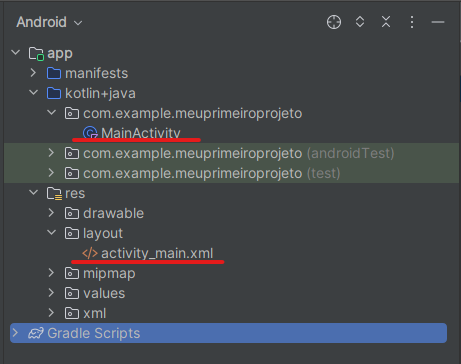
**CURSO DE KOTLIN E JAVA**

Ao iniciar um projeto, caso os arquivos principais não abram, é só acessar essas pastas:

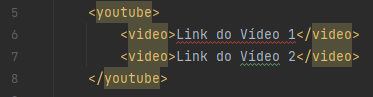


**Estrutura de um projeto**

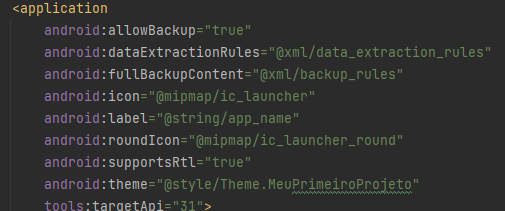
O Gradle Scripts é usado para fazer rotinas e adicionar recursos ao projeto.

O Android Manifest é usado para fazer configurações no aplicativo, já o active\_main é usado para criar a interface do app.

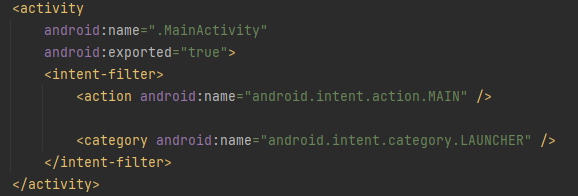
A abertura de uma tag é um símbolo de menor que, tag, símbolo de maior que.  
Ex:



Dentro do Android Manifest é possível pedir permissões para o usuário. Como por Exemplo: Utilizar a câmera ou até pedir a sua localização.

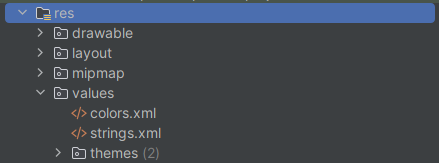


android:icon – É possível mudar qual ícone que irá aparecer no seu app.  
android:label – É o nome do seu app, ele será exibido na lista de todos os apps do smartphone.  
android:roundIcon – É o formato que seu app ficara. Simplificando seria as bordas do ícone.



O activity é usado para criar uma tela no seu projeto. Com determinadas configurações é possível definir qual será a tela de abertura do app.

**RES**



Essa pasta possui diversos arquivos interessantes, pois com ele é que podemos fazer a nossa personalização do app.

Drawable – Essa pasta é utilizada para alocar as imagens que será utilizada no app.  
Layout – É a pasta que armazena o activity\_main.xml  
Mipmap – É onde são armazenados os vetores que serão utilizados no app.  
Values -> Colors – São armazenadas as cores em formas de variáveis.  
Values -> Strings – São as variáveis, sendo elas textos que podem ser utilizados de uma forma mais simplificada para alterar um texto que está em diversos lugares.  
Themes – É uma formação de cor em várias regiões padrões do app.

**Layouts**

Os Layouts são sempre alinhados ao conto superior esquerdo, caso não sejam ligadas as extremidades da tela ou de elementos.

**Interoperabilidade**

O Java e o Kotlin podem trocar códigos entre si, pois eles têm interoperabilidade. É possível utilizar ações de um em outro, mas é preciso que os códigos sejam de acordo com a linguagem que está sendo utilizada.

