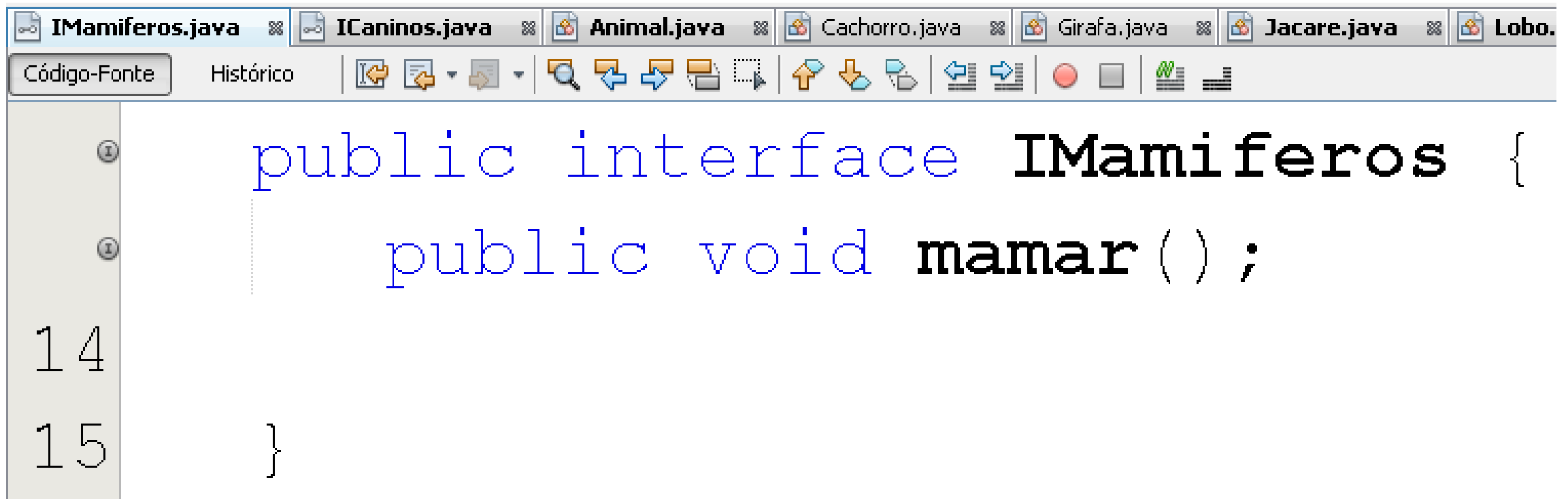
**Exemplo de Interface:** Criar pacote **animais** com as classes **Animal**, **Cachorro**, **Jacare**, **Girafa** e **Lobo**. Criar pacote **main** e classe **Main**. Criar o pacote **interface** com as Interfaces **ICaninos** e **IMamiferos**.





## 6. Interface

Na interface **IMamiferos** criar o método **mamar( )**;



```
public interface IMamiferos {
 public void mamar();
}
```

The screenshot shows an IDE window with several tabs: IMamiferos.java, ICaninos.java, Animal.java, Cachorro.java, Girafa.java, Jacare.java, and Lobo.java. The IMamiferos.java tab is active, showing the code for the IMamiferos interface. The code is as follows:

```
14 public interface IMamiferos {
15 public void mamar();
}
```

## 6. Interface

Na interface **ICaninos** criar os métodos **latir ( )**;  
e **uivar ( )**;

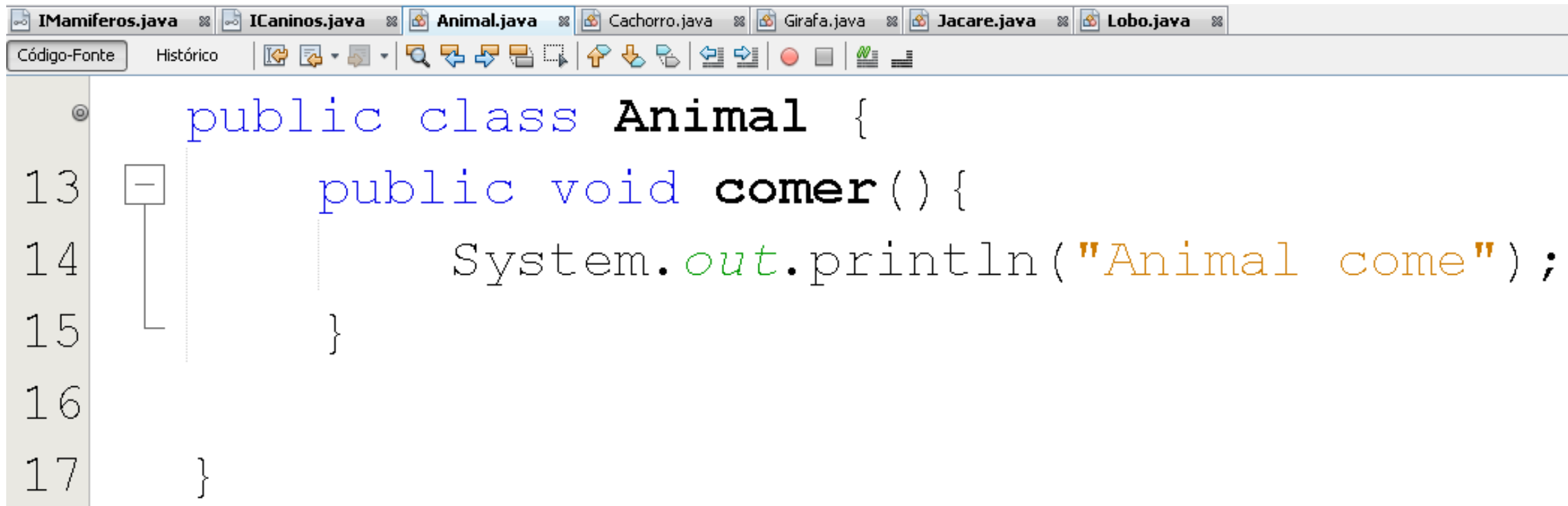
A screenshot of a Java IDE window showing the `ICaninos.java` file. The window has several tabs open: `IMamiferos.java`, `ICaninos.java` (selected), `Animal.java`, `Cachorro.java`, `Girafa.java`, and `Jacare.java`. The `ICaninos.java` tab is active, displaying the following code in a monospaced font:

```
public interface ICaninos {
 public void latir();
 public void uivar();
}
```

The line numbers 15 and 16 are visible on the left margin. The code is written in blue and black text. The IDE interface includes a toolbar with various icons for file operations and a status bar at the bottom left showing "Arquivo principal."

## 6. Interface

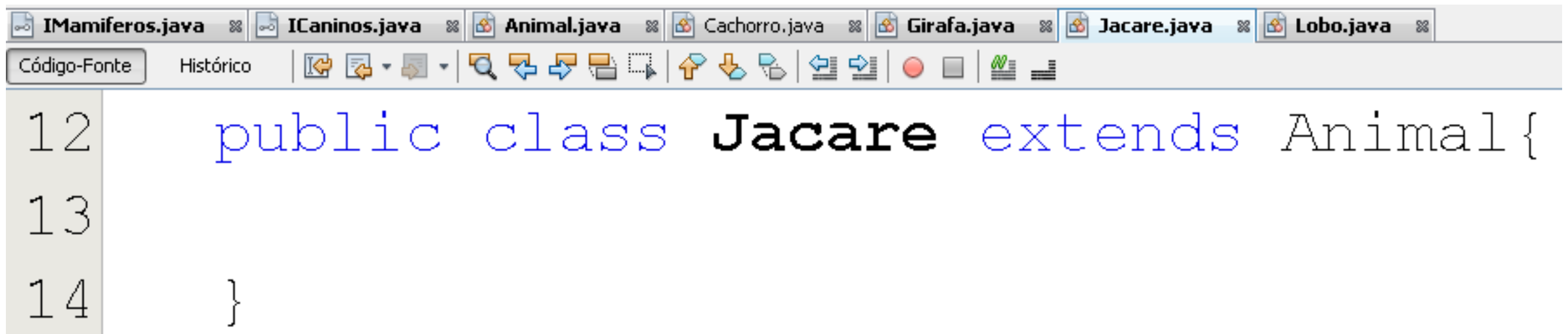
Na classe **Animal**, que será pai das demais classes, criar o método **comer**.



```
IMamiferos.java ICaninos.java Animal.java Cachorro.java Girafa.java Jacare.java Lobo.java
Código-Fonte Histórico
13 public class Animal {
14 public void comer() {
15 System.out.println("Animal come");
16 }
17 }
```

## 6. Interface

Colocar a classe **Jacare** como filha da classe **Animal**.

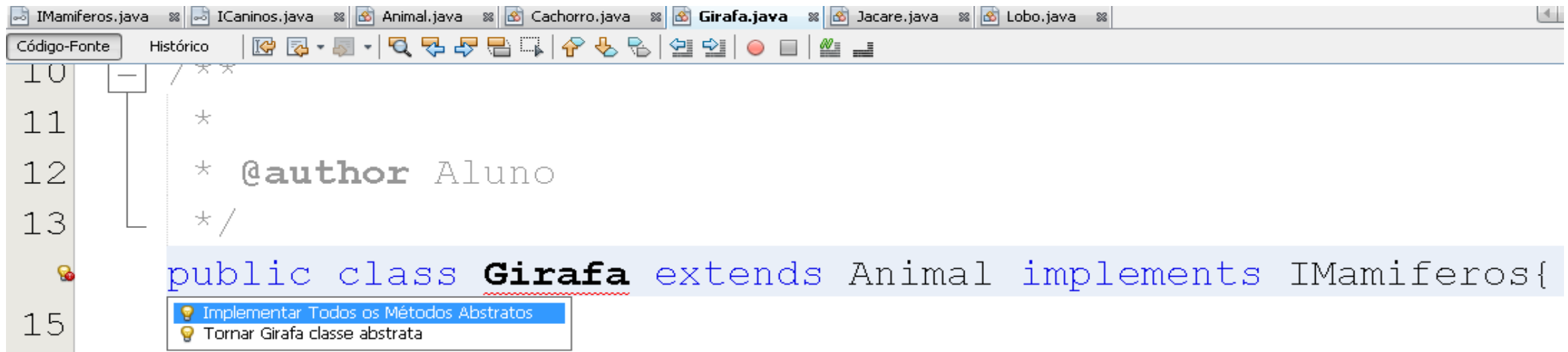


```
12 public class Jacare extends Animal{
13
14 }
```

The screenshot shows an IDE window with multiple tabs: IMamiferos.java, ICaninos.java, Animal.java, Cachorro.java, Girafa.java, Jacare.java (selected), and Lobo.java. The toolbar includes icons for file operations, search, and execution. The code editor displays the following Java code:

## 6. Interface

Colocar a classe **Girafa** como filha da classe **Animal** e contratar a interface **IMamiferos**.



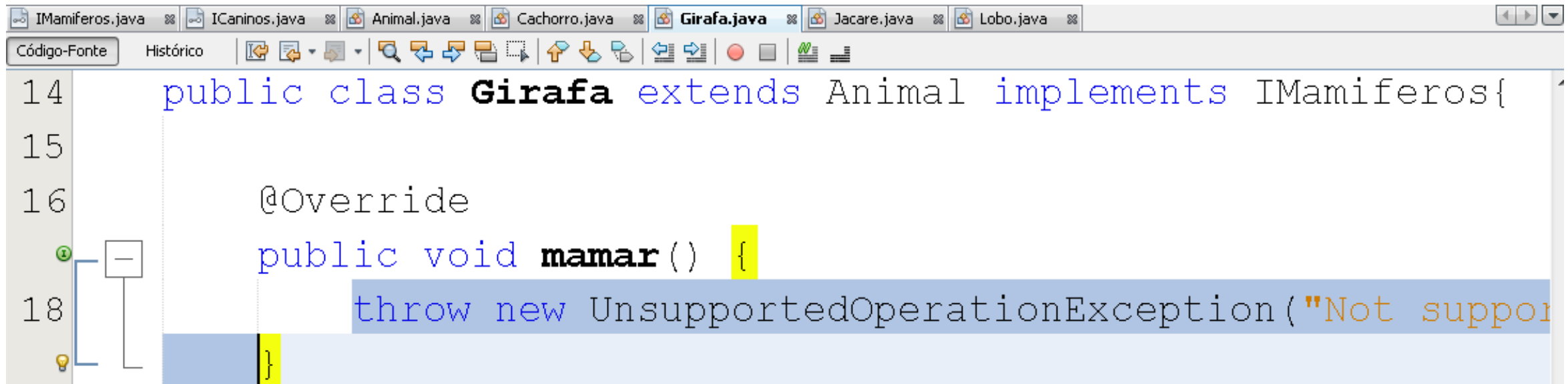
```
10 /**
11 *
12 * @author Aluno
13 */
14 public class Girafa extends Animal implements IMamiferos{
15
```

Implementar Todos os Métodos Abstratos  
Tornar Girafa classe abstrata

Clicar na lâmpada que aparecerá do lado esquerdo e clicar em implementar Todos os Métodos Abstratos.

