**Sintaxe JAVA**

Alguns conceitos importantes:

* A base da linguagem JAVA

POO

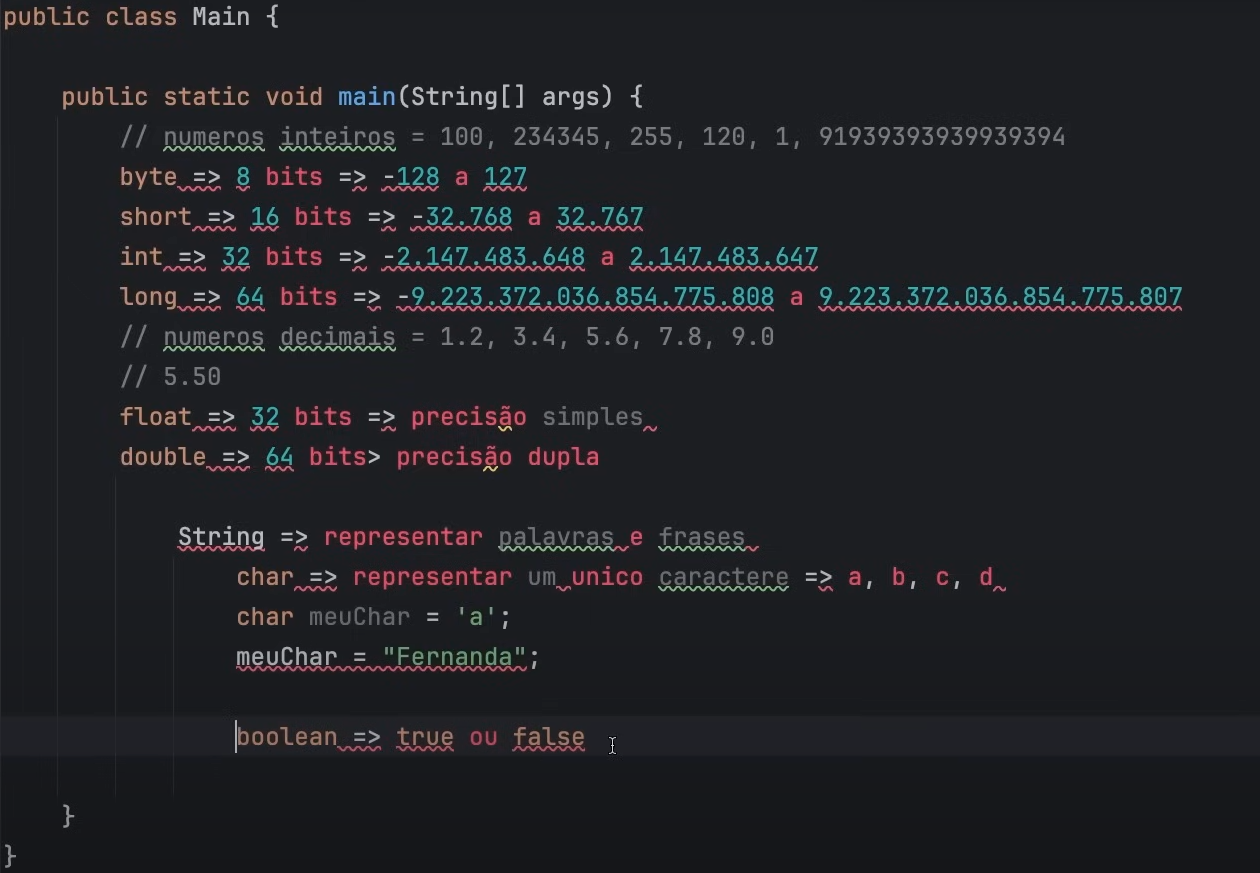
* Linguagem fortemente tipada, ou seja, linguagem que é construída com boas práticas e normas à risca para que seja tudo bem desenvolvido. Ao longo da execução, o tipo da execução não pode ser mudado, fazendo com que cada variável seja declarada exclusivamente para uma função específica.
* Uma linguagem independente de plataforma usando um Bytecode (Portátil) através da JVM.