**JAVA**

**Algoritmo**É uma sequência finita de ações executáveis que visam obter a solução para um tipo de problema.

**A linguagem Java é utilizada para diversas atividades**

* Programas de Computadores (Desktop e Web)
* Criação de Jogos
* Sistemas Automotivos
* Automação de Residências
* Máquinas Industriais
* TVs, Celulares, Tablets
* E muito mais

**Editores Online de Java**

<https://www.browxy.com/>

<https://www.jdoodle.com/online-java-compiler>

<https://replit.com/languages/java10>

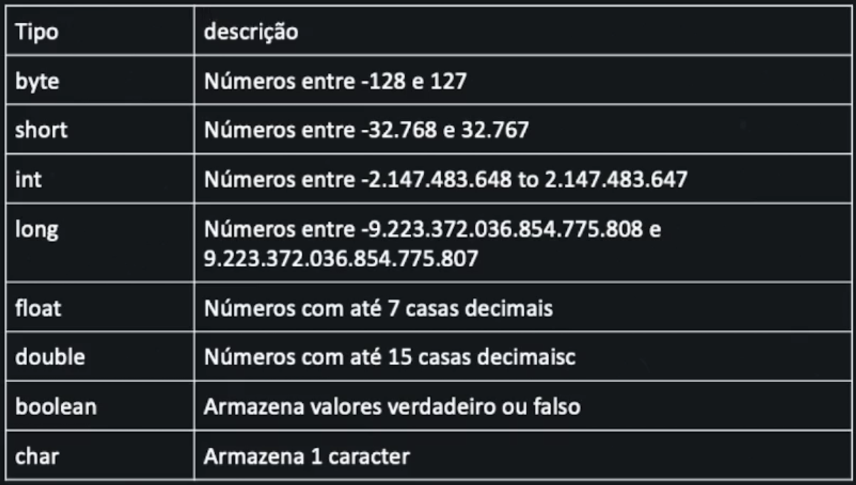
<https://www.w3schools.com/java/java_compiler.asp>

**Variáveis e Constantes**

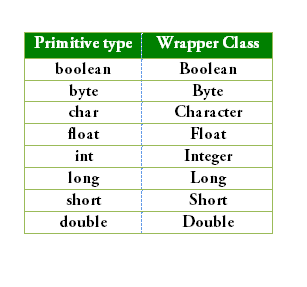
Como definir variável?  
String nomeDaVariavel = Resultado; - Variável String.  
Integer nomeDaVariavel = Resultado; - Variável Inteira.

Como definir constantes?  
final Double Pi = 3,14... – Para definir uma constante, é preciso colocar a palavra final e escolher o tipo. Constantes são normalmente declaradas com as letras maiúscula.

**Tipos de Variáveis (Tipos Primitivos)**



**Classes Wrapper**As Classes Wrapper é utilizada para funções especificas, mas o mais importante é priorizar os tipos primitivos.



**Operadores Aritméticos**

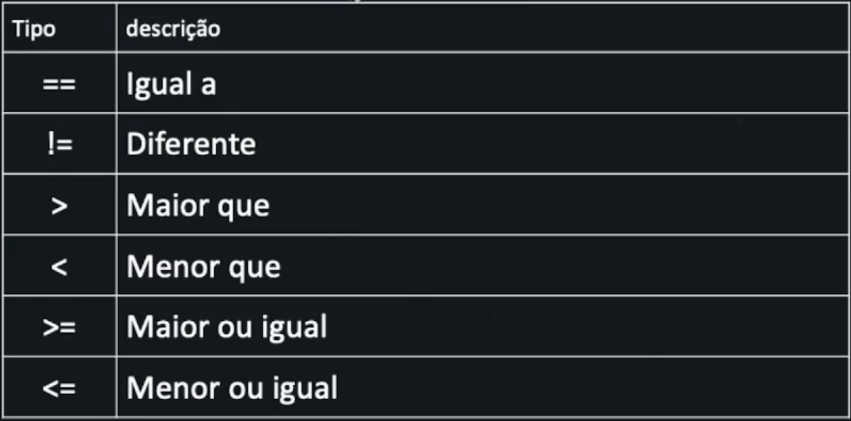
**Básicos**

****

**Ordem de Precedência**

1. **Parênteses**
2. **Multiplicação e Divisão**
3. **Subtração e Adição**

**Operadores relacionais**



**Operadores Lógicos**

****

**Estruturas condicionais (If)**

if(condição) { **SE**

}else if(condição) { SE SENÃO

}else { SENÃO

}

**Estruturas condicionais (Operador Ternário)**

(condição)?verdade:falso

Ex:

int idade = 18;  
String texto = (idade >= 18) ? "Maior Idade" : "Menor Idade;";  
System.*out*.println("Resultado: " + texto);

**Estruturas condicionais (Switch)**

Switch(valor a testar) {

case valor:

código a ser executado;

break;

}

Caso não seja colocado o break, O código irá executar todos os outros códigos.