## 6. Interface

Após implementar os métodos abstratos, irá aparecer o método **mamar** com a notação `@Override` e a frase *“throw new UnsupportedOperationException(“Not ...”)*”

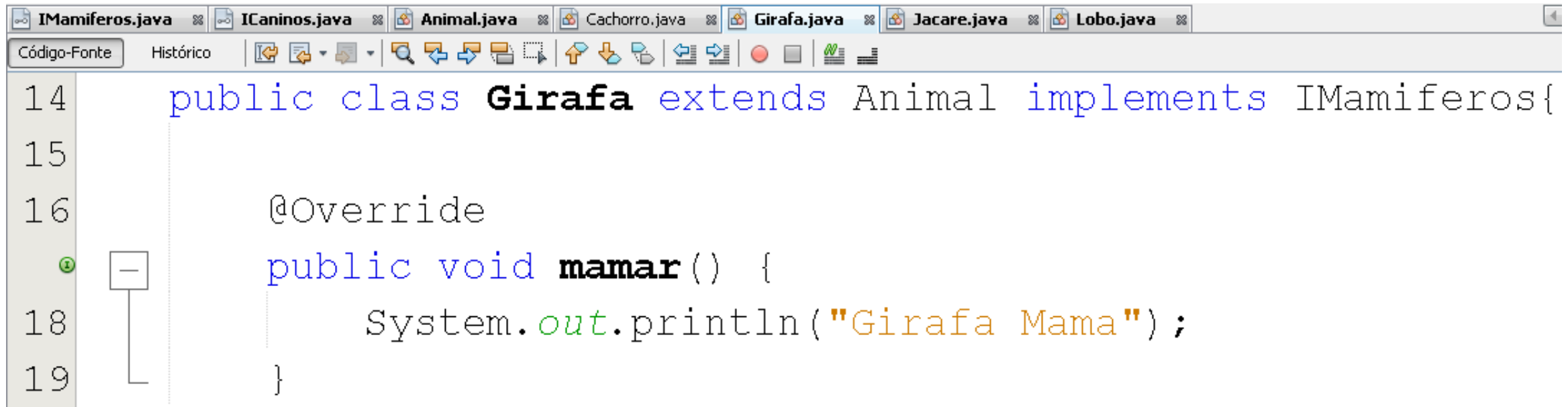


```
14 public class Girafa extends Animal implements IMamiferos{
15
16 @Override
17 public void mamar() {
18 throw new UnsupportedOperationException("Not support");
19 }
```

The screenshot shows an IDE window with several tabs: IMamiferos.java, ICaninos.java, Animal.java, Cachorro.java, Girafa.java (active), Jacare.java, and Lobo.java. The code editor displays the implementation of the `mamar` method in the `Girafa` class. The method is annotated with `@Override` and throws a `UnsupportedOperationException` with the message "Not support". The code is syntax-highlighted, and the IDE interface includes a toolbar and a sidebar with a search icon and a lightbulb icon.

## 6. Interface

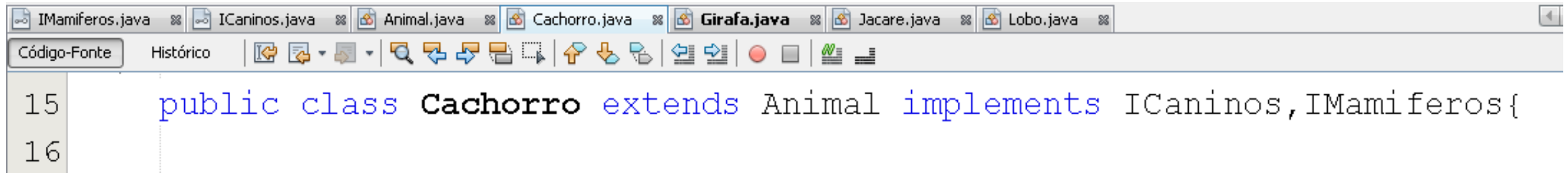
Apagar a frase “*throw new UnsupportedOperationException*(“Not ...”)” e adicionar a ação para o método.



```
14 public class Girafa extends Animal implements IMamiferos{
15
16 @Override
17 public void mamar() {
18 System.out.println("Girafa Mama");
19 }
```

## 6. Interface

Colocar a classe **Cachorro** como filha da classe **Animal** e contratar as interfaces **ICaninos** e **IMamiferos**.



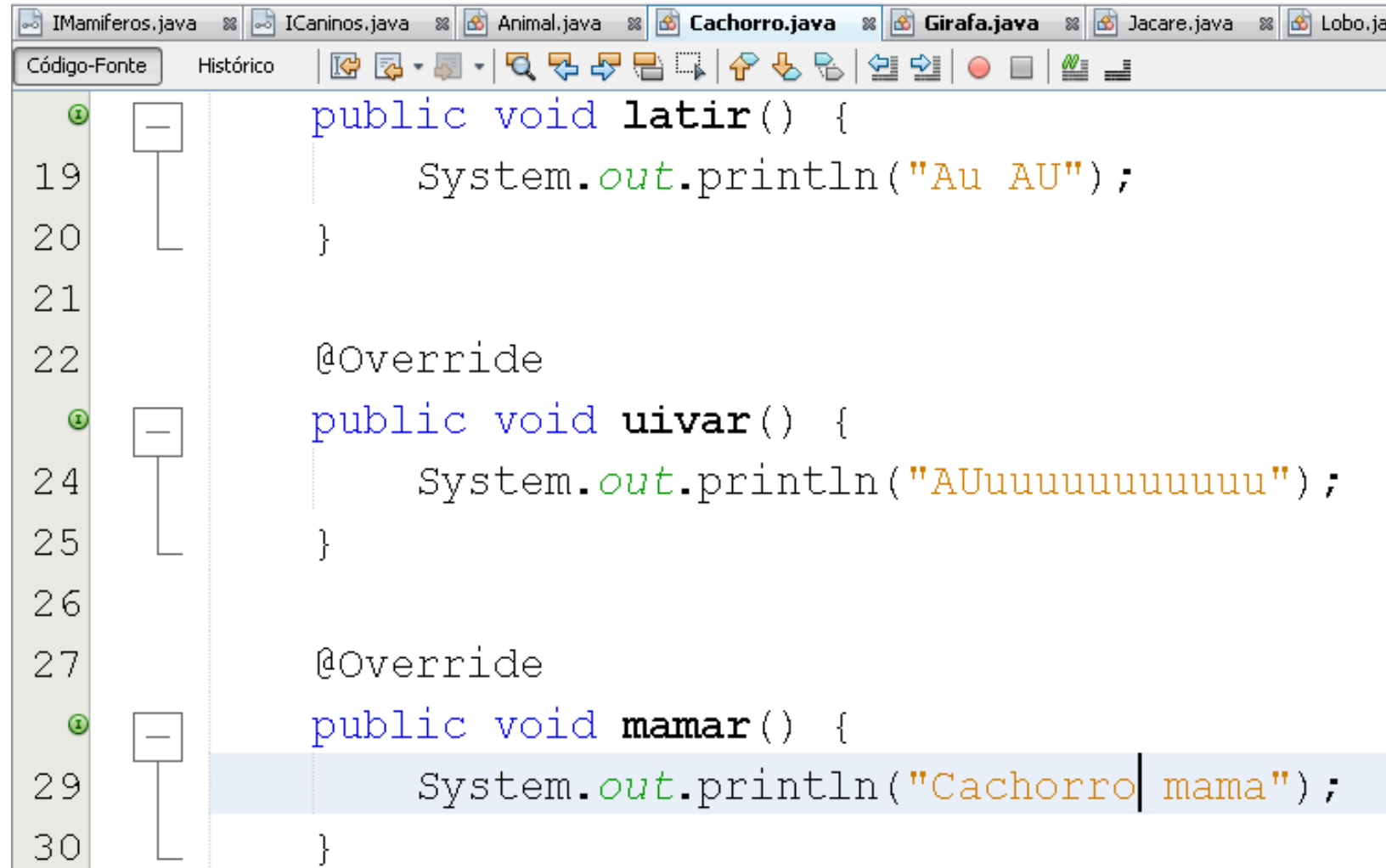
```
IMamiferos.java ICaninos.java Animal.java Cachorro.java Girafa.java Jacare.java Lobo.java
Código-Fonte Histórico [Icons]
15 public class Cachorro extends Animal implements ICaninos, IMamiferos{
16
```

Implementar os métodos abstratos e apagar a frase que aparecerá, fazendo o mesmo procedimento da classe **Girafa**.



## 6. Interface

Adicionar as ações dos métodos na classe **Cachorro**.



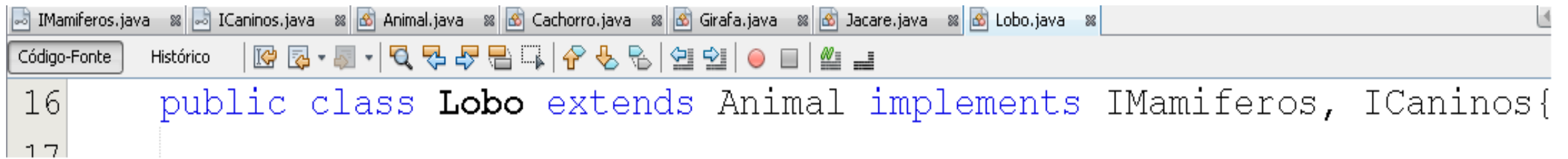
```
IMamiferos.java ICaninos.java Animal.java Cachorro.java Girafa.java Jacare.java Lobo.java
Código-Fonte Histórico
public void latir() {
 System.out.println("Au AU");
}

@Override
public void uivar() {
 System.out.println("AUuuuuuuuuuuuuuu");
}

@Override
public void mamar() {
 System.out.println("Cachorro mama");
}
```

## 6. Interface

Na classe **Lobo**, colocar como filha da classe **Animal** e contratar as interfaces **IMamiferos** e **ICaninos**.

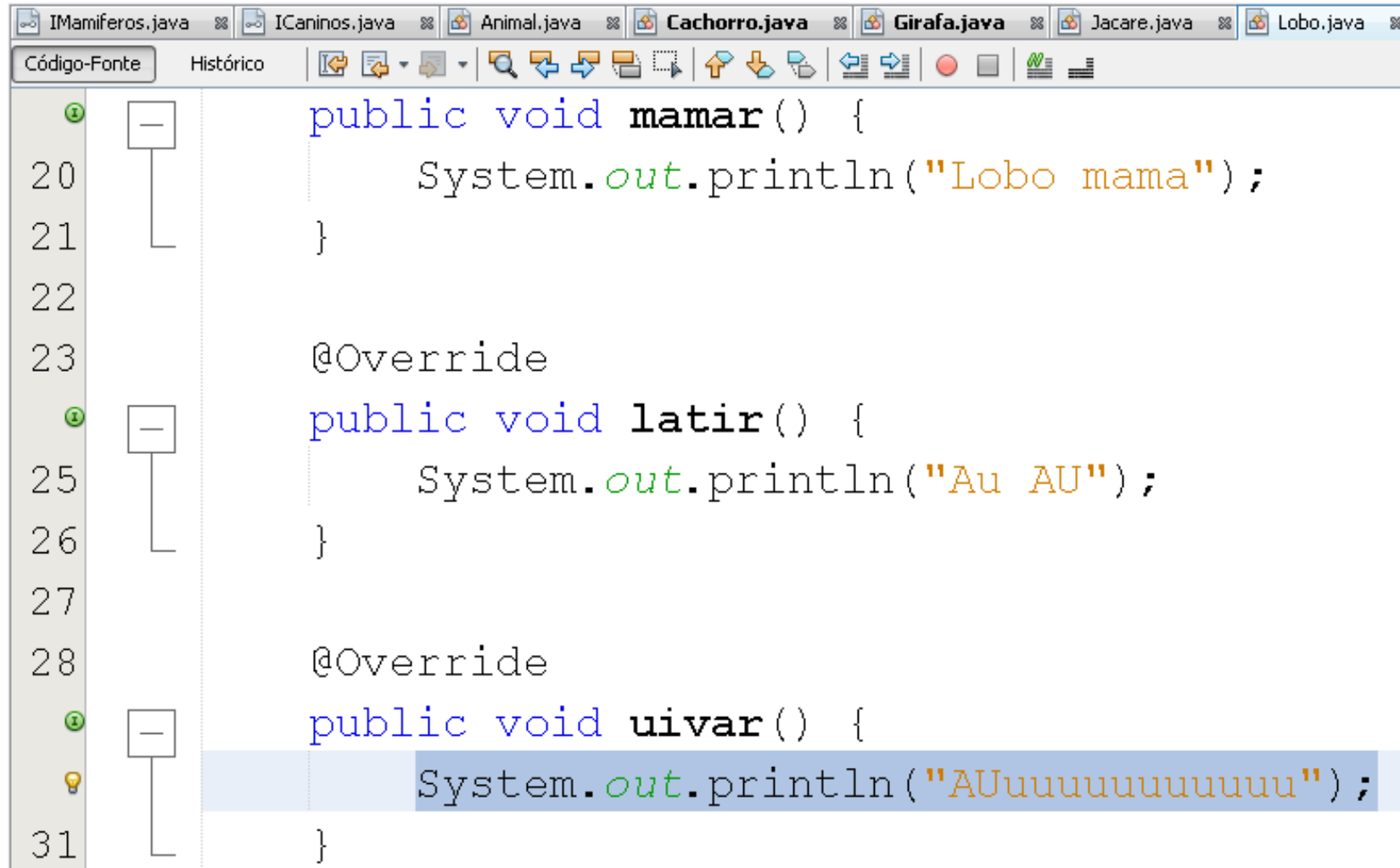


```
IMamiferos.java ICaninos.java Animal.java Cachorro.java Girafa.java Jacare.java Lobo.java
Código-Fonte Histórico
16 public class Lobo extends Animal implements IMamiferos, ICaninos{
17
```

Implementar os métodos abstratos e apagar a frase que aparecerá, fazendo o mesmo procedimento das classes **Girafa** e **Cachorro**.