**Sintaxe básica**

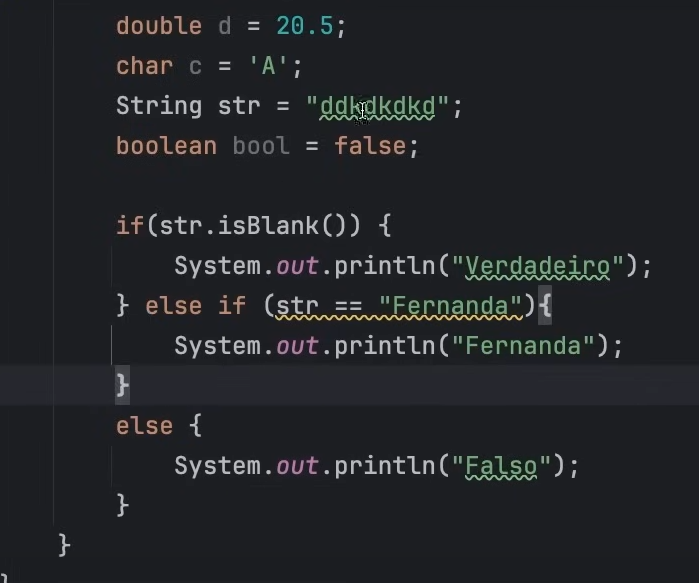
* variáveis são definidas da seguinte maneira em Java:
* public class Main {
* int minhaidade = 24;
* String meuNome = "Leonardo";
* }
* No caso: “tipo nome\_da\_variável = valor;”

Alguns destes tipo são: int, string, var, bool. No caso de “var” é usado para que a JVM identifique automaticamente o tipo da variável e a transforme no tipo correto para o programa.

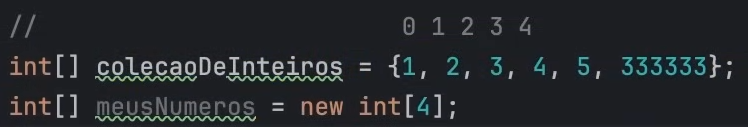
**Tipos primitivos**

****

**Condicionais**



**Vetores**

****

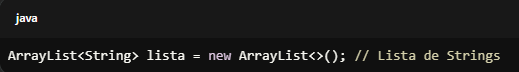
1. Estrutura de Dados: Vetores podem armazenar uma coleção de elementos.
2. Acesso por Índice: Permite acesso aos elementos por meio de índices, começando em 0.
3. Em Java, um vetor é mutável, o que significa que você pode alterar os elementos após a criação, mas não pode mudar o tamanho do vetor.
4. Todos os elementos devem ser do mesmo tipo.
5. Geralmente usado para armazenar coleções homogêneas de dados onde o tamanho é conhecido.

**Arraylists (Classe)**

O ArrayList em Java é uma das classes da coleção que oferece uma implementação dinâmica de uma lista. Aqui estão os principais pontos sobre como funciona:

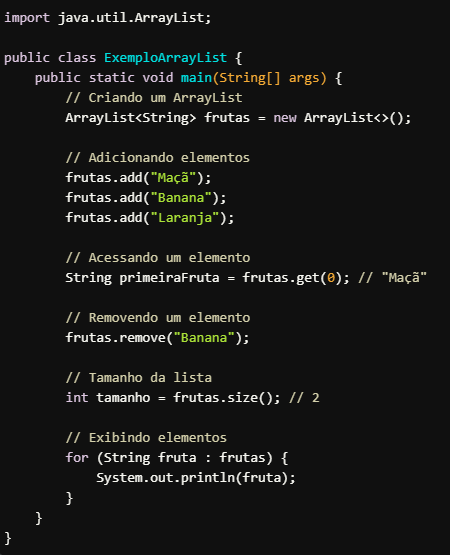
**Estrutura de Dados:** O ArrayList é uma lista que pode crescer e encolher conforme necessário. Ele permite armazenar elementos de forma dinâmica.