**Arrays**



Os Arrays basicamente armazenam vários valores em posições, como se fosse uma lista de outras linguagens. No JAVA é para declarar um array é preciso colocar colchetes no tipo da variável.  
Ex:

String[] nomes = {"Paulo", "Jamilton", "Biro", "Anna"};

int[] numeros = {200, 500, 800, 1000};

**Arrays Multidimensionais**

É basicamente um array dentro do outro. Declaramos eles com dois colchetes ao lado do tipo da variável.

Ex:

String[][] conversas = {  
 {"João", "Olá, tudo bem"}, //0  
 {"Maria", "Tudo certo"}, //1  
 {"Pedro", "Não vi"}, //2  
};

Exemplo simplificado do array anterior

0 1  
0 João Olá, tudo bem  
1 Maria Tudo certo  
2 Pedro Não vi

**Estruturas de repetições (Loops)**

**While:**

While(condição) {

Executa, enquanto a condição for verdadeira.

}

**Forma de incremento**

Ex:

int numero = 10;  
numero = numero + 1;

numero += 1;

numero++;

**FOR**

O for é uma estrutura de repetição bem completa, pois ela já cria a variável inicializadora, condição e o incremento.

Ex:

for(int i = 0; i < 5; i++){

Código para execução

}

**DO WHILE**

O Do While é uma estrutura que faz primeiro uma execução e depois irá fazer o teste logico, sendo útil para programas que irá precisar colocar um ou dois itens fixos e o resto variável.

Ex:

do {

Código para execução

} while(condição);

**Programação Orientada a Objetos**

Evolução das linguagens de programação:



Os 4 pilares da orientação a objetos são: Abstração, Encapsulamento, Herança e Polimorfismo.

**Classes:**

De uma forma mais leiga de se falar, as classes é basicamente uma planta de uma casa.

**Objetos:**

Os objetos é a casa já construída e habitável.

**Atributos:**

Basicamente definem o que uma casa é. Ex: Cor, Tipo de janela, Vagas na garagem e etc.

**Métodos:**São as ações que a casa pode fazer. Ex: Abrir portão, Abrir janelas, Ligar as luzes e etc.

**Convenções de nomenclatura**

camelCase – Depois da primeira palavra, todas as outras começam com a letra maiúsculas.

snake\_case – Separa as palavras com um underline.

PascalCase – Todas as primeiras letras são maiúsculas.

kebab-case – É utilizado o traço para separar as palavras.

**Pilar 01 – Abstração**

* Definir o que do mundo real será trazido para o seu sistema.
  + Modelo – Modelos são classes.
  + Entidade – São os atores que estão relacionados com o sistema.
  + Identidade
  + Características (Atributos)
  + Ações (Métodos)

**Diferentes Métodos**

É possível fazer vários tipos de métodos, sendo eles com chamada, sem chamada e vazio.

Método Void: É um método sem retorno. Ele não consegue receber uma variável e retornar um valor especifico.

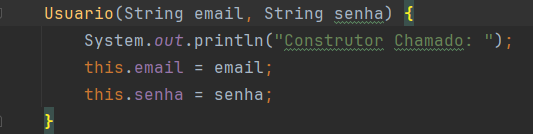
Método com retorno: Para usar o método com retorno, é só trocar o void por um tipo de variável, sendo assim, quando colocar o return no método, ele irá retornar um valor.

Método com Parâmetros: Os métodos com parâmetros são aqueles que recebe algo e é possível fazer algo cálculo, ou até usar o parâmetro recebido para devolver informações especificas.

**Construtores**

Os construtores levam o mesmo nome das classes e são usados para iniciar variáveis, ou seja, não será preciso passa valores criando variáveis depois, é só passar direto no elemento construtor.