## 6. Interface

Adicionar as ações dos métodos na classe **Lobo**.



```
IMamiferos.java ICaninos.java Animal.java Cachorro.java Girafa.java Jacare.java Lobo.java
Código-Fonte Histórico
public void mamar() {
 System.out.println("Lobo mama");
}

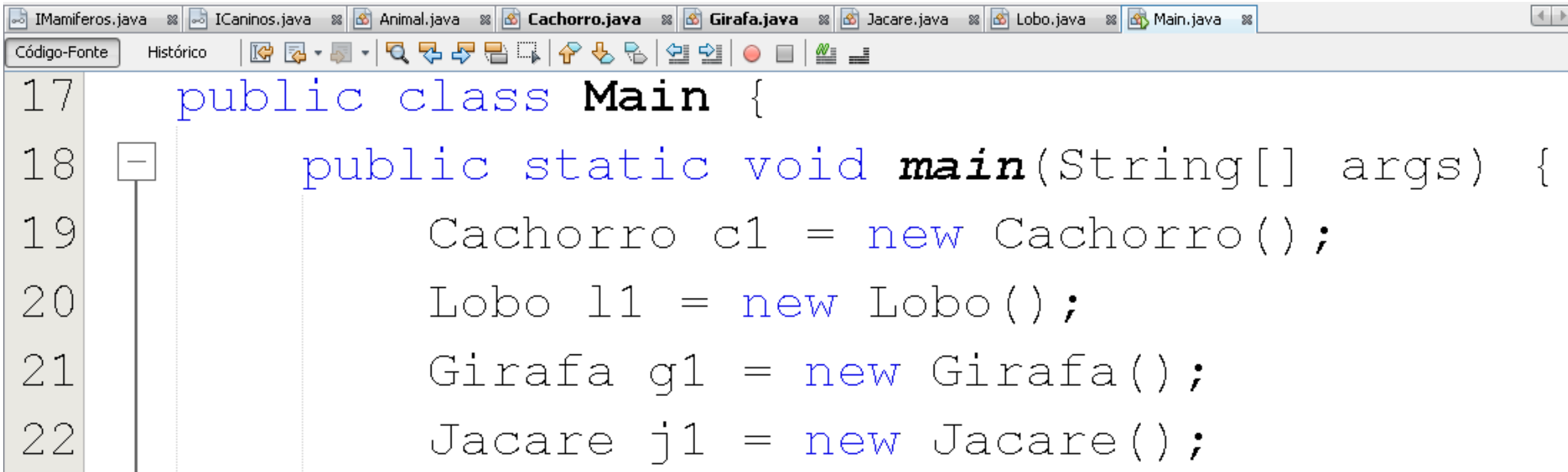
@Override
public void latir() {
 System.out.println("Au AU");
}

@Override
public void uiivar() {
 System.out.println("AUuuuuuuuuuuuuuu");
}
```

Repare que a ordem que se contrata a interface é a mesma que as ações aparecem.

## 6. Interface

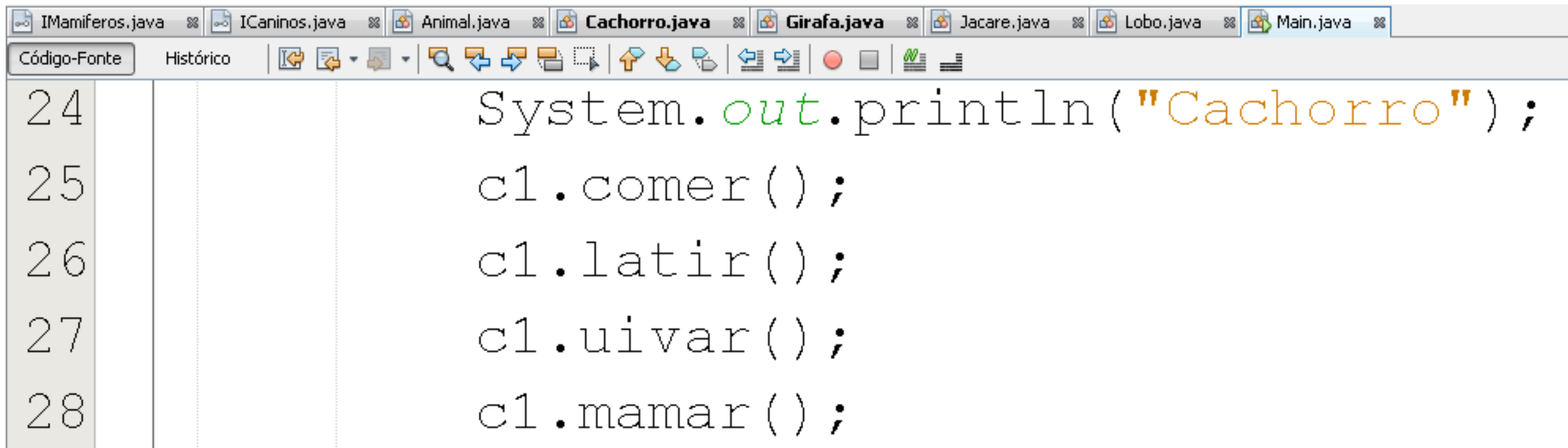
Na classe **Main** instanciar os objetos das classes **Cachorro**, **Lobo**, **Girafa** e **Jacare**.

A screenshot of a Java IDE window. The title bar shows several open files: IMamiferos.java, ICaninos.java, Animal.java, Cachorro.java, Girafa.java, Jacare.java, Lobo.java, and Main.java. The 'Main.java' file is selected and its source code is displayed in the editor. The code defines a public class Main with a main method that creates instances of Cachorro, Lobo, Girafa, and Jacare. Line numbers 17 through 22 are visible on the left side of the editor.

```
17 public class Main {
18 public static void main(String[] args) {
19 Cachorro c1 = new Cachorro();
20 Lobo l1 = new Lobo();
21 Girafa g1 = new Girafa();
22 Jacare j1 = new Jacare();
```

## 6. Interface

Após instanciar os objetos, chamar os métodos da classe **Cachorro**.



The screenshot shows an IDE window with several tabs: IMamiferos.java, ICaninos.java, Animal.java, Cachorro.java, Girafa.java, Jacare.java, Lobo.java, and Main.java. The 'Código-Fonte' (Source Code) tab is active. The code in Main.java is as follows:

```
24 System.out.println("Cachorro");
25 c1.comer();
26 c1.latir();
27 c1.uivar();
28 c1.mamar();
```

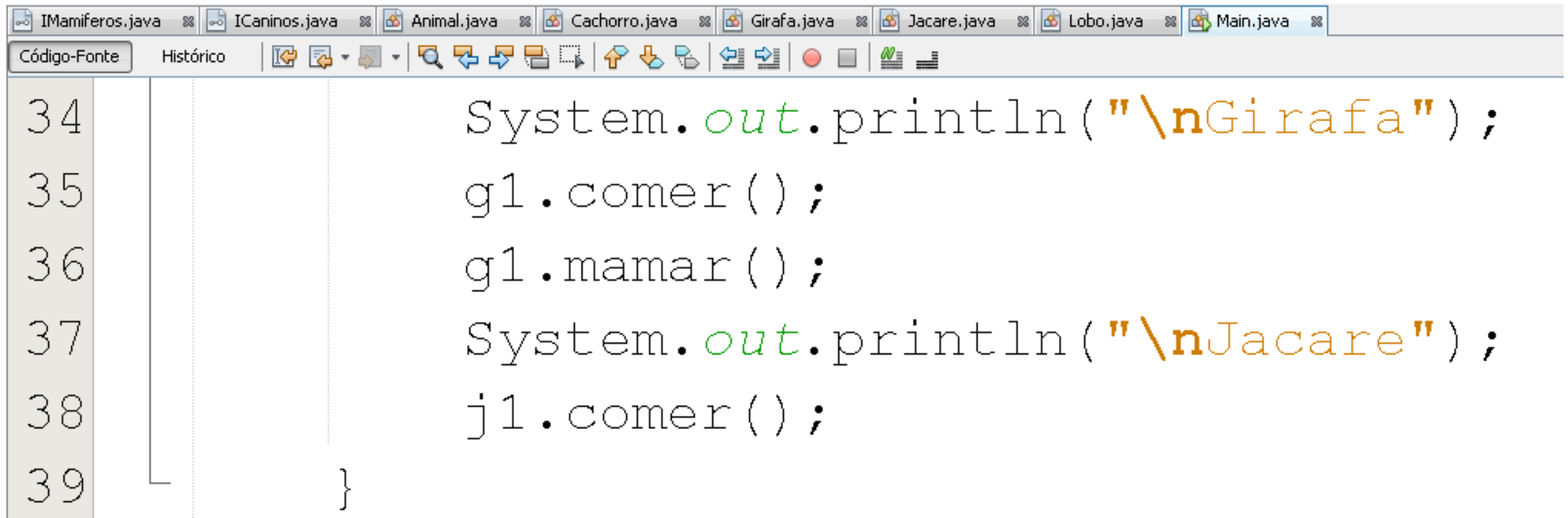
## 6. Interface

Após instanciar os objetos, chamar os métodos da classe **Lobo**.

```
29 System.out.println("\nLobo");
30 l1.comer();
31 l1.latir();
32 l1.uivar();
33 l1.mamar();
```

## 6. Interface

Após instanciar os objetos, chamar os métodos da classe **Girafa** e **Jacare**.



```
IMamiferos.java ICaninos.java Animal.java Cachorro.java Girafa.java Jacare.java Lobo.java Main.java
Código-Fonte Histórico
34 System.out.println("\nGirafa");
35 g1.comer();
36 g1.mamar();
37 System.out.println("\nJacare");
38 j1.comer();
39 }
```