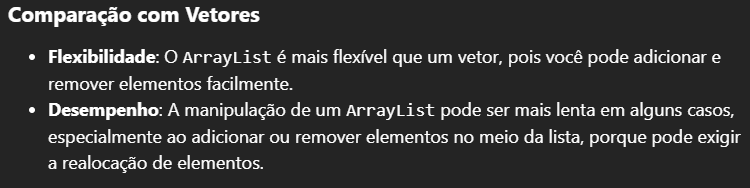
* Ao contrário de um vetor, o ArrayList pode alterar seu tamanho automaticamente conforme você adiciona ou remove elementos.
* Por padrão, o ArrayList pode armazenar objetos de qualquer tipo. A partir do Java 5, você pode usar generics para especificar o tipo de dados que o ArrayList vai armazenar.

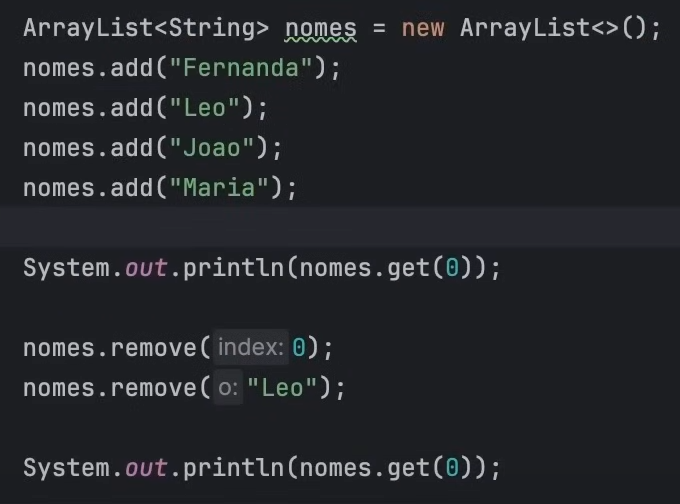


* Você pode acessar elementos usando índices, assim como em um vetor.
* O ArrayList oferece métodos prontos para adicionar, remover e pesquisar elementos, como add(), remove(), get(), entre outros.

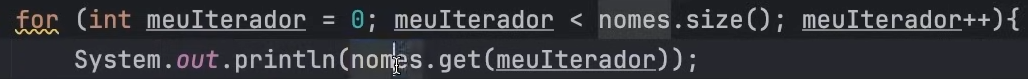
Aqui está um exemplo básico de como usar um ArrayList em Java:

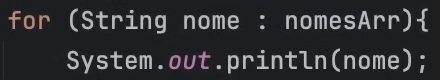






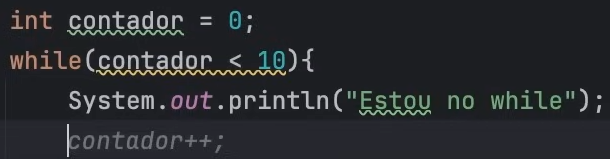
**Loops**

Quando se tem uma noção do tamanho de um vetor, se usa o loop o “for” para imprimir todos os resultados presentes na “lista”. 



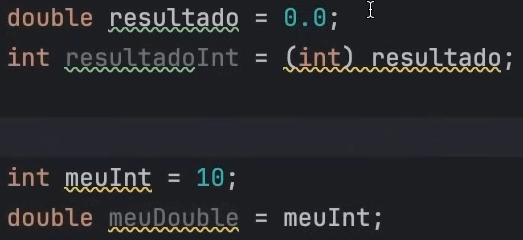
Essas são duas maneiras de se fazer o loop for em JAVA.

Também se usa o loop while que é pra se ter loops infinitos até que o resultado esperado seja alcançado que é através do while.



**Casting**

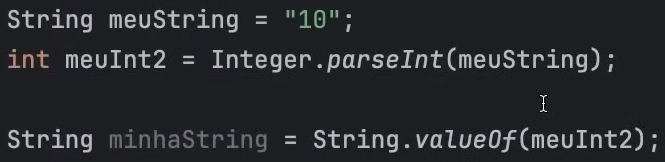
Basicamente o casting é a alteração do tipo da variável, então se você precisa que uma string se torne inteiro por exemplo, você precisa usar o conceito de casting para criar uma variável nova com o tipo esperado.