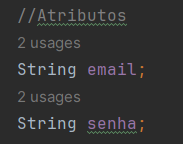
Método construtor:



Inicializando um construtor com valores:



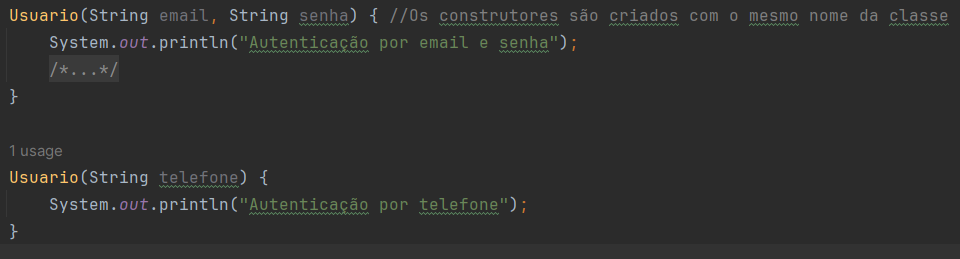
O this é usado para refencia uma variável que está fora do construtor.



**Sobrecarga de Construtor**

Assinatura de método: É composto por nome + parâmetros e tipos, ou seja, esses não podem ser todos iguais.

A sobrecarga de um construtor é quando é criado dois ou mais construtores com mesmo nome, mas com uma assinatura de método diferente.

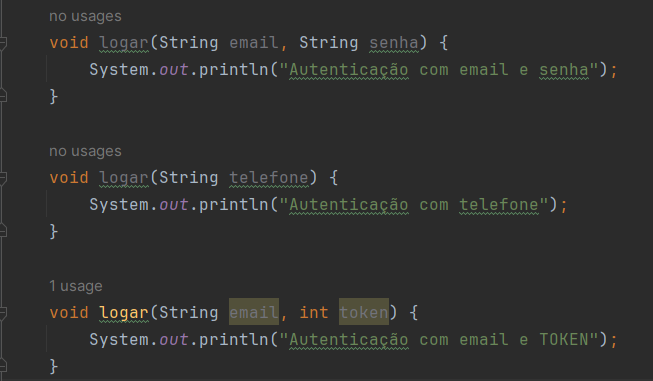


Esses dois construtores tem o mesmo nome, mas mudam os parâmetros que são passados em cada um. É útil para fazer várias validações de logins e etc.

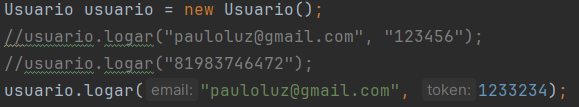
**Sobrecarga de Métodos**

A sobrecarga de métodos é igual ao de construtor. É a criação de um método com a assinatura de método.

Ex:



Cada um desses irá usar métodos diferentes, pois os parâmetros são diferentes.



**Pilar 02 – Encapsulamento**

Encapsular: Envolver em uma cápsula, incluir ou proteger algo em uma cápsula, rodear em uma cápsula.

Um exemplo é: Esconder detalhes da implementação, dando mais segurança a sua aplicação. O encapsulamento serve para controlar o acesso aos atributos e métodos de uma classe.

No encapsulamento não é preciso saber os detalhes de como funciona, mas sim fazer funcionar.

**Modificadores de acesso:**

public – Todas as classes terão acesso.

private – Apenas pode ser acessado dentro da classe

protected – Pode ser acessado dentro do mesmo pacote e/ou subclasse.