## 6. Interface

## Veja como fica o resultado com a Interface

```
Saída - GPI Aula2 - ex 2 Interface (run)

run:
Cachorro
Animal come
Au AU
AUUUUUUUUUUUUUU
Cachorro mama

Lobo
Animal come
Au AU
AUUUUUUUUUUUUUU
Lobo mama

Girafa
Animal come
Girafa Mama

Jacare
Animal come
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

Repare que os métodos das interfaces aparecem nas classes. Note que como jacaré não é mamífero, ele só tem o método comer.



## 6. Interface

**Exercício:** Criar as classes **Brasil**, **SP**, **RJ**, **RS** e **BA**.

A Classe **Brasil** será pai das demais classes e terá o método abstrato **ImpostoLuz** e método **DescontoGas** com **10% de desconto**.

A Classe **SP** terá **ImpostoLuz** de **R\$50,00** e **descontoGas** de **5%**.

As classes **RJ**, **RS** e **BH** terão **ImpostoLuz** de **R\$75,00**; **R\$45,00** e **R\$30,00** respectivamente.

Criar as Interfaces **Região Sul** com os métodos **comida típica** e **dança típica**. **Região Nordeste** com método **comida mais vendida** e **Região Sudeste** com o método **Festa Típica**.

Dado o valor, aplicar o desconto e adicionar o imposto.

## 7. Encapsulamento

Encapsulamento ou Modificadores de acesso ou Especificador de acesso.

Aplicado em classes. É utilizado para saber quais classes tem acesso aos métodos e atributos.

Existem 4 tipos de modificadores de acesso:

## 7. Encapsulamento

-**public** -> Permite que todas as classes do projeto tenham acesso aos métodos e atributos do tipo public.

Exemplo: **public** String nome;

- **protected** -> Permite que todas as classes do mesmo pacote e as classes filhas tenham acesso aos métodos e atributos do tipo protected.

Exemplo: **protected** int idade;

## 7. Encapsulamento

**-private** -> Permite que somente a própria classe tenha acesso aos métodos e atributos do tipo private.

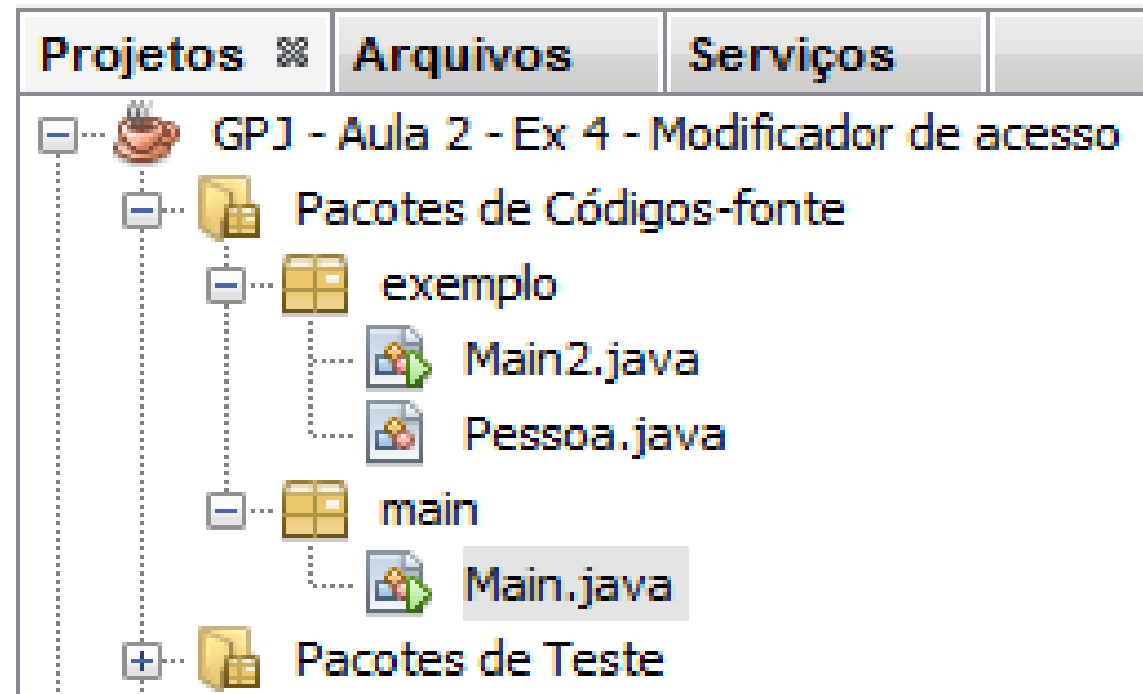
Exemplo: `private String RG;`

**-nenhum modificador(default)** -> Permite que somente as Classes do mesmo pacote tenham acesso aos métodos e atributos.

Exemplo: `String CPF;`

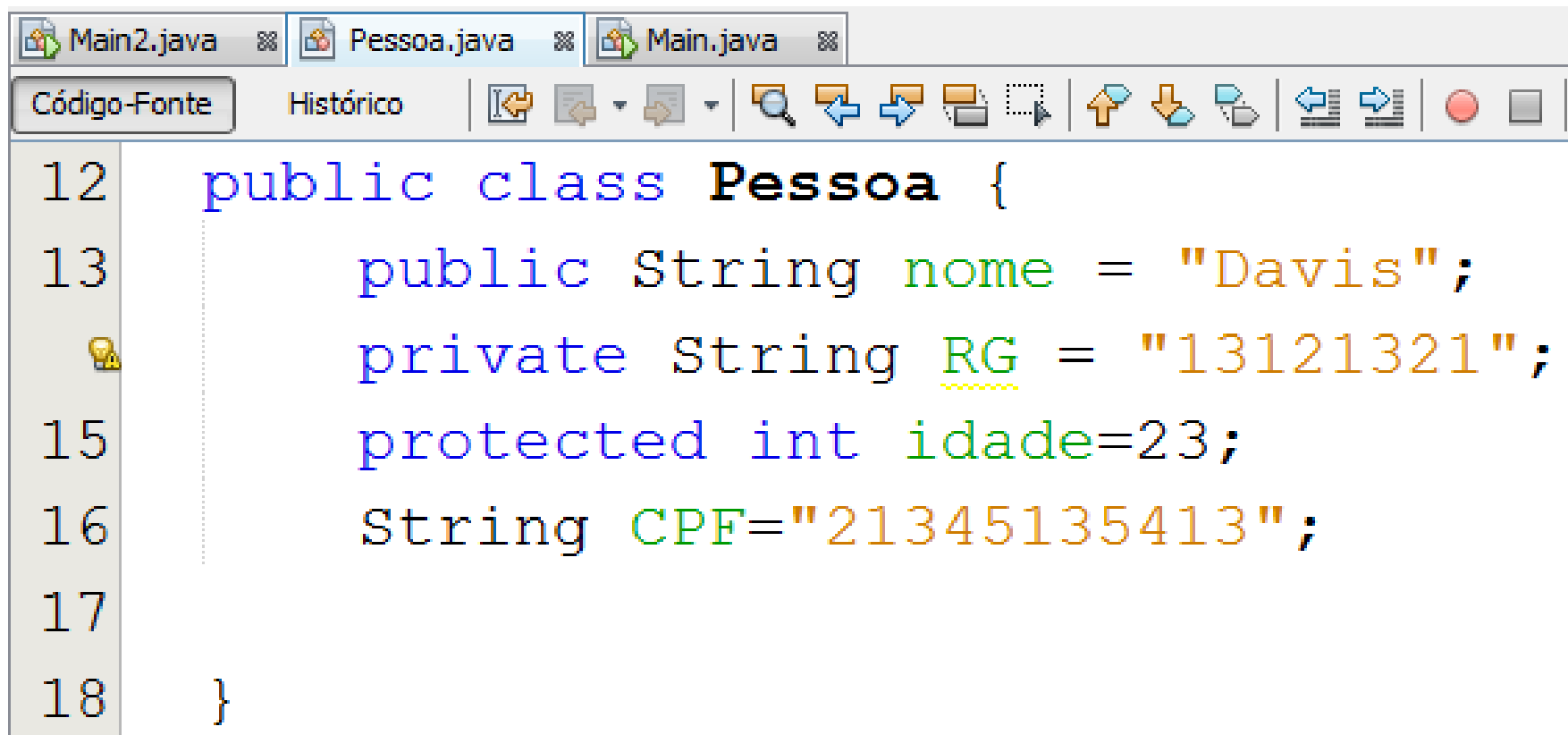
## 7. Encapsulamento

**Exemplo de Encapsulamento:** Criar o pacote **exemplo** com as classes **Pessoas** e **Main2**. Criar o pacote **main** com a classe **Main**.



## 7. Encapsulamento

Na classe **Pessoa** criar os atributos **nome** (public), **RG** (private), **idade** (protected) e **CPF** (sem modificador).

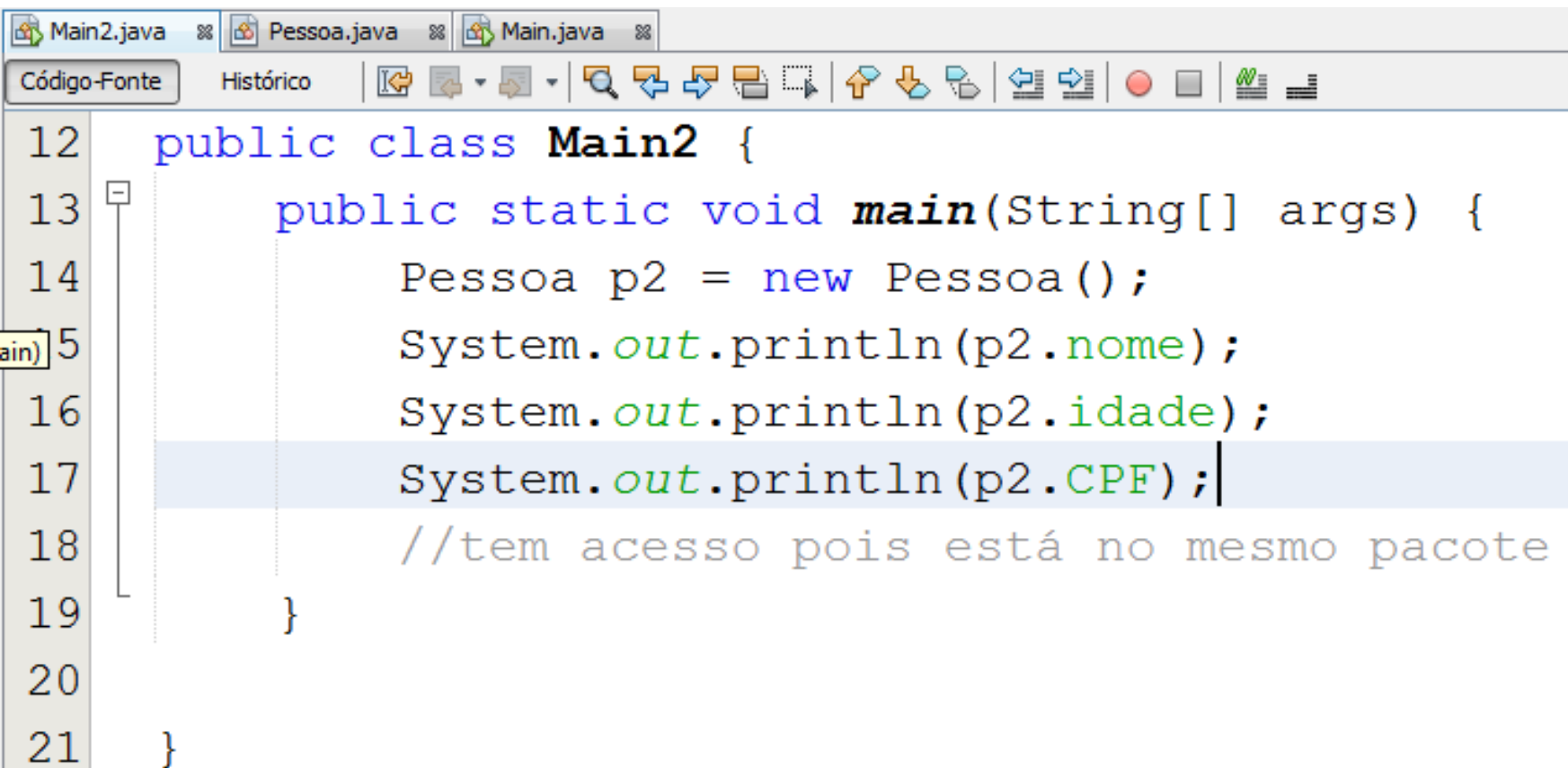


The screenshot shows an IDE window with three tabs: Main2.java, Pessoa.java (active), and Main.java. The active tab displays the source code for the Pessoa class. The code defines four attributes: nome (public String), RG (private String), idade (protected int), and CPF (String). The code is as follows:

```
12 public class Pessoa {
13 public String nome = "Davis";
14 private String RG = "13121321";
15 protected int idade=23;
16 String CPF="21345135413";
17
18 }
```

## 7. Encapsulamento

Na classe **Main2** instanciar o objeto da classe **pessoa** e chamar os atributos.

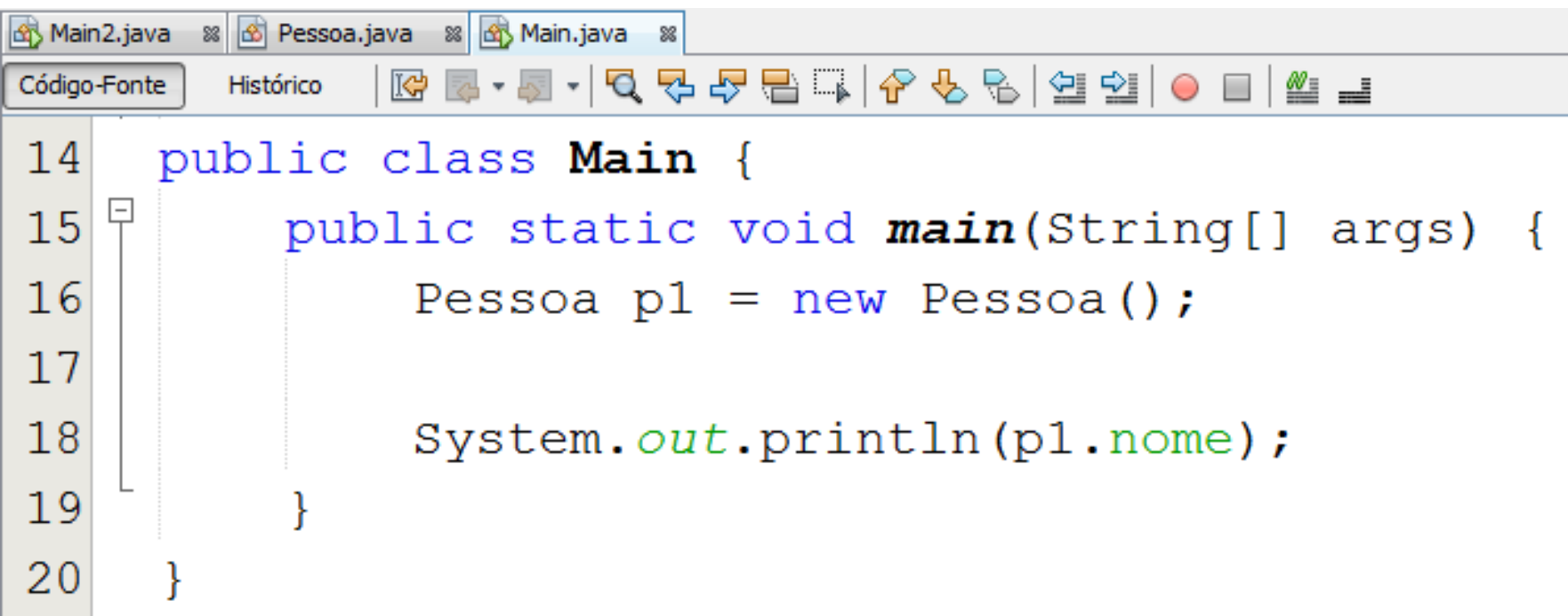


```
12 public class Main2 {
13 public static void main(String[] args) {
14 Pessoa p2 = new Pessoa();
15 System.out.println(p2.nome);
16 System.out.println(p2.idade);
17 System.out.println(p2.CPF);
18 //tem acesso pois está no mesmo pacote
19 }
20
21 }
```

**Veja:** que não é possível chamar o atributo **RG**, pois o mesmo é do tipo **private** e está em outra classe.

## 7. Encapsulamento

Na classe **Main** instanciar o objeto da classe **pessoa** e chamar os atributos.