Casting implícito e explícito.

No caso de strings se usa métodos para transformar de string >> int ou

de int >> String.



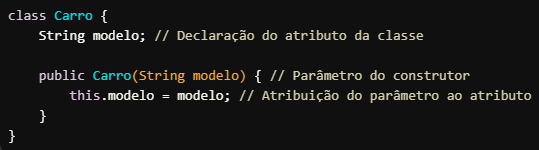
Há casos onde a lógica deverá ser um pouco mais elaborada para fazer a separação correta e mutação de variável para outro tipo esperado.

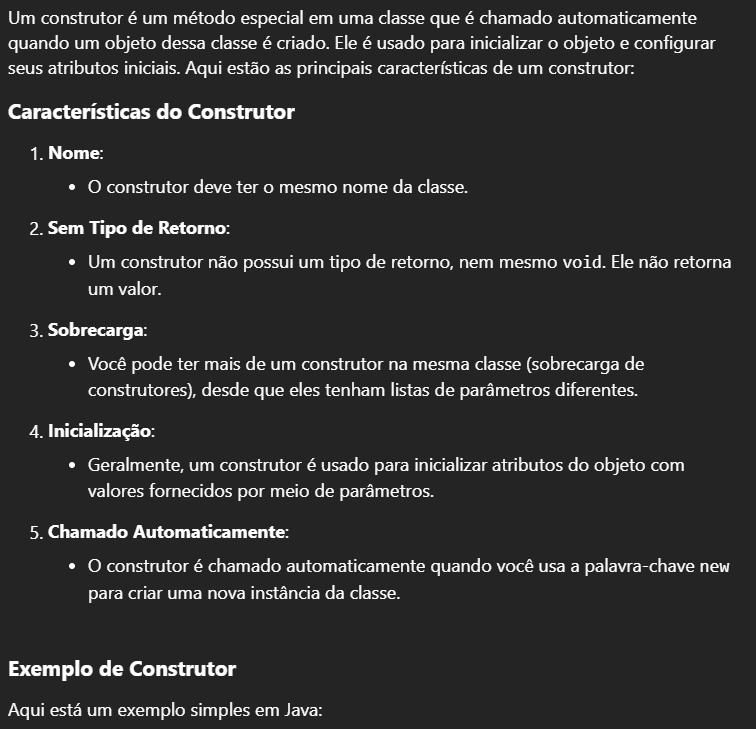
**Compilando em JAVA**

Basicamente para compilar um programa em JAVA é necessário você abrir o terminal e executar os seguintes comando: “javaC ‘Main’.java” o que fará com que seu programa seja compilado e para ser executado você digita “java ‘Main’”, e assim a JVM vai colocar as funcionalidades do algoritmo em prática.

**POO - Programação Orientada a Objeto**

**Classes, construtores, métodos e objetos**



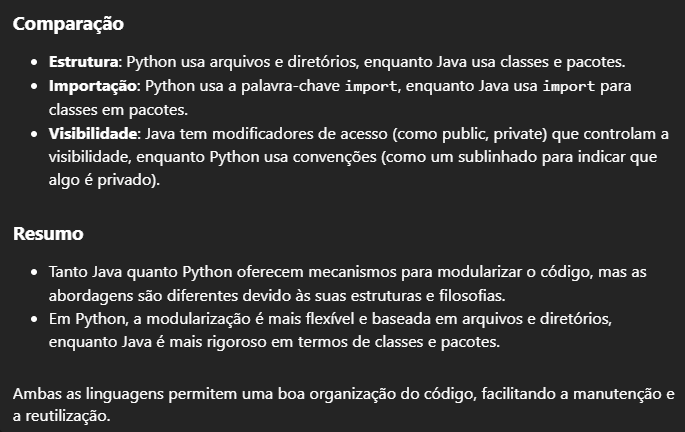




Curiosidade: Métodos assim como funções podem ou não retornar algum valor.