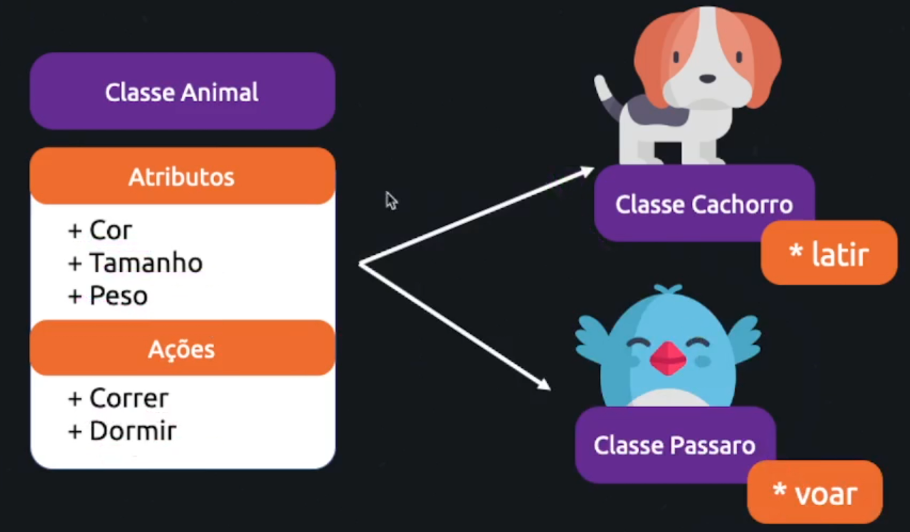
default – Caso não tenha sido definido nenhum modificador, permitindo o acesso dentro do pacote e classe.

Os modificadores podem ser aplicados nas classes, atributos e métodos.

**Pilar 03 – Herança**

Para não ficar repetindo código, é possível criar vários atributos uma única vez os atributos que são iguais, e aqueles que são diferentes é possível colocar em uma única classe.

Ex:

Assim é estabelecido uma relação de herança entre atributos e métodos.

Dry -> Don't Repeat Yourself - Não repita você mesmo - Não repita código, a não ser que seja extremamente necessário.

Extends – É usado para ligar uma classe a outra, sendo a que é ligada é a filha, e a que está ligando é a Pai ou SuperClasse. Quando esses classes forem ligadas, é possível acessar Métodos e Atributos da Classes Pai.

**Pilar 04 – Polimorfismo**

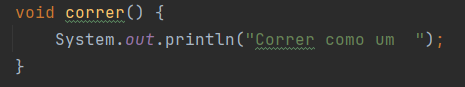
Poli -> Muitas  
Morfo -> Formas

Permite que um mesmo nome (Métodos) represente vários comportamentos diferentes.

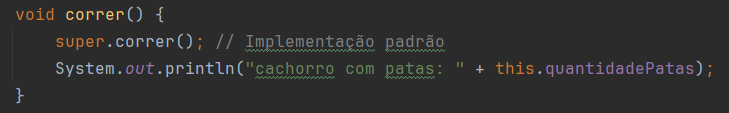
Existem dois tipos, sendo eles a Sobrecarga e a Sobreposição.

A sobrecarga é usada para não precisar ficar repetindo códigos. É possível pegar um método que irá ser usado e vários locais e fazer com que ele repita várias vezes, mas sem ficar repetindo o código.

Ex:

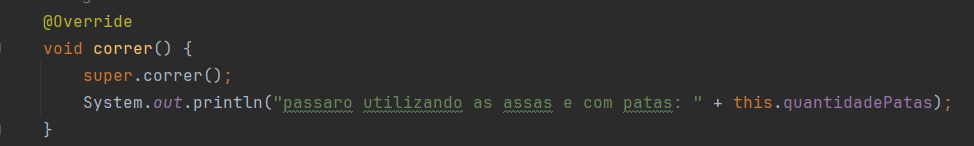


Método correr que vem da classe pai.



Método da classe filha. A palavra super.correr() vai referenciar o método da classe pai, sendo assim irá exprimir “Correr como um cachorro com patas”.

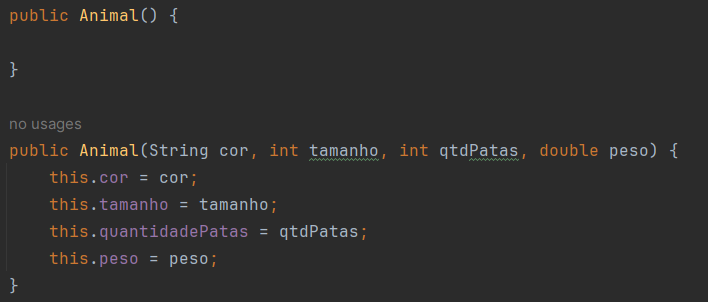
Quando um método for sobreposto é preciso colocar o @Override para mostar que esse método está sendo repetido de outra classe.