```
14 public class Main {
15 public static void main(String[] args) {
16 Pessoa p1 = new Pessoa();
17
18 System.out.println(p1.nome);
19 }
20 }
```

**Veja:** só é possível chamar o atributo **nome**, pois ele está em outro pacote e é o único do tipo public.



## 7. Encapsulamento

Executando o **Main2** repare que os atributos **nome**, **idade** e **CPF** aparecem na saída.

⋮ Saída - GPJ - Aula 2 - Ex 4 - Modificador de acesso (run)



run:



Davis



23



21345135413

CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)

|

## 7. Encapsulamento

Executando o **Main** repare que só o atributo **nome** que aparece na saída.

⋮ Saída - GPJ - Aula 2 - Ex 4 - Modificador de acesso (run)



run:

Davis

CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)

|

## 7. Encapsulamento

### Exercícios:

Criar um programa com o pacote **saudacao** com as classes **Saudacao** (com os métodos **introducao()**, **dialogo()** e **despedida()**) e **Apresentacao** com o método **apresentar()**. Criar o pacote **main** com Classe **Main** que só terá acesso ao método **apresentar()**.

## 7. Encapsulamento

### Exercícios:

Criar um programa de uma Fabrica onde terá as etapas de produzir um carro. Teremos os métodos **produzirCarro()**, **construirBase()**, **colocarComponentes()**, **pintarCarro()** e **despacharCarro()**. O único método visível no Main será o **produzirCarro()**.

## 7. Encapsulamento

### Exercícios:

Criar um programa para calcular compra no Exterior, no qual você entrará com o valor do produto e os métodos **taxaImportação(+20%)**, **taxaFrete(+35)** e **taxaSeguro(+5%)** farão a conta, somando suas devidas taxas respectivamente.

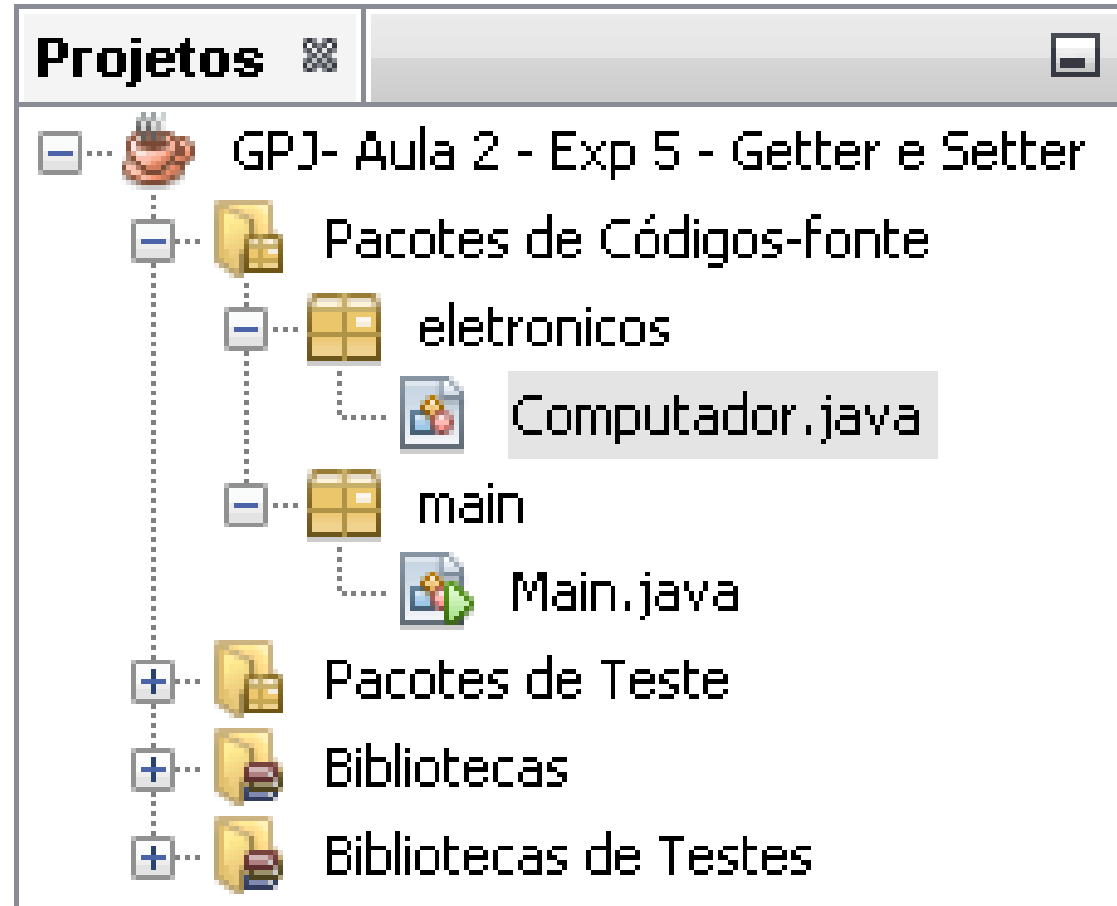
## 8. Getter & Setter

- Para burlar atributos do tipo private utilizamos o Getter & Setter.
- Atalho para chamar o Getter & Setter -> **Alt + Insert**
- O acesso ao atributo é efetuado através de método, mesmo que ele seja do tipo private.
- Usa-se get para pegar o valor e set para definir o valor.

## 8. Getter & Setter

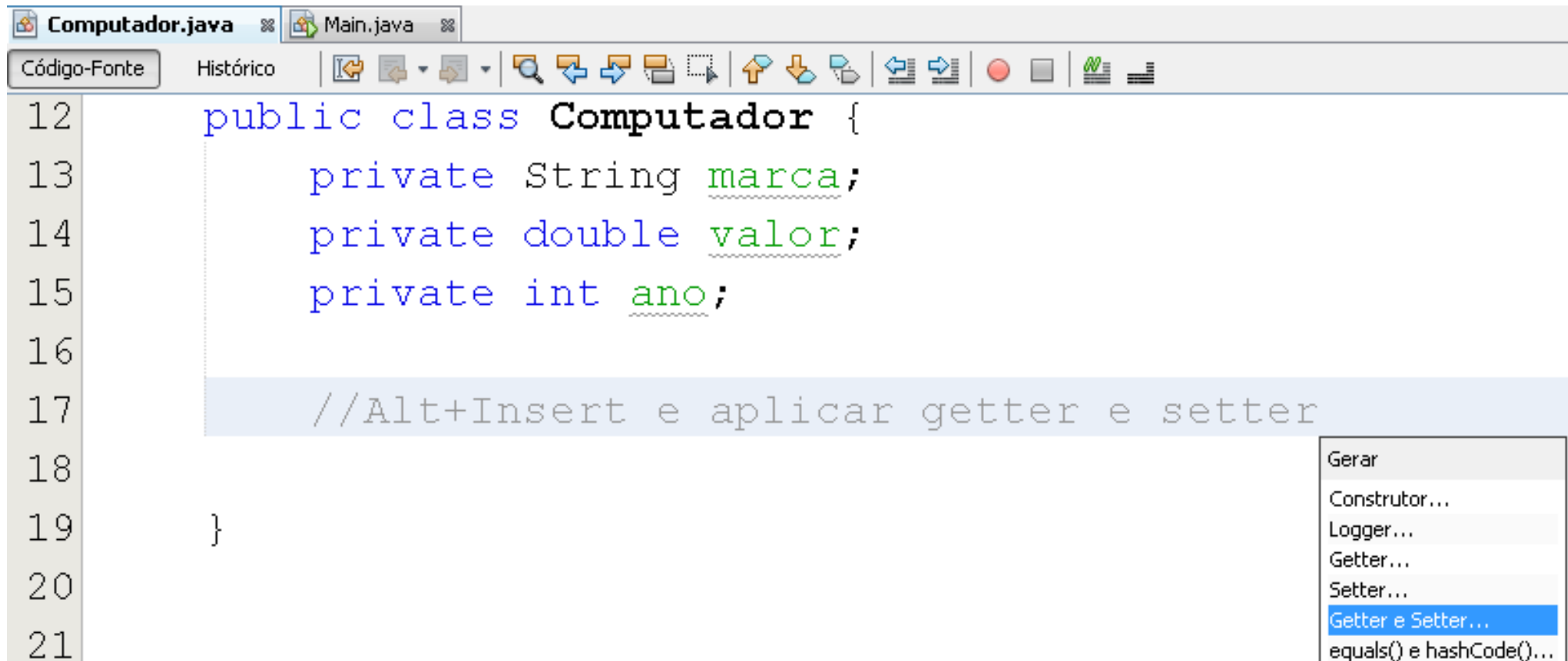
Exemplo de Getter & Setter:

Criar o pacote **eletrônicos**  
com a classe **Computador**.  
Criar o pacote **main** com a  
classe **Main**.



## 8. Getter & Setter

Na classe Computador, criar os atributos **marca**, **valor** e **ano**, depois criar os métodos **Getter** e **Setter** .



The screenshot shows an IDE window with two tabs: 'Computador.java' and 'Main.java'. The 'Código-Fonte' (Source Code) tab is active. The code in 'Computador.java' is as follows:

```
12 public class Computador {
13 private String marca;
14 private double valor;
15 private int ano;
16
17 //Alt+Insert e aplicar getter e setter
18
19 }
20
21
```

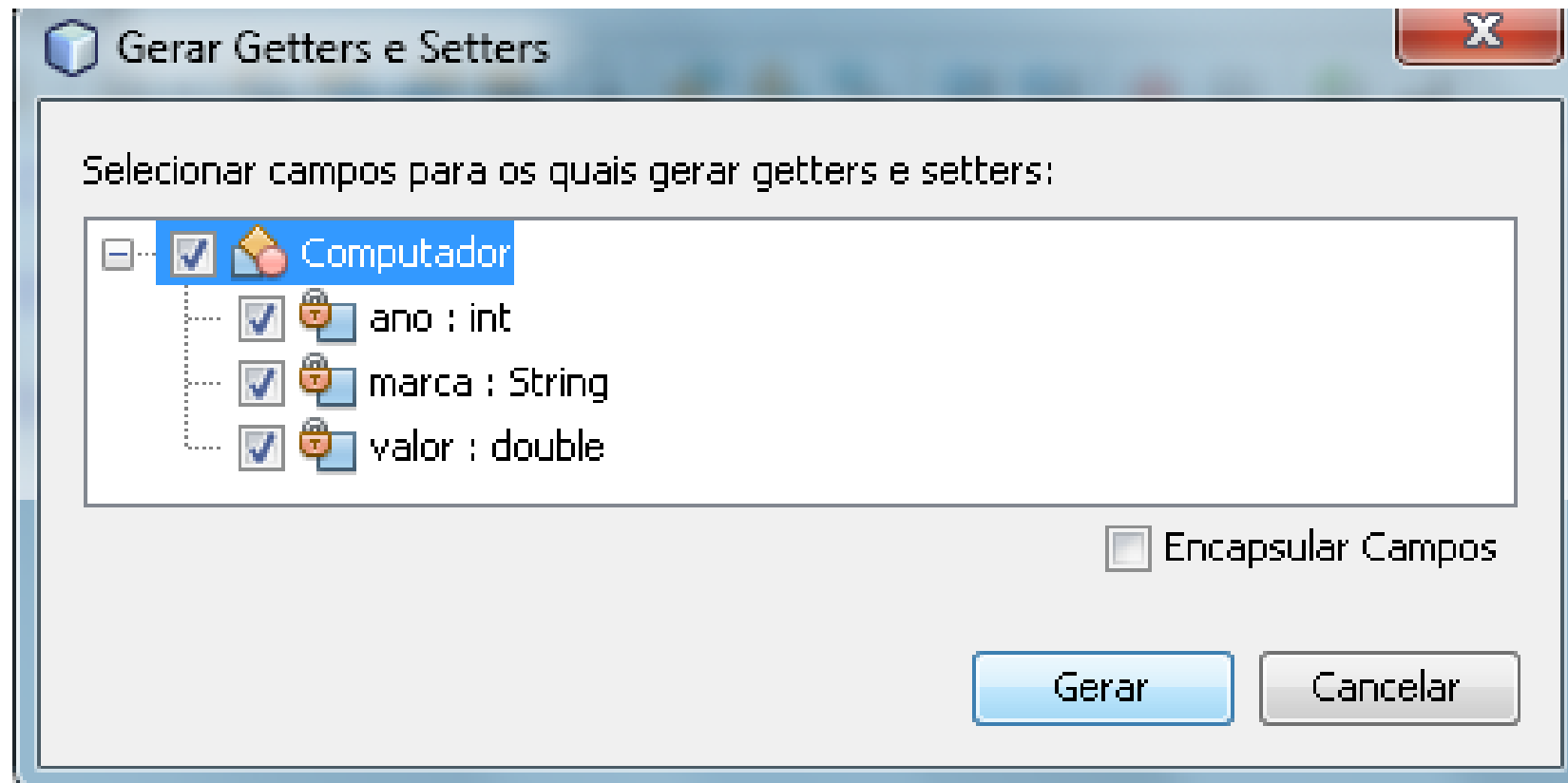
A context menu is open over the code, showing the following options:

- Gerar
- Construtor...
- Logger...
- Getter...
- Setter...
- Getter e Setter...** (highlighted)
- equals() e hashCode()...



## 8. Getter & Setter

Selecionar todas as opções e clicar em “Gerar” para que seja gerado os métodos nos atributos.



## 8. Getter & Setter

Note que foi gerado um método para pegar (get) o valor e outro método para atribuir (set) o valor para cada atributo criado.

```
public String getMarca() {
 return marca;
}

public void setMarca(String marca) {
 this.marca = marca;
}

public double getValor() {
 return valor;
}

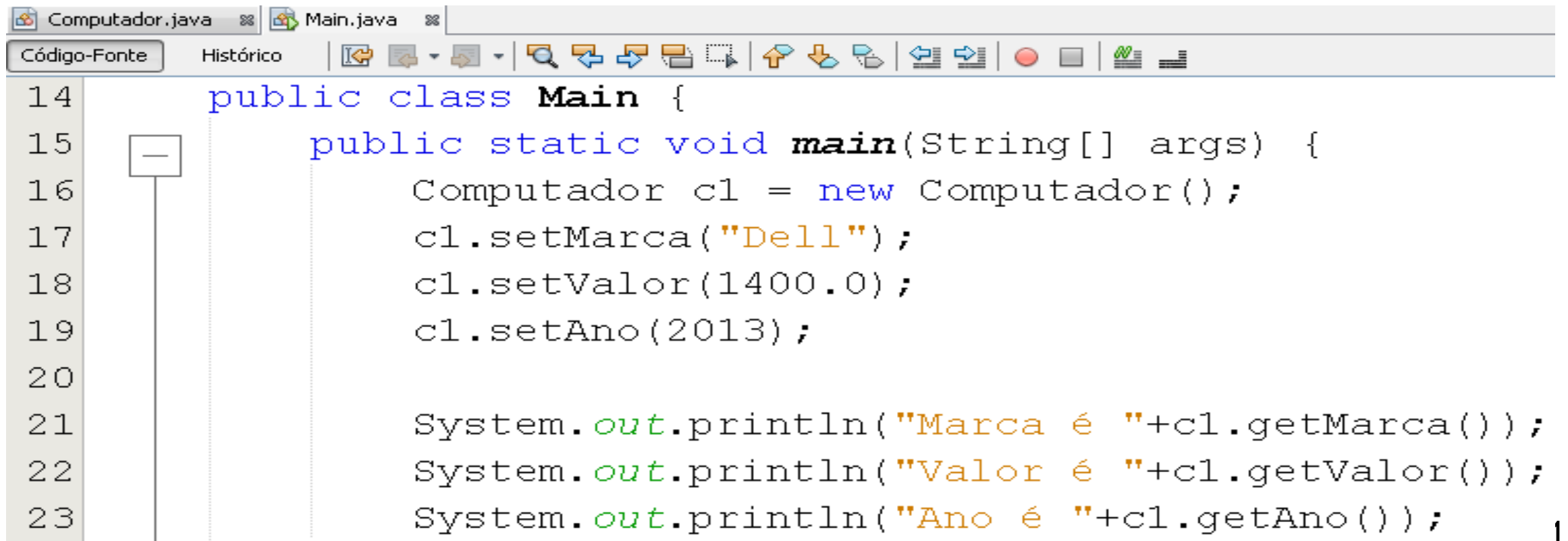
public void setValor(double valor) {
 this.valor = valor;
}

public int getAno() {
 return ano;
}

public void setAno(int ano) {
 this.ano = ano;
}
```

## 8. Getter & Setter

Na classe **Main** instanciar o objeto e atribuir valores (set) para cada Atributo. Pedir para mostrar os valores (get).



```
Computador.java Main.java
Código-Fonte Histórico
14 public class Main {
15 public static void main(String[] args) {
16 Computador c1 = new Computador();
17 c1.setMarca("Dell");
18 c1.setValor(1400.0);
19 c1.setAno(2013);
20
21 System.out.println("Marca é "+c1.getMarca());
22 System.out.println("Valor é "+c1.getValor());
23 System.out.println("Ano é "+c1.getAno());
1
```

## 8. Getter & Setter

Ao executar o programa, repare que os valores atribuídos são mostrados, mesmo que o atributo seja do tipo private.

**❯ Saída - GPJ- Aula 2 - Exp 5 - Getter e Setter (run) #2**



run:

Marca é Dell

Valor é 1400.0

Ano é 2013

CONSTRUÍDO COM SUCESSO (tempo total: 3 segundos)

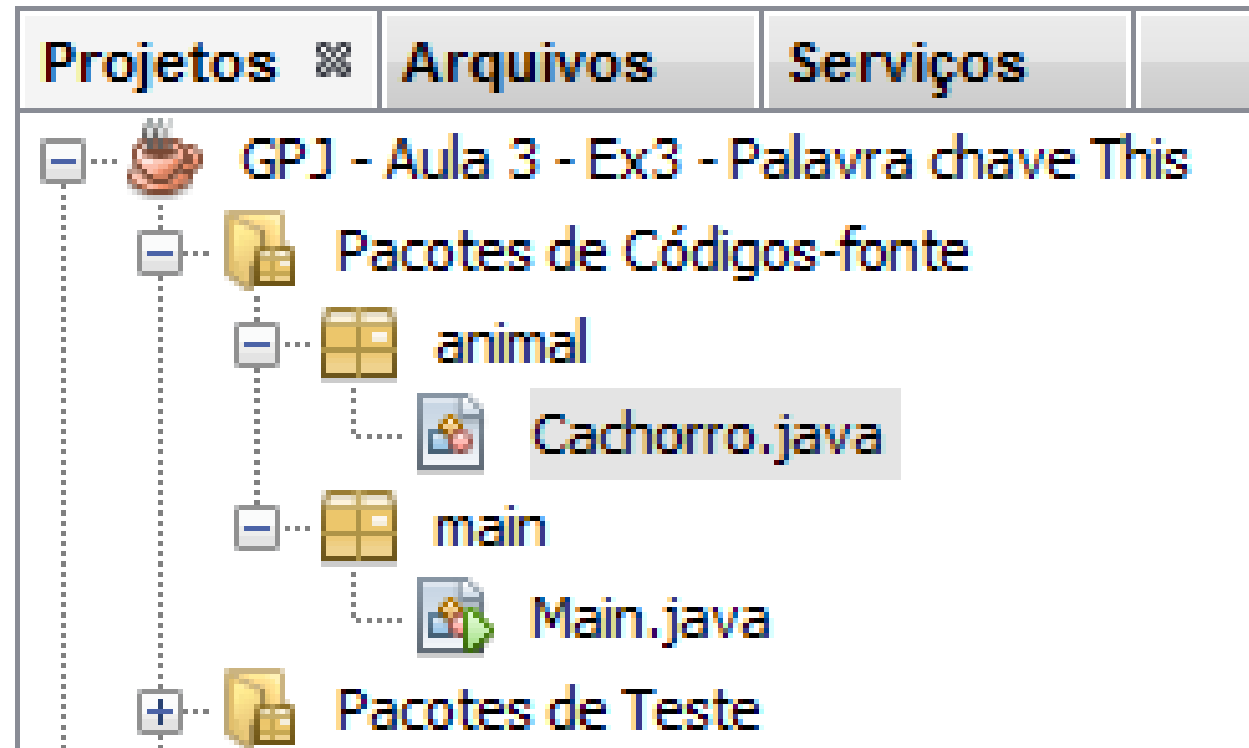
## 9. Super / This

### This

- Utilizado para acessar os métodos da própria Classe.
- Muito usado para diferenciar o atributo do argumento do método.
- Por default(padrão), se não colocar nada, os métodos e os atributos são *this*  
`this.método()`

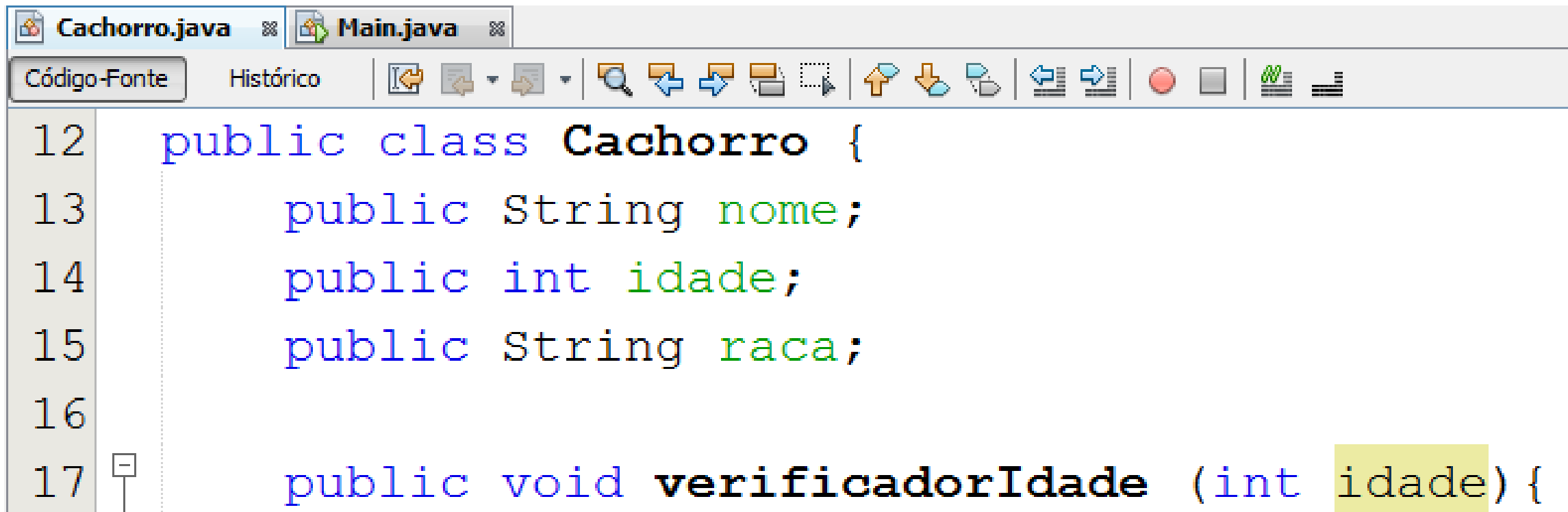
## 9. Super / This

**Exemplo de this:** Criar o pacote **animal** com a classe **Cachorro** e o pacote **main** e a classe **Main**.



## 9. Super / This

Na classe **Cachorro** criar os atributos **nome**, **idade** e **raca** e o método **verificadorIdade**.



```
Cachorro.java Main.java
Código-Fonte Histórico
12 public class Cachorro {
13 public String nome;
14 public int idade;
15 public String raca;
16
17 public void verificadorIdade (int idade) {
```

## 9. Super / This

No método **verificadorIdade** criar uma condicional *if* para saber se o cachorro é velho utilizando o *this*.