**Modularização**



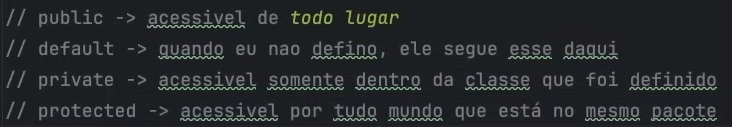


**Modificadores de acesso**

Para que uma classe seja “visível” de qualquer parte nos diretórios de um pacote em JAVA basta declará-lo como **“public”**, se ao invés disso a classe for escrita como “class Main” ela será apenas uma classe destinada a ser usada dentro daquele pacote em específico **(private classe** ou **classe default**). Também tem o modificador protected que é usado para ser usado para dentro da própria classe, dentro das subclasses que estão dentro de um mesmo arquivo e dentro de outras classes que estão dentro de um mesmo pacote, ou seja, fora do pacote esse tipo de modificador não permite ser visível para acesso.

Modificadores também podem ser usados em métodos.

OBS: não pode se ter 2 classes públicas dentro de um mesmo arquivo JAVA.



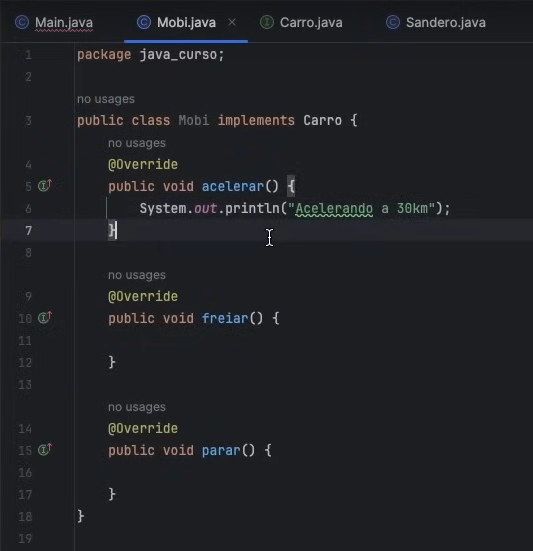
**Interfaces e classes abstratas**

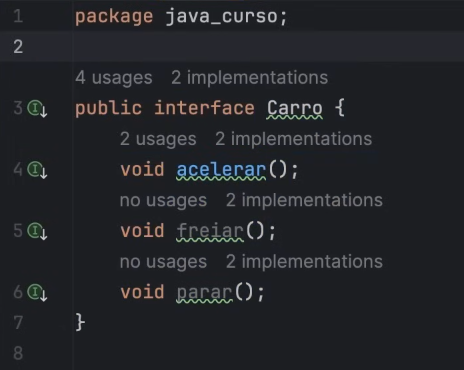
**Interface:** Define a carcaça de uma classe e mostra ao usuário como aquela classe deve ser mostrada visualmente.

A interface define como uma classe vai ter que seguir “normas” e como ela vai ter que se comportar diante da “função” em que foi designada no projeto.

Para implementar uma interface em uma classe, basta usar “implements” antes de declarar as funcionalidades.

**OBS: “@Override” significa que as funções tem um “contrato”, ou seja, estão definidas em uma interface para que sejam seguidas da maneira que deve ser.**



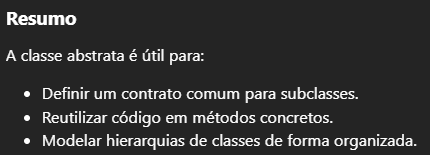


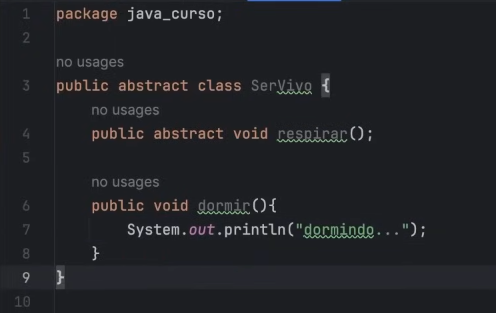
**Classe abstrata**

Uma classe abstrata pode definir um conjunto de métodos que todas as subclasses devem implementar, garantindo que compartilhem uma interface comum.