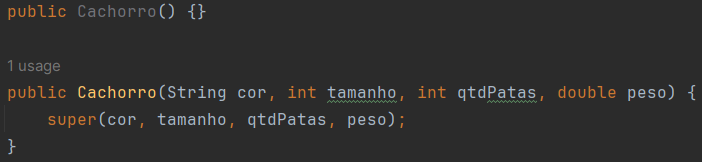
**Construtores com Herança**

Ao invés de ficar criando vários construtores com as mesmas informações, é possível criar uma vez na classe pai e executar nas classes filhas. Sendo assim, retem mais linhas de código.

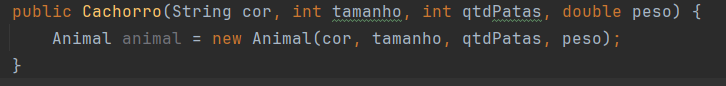
Ex:



Aqui foi criado os construtores na classe pai.



Já aqui foi usado o construtor da classe pai na classe filha. O super é o que faz a referencia de uma instancia da classe pai.



Essa é a forma não simplificada do super, mas é sempre para usar o super, pois é uma boa prática.

**Classe Abstrata vs Classe Concreta**

O objetivo da classe abstrata é fazer com que ela não seja instanciada, ou seja, que ela apenas seja herdada de outras classes como um molde para outras classes, sendo elas concretas.

Para mostrar que uma classe é abstrata, é preciso colocar abstract antes do nome class.  
Ex:

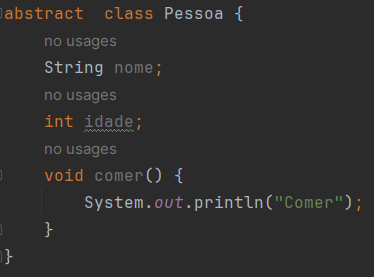
 Assim é possível dizer que ela não será instanciada.

**Interface**

* Pode-se dizer a grosso modo, que uma interface é um contrato que quando assumido por uma classe, deve ser executado.
* Interface é utiliza, pois podemos ter muitos objetos (classes) que podem possuir a mesma ação (métodos), porém, podem executá-las de maneiras diferentes.

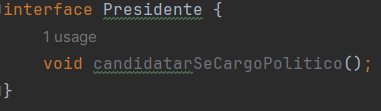
Ex:

Aqui temos uma classe pessoa que tem essas informações.

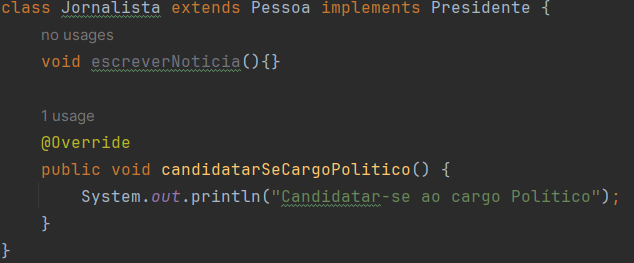


Mas se precisarmos colocar alguém para ser Presidente, mas apenas as pessoas que querem, são as que podem tentar a candidatura.

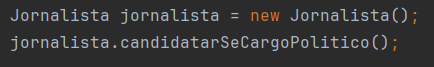
Então precisamos criar uma interface para que ela possa ser adicionada a uma classe que deseja usar o seu método.



E para implementar esse método em uma classe, precisamos utilizar o implements Presidente.



Quando utilizamos esse implements, ele irá criar um método vazio com uma override para que possa ser utilizado.



Para utilizar é preciso instanciar uma a classe e utilizar como se fosse um método normal.

**Métodos Abstratos e Passagem de Parâmetros**

Métodos Abstratos: Se o método for abstract, todas as classes filhas vão precisar usá-lo, mas de forma diferente.

Ex:

 Cada classe que tiver extends vai precisar usar esse método.

Sobre a passagem de parâmetros, existem dois tipos.

* Por valor – Ele irá pegar o valor de uma variável e passar para o método
* Por referência – Ele irá pegar a referencia do objeto e passar pelo método onde está o objeto.