```
18 //com o this ele faz em questão ao atributo
19 if (this.idade >10) {
20 System.out.println("Cachorro velho");
21 }
22 else{
23 System.out.println("Cachorro novo");
24 }
```



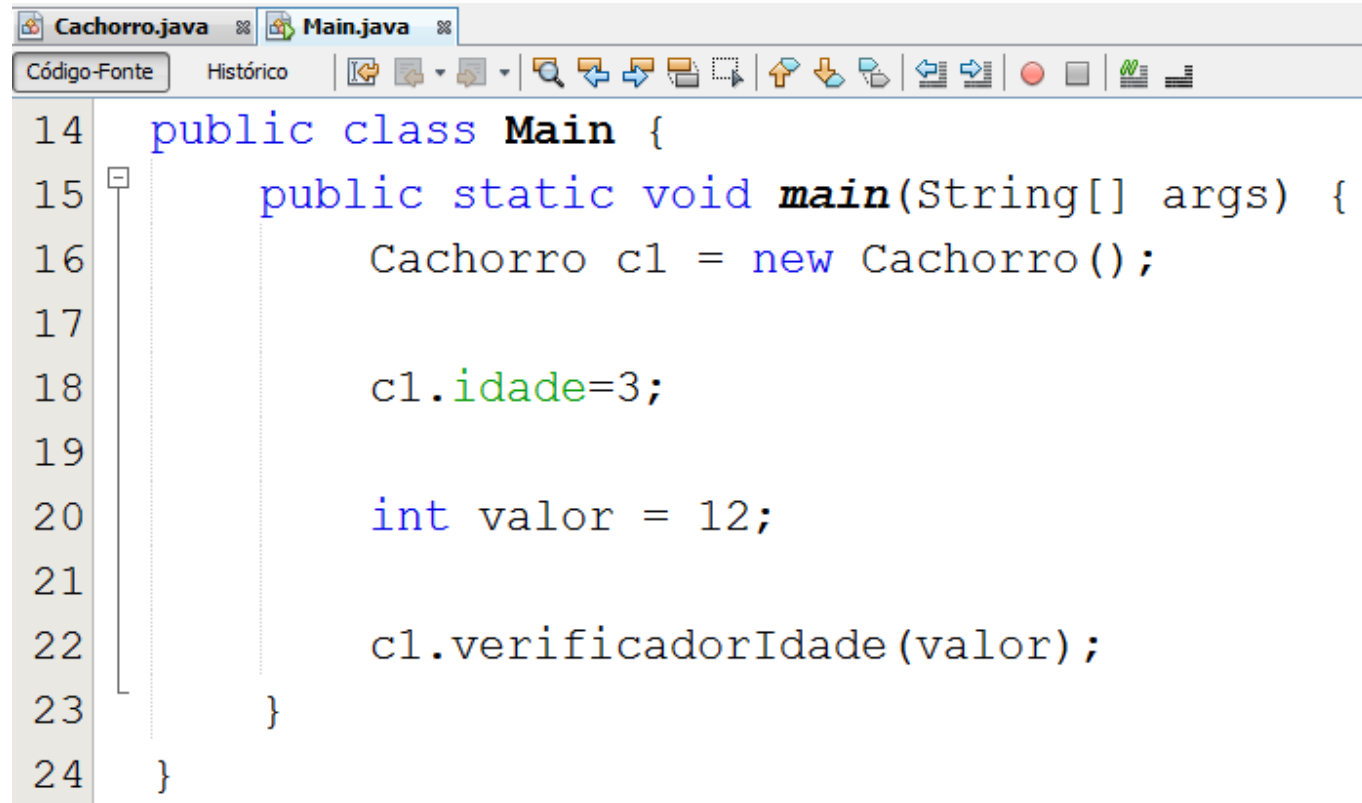
## 9. Super / This

Ainda no método **verificadorIdade** criar outra condicional *if* para saber se o cachorro é velho sem o *this*.

```
25 //Sem o this faz em valor da variável(argumento).
26 if (idade >10) {
27 System.out.println("Cachorro velho");
28 }
29 else {
30 System.out.println("Cachorro novo");
31 }
```

## 9. Super / This

Na classe **Main** instanciar o objeto da classe **Cachorro**, atribuir valor ao método **idade**, criar a variável **valor** e passar a variável como argumento para o método **verificador**.



```
14 public class Main {
15 public static void main(String[] args) {
16 Cachorro c1 = new Cachorro();
17
18 c1.idade=3;
19
20 int valor = 12;
21
22 c1.verificadorIdade(valor);
23 }
24 }
```

## 9. Super / This

Veja o resultado e repare que com o *this* ele compara em relação ao atributo e sem o *this* compara em relação a variável.

⋮ Saída - GPJ - Aula 3 - Ex3 - Palavra chave This (run)



run:



Com this

Cachorro novo



Sem this



Cachorro velho

CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)

## 9. Super / This

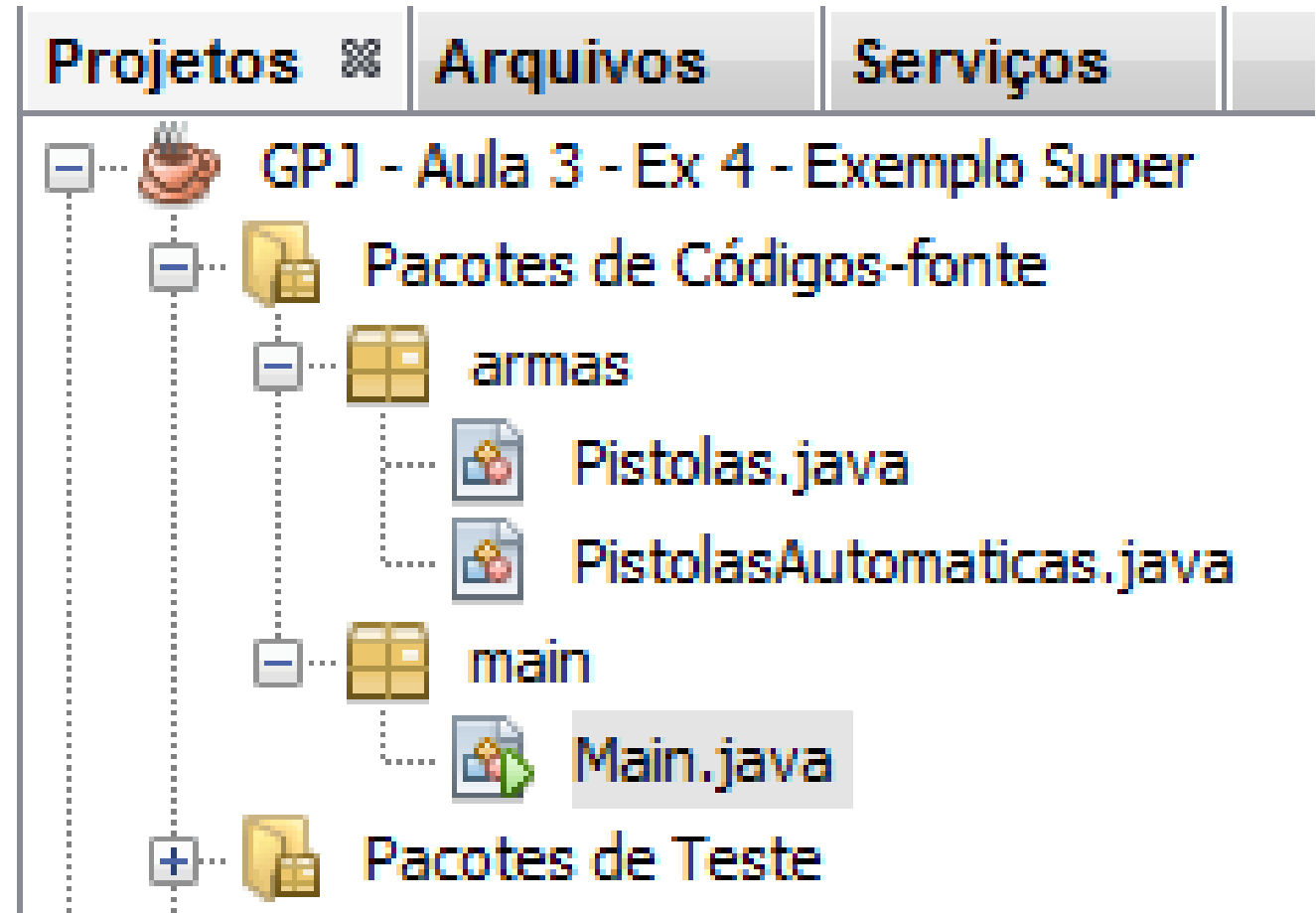
### Super

- Comando de acesso aos métodos da Classe Pai.
- Utilizado em herança para aproveitar os métodos da classe pai e adicionar mais códigos.
- Pode ser utilizado em atributo, porém é mais usual em métodos.

`super.método()`

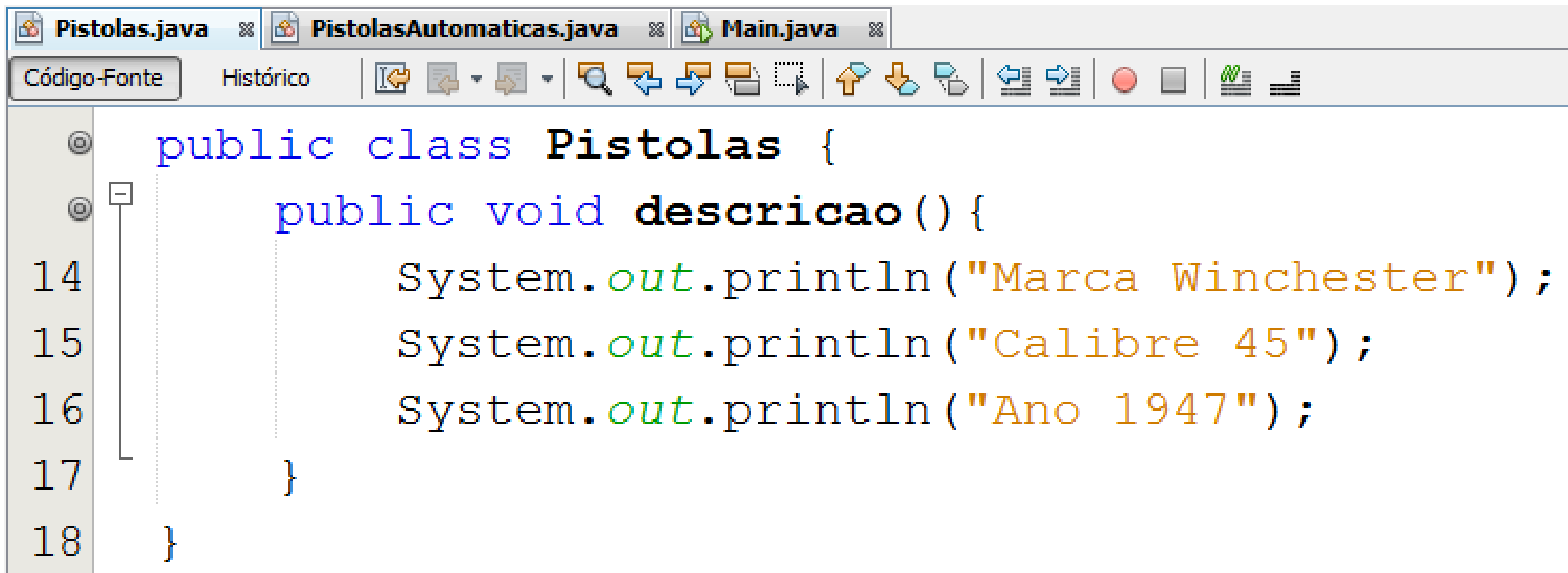
## 9. Super / This

**Exemplo de Super:** Criar o pacote **armas** com as classes **Pistolas** e **PistolasAutomaticas** e o pacote **main** e a classe **Main**.



## 9. Super / This

Na classe **Pistolas** criar o método **descricao()** com a **marca**, **Calibre** e **ano de fabricação**.

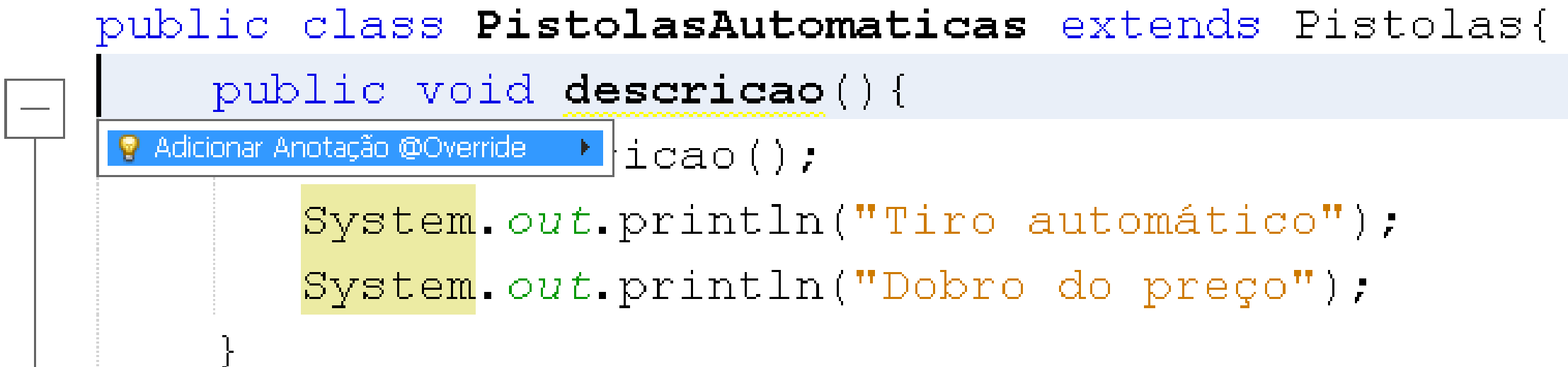


