**Algumas Teorias**

**Miscelânia**

**When:** É uma espécie de Switch Case, sem a necessidade do break:

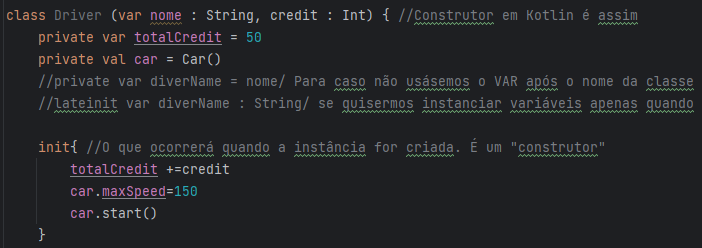
when{  
 bmi<18.5 ->{  
 resultText="Underweight"  
 color=R.color.*underweight* }  
 bmi in 18.5..24.99->{  
 resultText="Healthy"  
 color=R.color.*normalWeight* }  
 bmi in 25.0..29.99->{  
 resultText="Overweight"  
 color= R.color.*overWeight* }  
 bmi>29.9 ->{  
 resultText="Obese"  
 color=R.color.*obese* }  
}

**Safe Call Operator (?):** Representado por uma interrogação, é usado em parâmetros de funções ou sempre que quiser evitar um NullPointerException, onde houver a possibilidade de um Null ocorrer. Por isso, é muito comum ser usado em métodos de validação:

private fun validateInput(weight:String?,height:String?):Boolean {// The ? is the SafeCallOperator. It avoids NullPointerException  
 return when {  
 weight.*isNullOrEmpty*() -> {  
 Toast.makeText(this, "Weight is empty", Toast.*LENGTH\_SHORT*).show()  
 return false  
 }  
  
 height.*isNullOrEmpty*() -> {  
 Toast.makeText(this, "Height is empty", Toast.*LENGTH\_SHORT*).show()  
 return false  
 }  
  
 else -> {  
 return true  
 }  
 }  
}

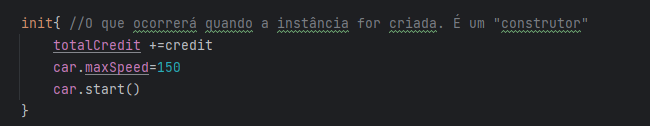
**Construtores e Classes**

Ao instanciar uma classe, podemos fazer um “construtor” ao abrir um parêntese após o nome da própria Classe. Se usarmos val ou var, criaremos uma variável que pode ser utilizada, se não usarmos, será apenas um parâmetro:



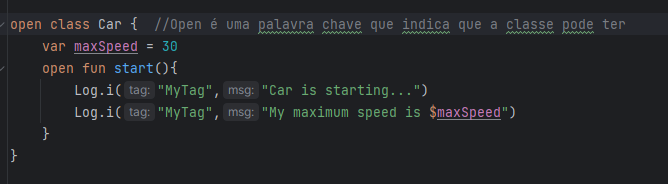
Aqui, “nome” é uma variável que pode ser utilizada por métodos, e é um atributo da classe driver. Já “credit” apenas armazena um valor, e não pode ser usado diretamente como uma variável.

**Init:** Porém, semelhante a um construtor em Java, temos o método init. Ele ocorre sempre que um objeto é instanciado:

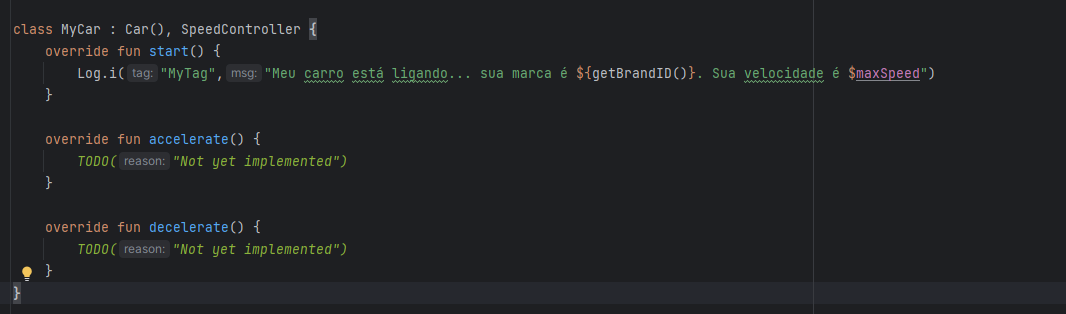


**Herança**

**Open:** Open é uma palavra chave que indica que uma classe pode se tornar uma classe Pai.



E assim seria uma classe filha:



**Override:** Podemos sobrescrever métodos nas classes filhas. Para isso usamos a palavra chave “open” no método desejado, e override na classe filha. Exemplo:

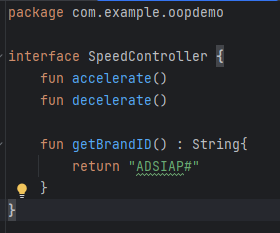
Na classe pai:

open fun start(){  
 Log.i("MyTag","Car is starting...")  
 Log.i("MyTag","My maximum speed is $maxSpeed")  
}

Na classe filha:

override fun start() {  
 Log.i("MyTag","Meu carro está ligando... sua marca é ${getBrandID()}. Sua velocidade é $maxSpeed")  
}

**Interface:** Uma interface pode ser implementada por uma classe para suas funcionalidades definidas. Ela gera uma espécie de “contrato”.

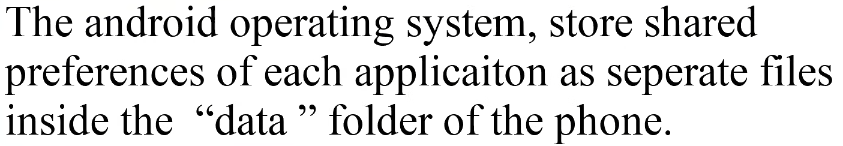
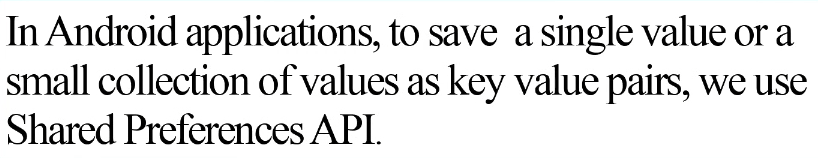


Note que a interface não precisa usar a palavra chave “open”. Na classe MyCar, além de estendermos a classe Car, coloquei a interface SpeedController. Além disso, a interface pode ter métodos abstratos (accelerate, decelerate) e concretos (getBrandID).

Nem toda função é um método. Métodos são aquelas relacionadas a objetos. Logo, uma função seria equivalente a um método “static” em Java.

**Shared Preferences**

Importante mecanismo de armazenamento de dados para pequenas quantidades de dados.



Um exemplo seria ao configurar um app. Imagine que ao colocar seus dados, você é interrompido e fecha o aplicativo. Mesmo voltando dias depois, seus dados ainda estariam lá.

É preciso criar variáveis de referência tanto para o arquivo de SharedPreferences tanto para o editor, que é o responsável por fazer as mudanças:

private lateinit var sf:SharedPreferences  
private lateinit var editor:SharedPreferences.Editor