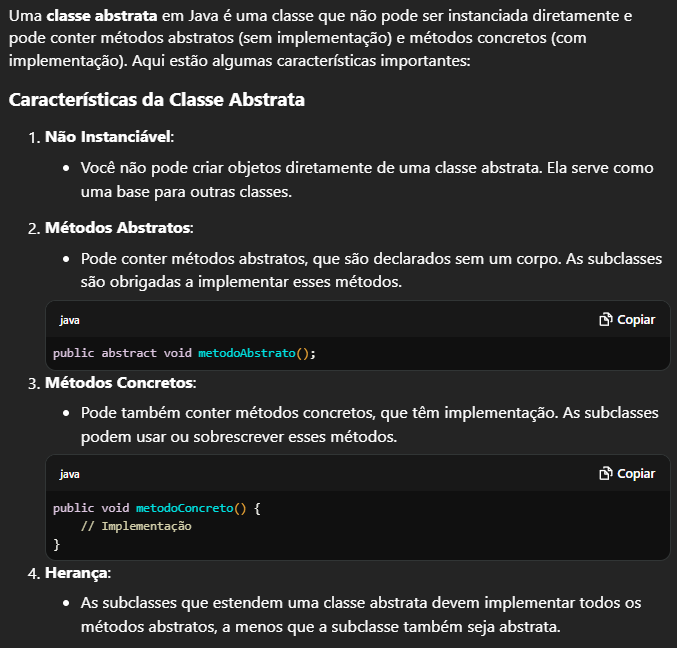
**Vantagens**

* Classes abstratas podem conter métodos concretos que são comuns a todas as subclasses, evitando duplicação de código.
* Facilita a criação de hierarquias de classes que compartilham características comuns, promovendo um design mais limpo e organizado.



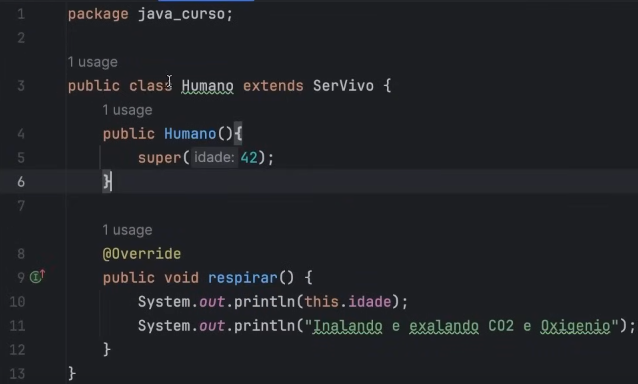






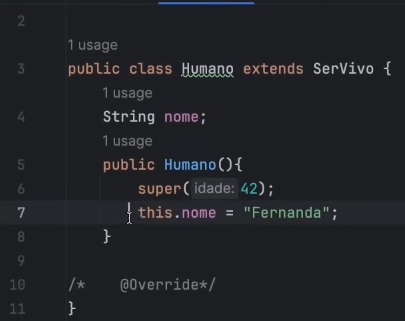
**Herança e polimorfismo**

Basicamente herança é a parte que você usa para estender a funcionalidade de uma classe buscando mais funcionalidades em outras classes.

**Classe: Humano.java**

**Superclasse: SerVivo.java**





“super()” é usado para adicionar um dado que será retornada depois no terminal, este dado está especificado e sendo solicitado dentro da classe humano.

**Polimorfismo**

Basicamente o polimorfismo é um conceito que permite que uma única interface como um método ou uma classe seja usada para apresentar diferentes tipos de ações a objetos.

São objetos dentro de uma classe que são do mesmo tipo mas agem de maneiras diferentes.

**Tratamento de exceções**

O tratamento de exceções em Java é uma maneira de lidar com erros que podem ocorrer durante a execução de um programa. O Java utiliza blocos try, catch e finally para gerenciar exceções. Aqui está um exemplo básico:+