```
Pistolas.java PistolasAutomaticas.java Main.java
Código-Fonte Histórico
public class Pistolas {
 public void descricao() {
 System.out.println("Marca Winchester");
 System.out.println("Calibre 45");
 System.out.println("Ano 1947");
 }
}
```

## 9. Super / This

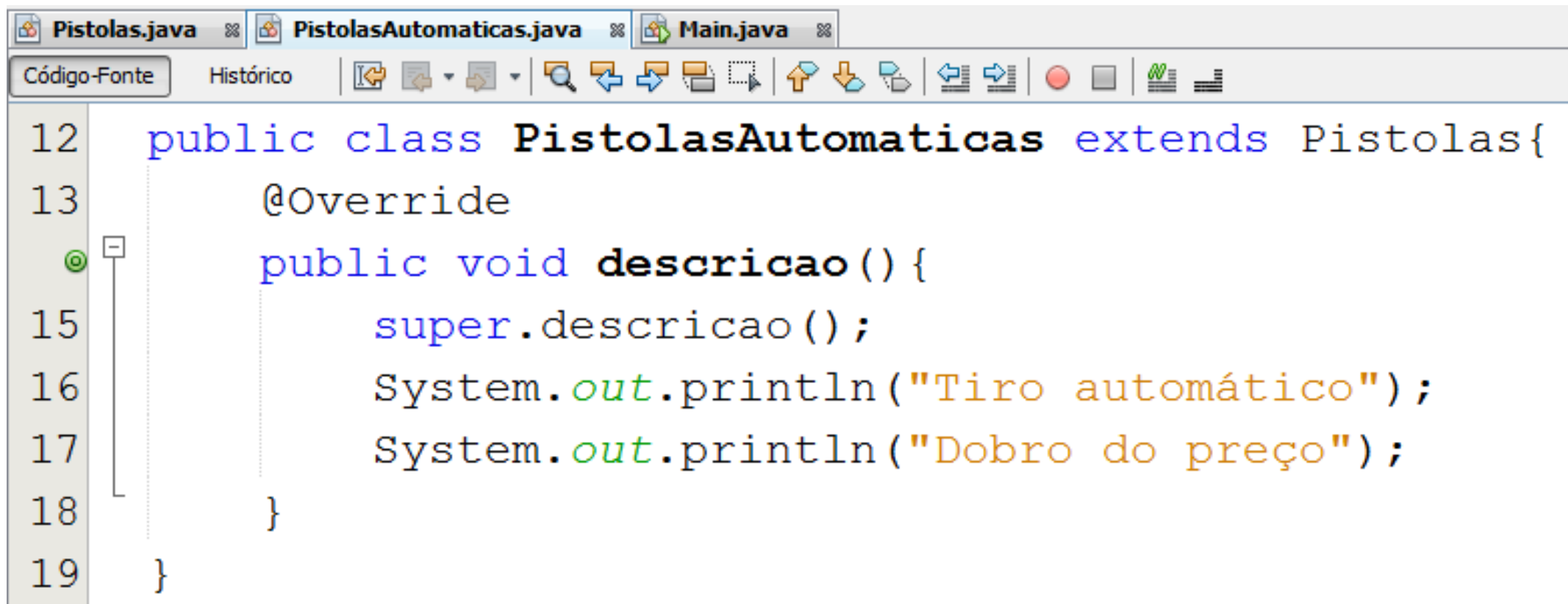
Colocar a classe **PistolasAutomaticas** como filha da classe **Pistolas**, criar o método **descricao()** e adicionar **@Override**.

```
14 public class PistolasAutomaticas extends Pistolas{
15 public void descricao() {
16 System.out.println("Tiro automático");
17 System.out.println("Dobro do preço");
18 }
19 }
```



## 9. Super / This

Após adicionar `@Override`, chamar a palavra chave *super* com o método **descricao** e adicionar **Tiro** e **dobro do preço**.

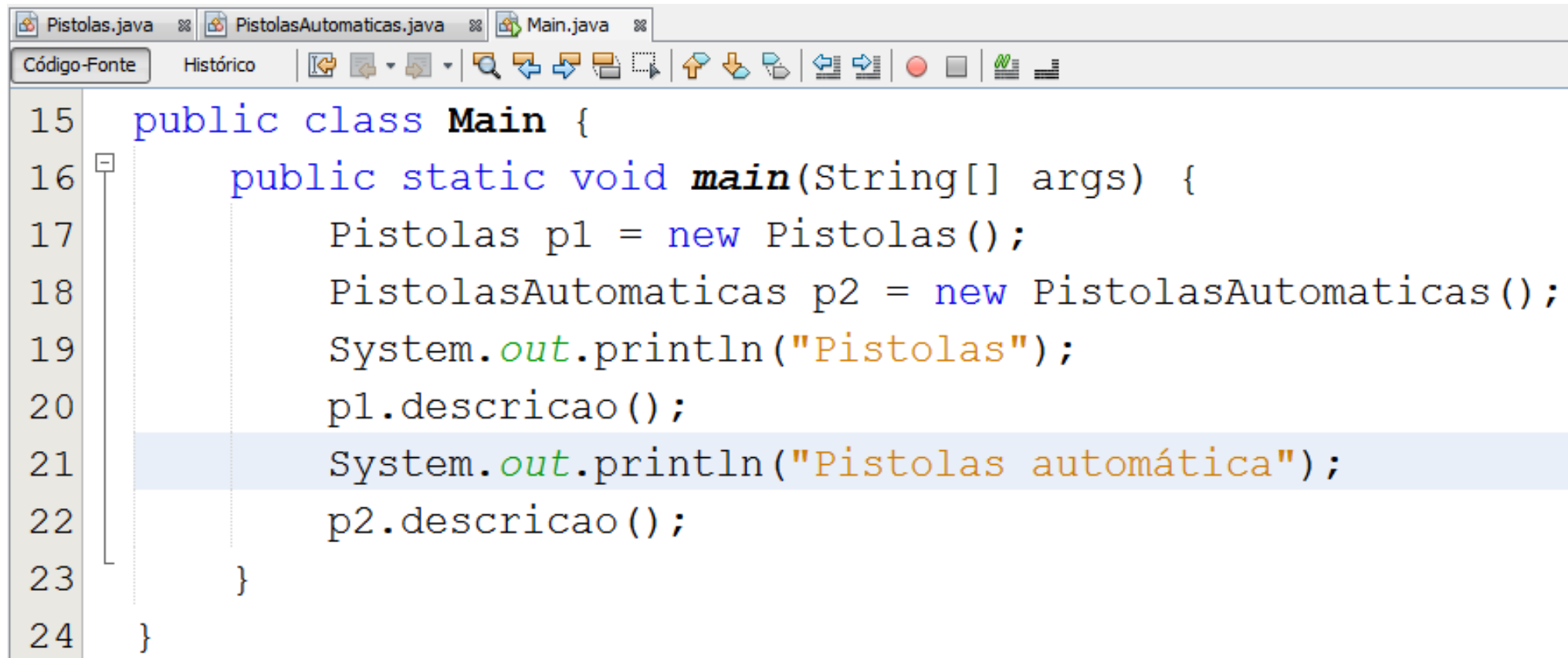


```
12 public class PistolasAutomaticas extends Pistolas{
13 @Override
14 public void descricao() {
15 super.descricao();
16 System.out.println("Tiro automático");
17 System.out.println("Dobro do preço");
18 }
19 }
```



## 9. Super / This

Na classe **Main** instanciar os objetos **Pistolas** e **PistolasAutomaticas** e chamar o método em cada objeto.

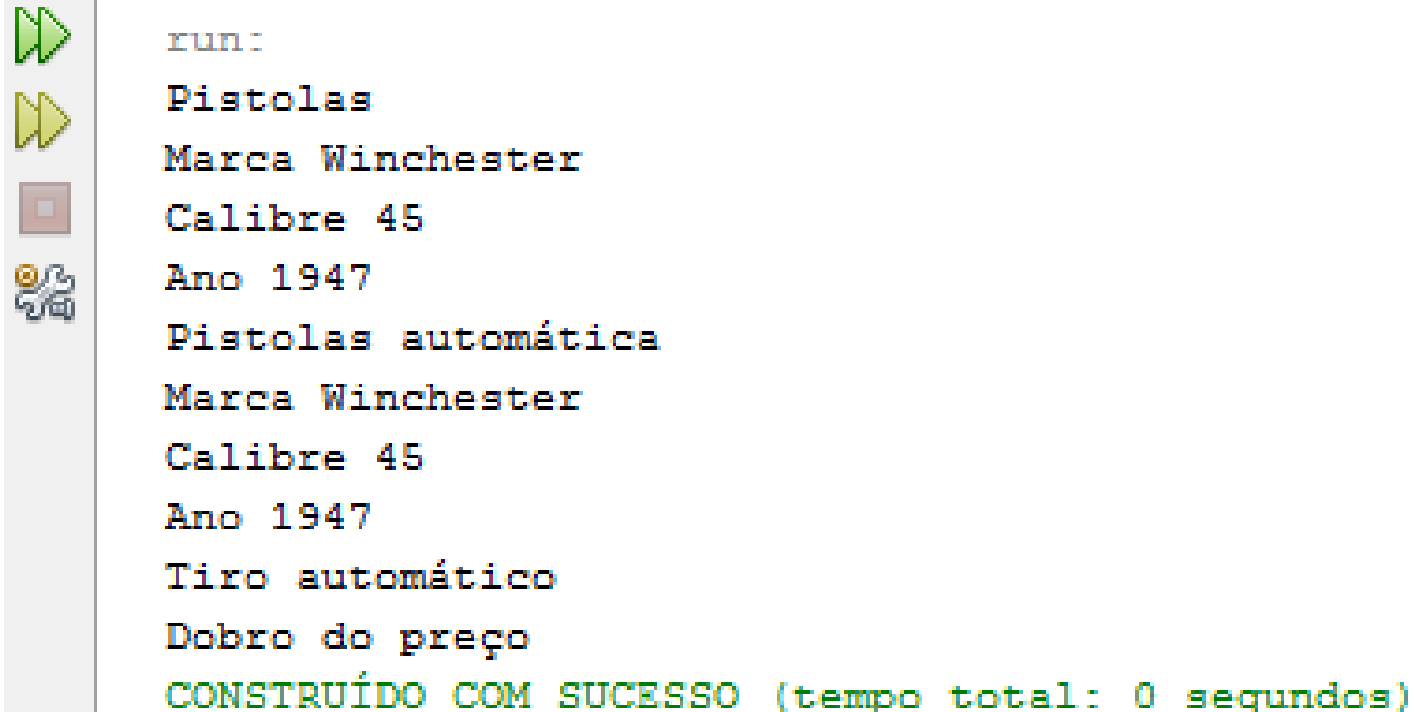
A screenshot of a Java IDE window with three tabs: Pistolas.java, PistolasAutomaticas.java, and Main.java. The 'Main.java' tab is active, showing the source code. The code defines a public class Main with a main method. Inside the main method, two objects are instantiated: Pistolas p1 and PistolasAutomaticas p2. The describe method is called on both objects. The line 'System.out.println("Pistolas automática");' is highlighted in blue. The line numbers 15 through 24 are visible on the left side of the code editor.

```
15 public class Main {
16 public static void main(String[] args) {
17 Pistolas p1 = new Pistolas();
18 PistolasAutomaticas p2 = new PistolasAutomaticas();
19 System.out.println("Pistolas");
20 p1.descricao();
21 System.out.println("Pistolas automática");
22 p2.descricao();
23 }
24 }
```

## 9. Super / This

Veja que em “Pistolas automáticas” ele herdou o método da classe “Pistolas” e ainda acrescentou o tipo de tiro e o preço.

⋮ Saída - GPJ - Aula 3 - Ex 4 - Exemplo Super (run)



The image shows a screenshot of an IDE's output window. On the left side, there is a vertical toolbar with four icons: a green double arrow pointing right, a yellow double arrow pointing right, a red square, and a blue icon with a gear and a star. The main area of the window displays the output of a program run. The text is as follows:

```
run:
Pistolas
Marca Winchester
Calibre 45
Ano 1947
Pistolas automática
Marca Winchester
Calibre 45
Ano 1947
Tiro automático
Dobro do preço
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```