Sumário

[Classes 1](#_Toc78385175)

[Declaração de classes 2](#_Toc78385176)

[Construtores 3](#_Toc78385177)

[Construtores e propriedades 4](#_Toc78385178)

[Imutabilidade dos parâmetros dos construtores secundários 7](#_Toc78385179)

[Instanciando uma classe 7](#_Toc78385180)

[Funções membro 8](#_Toc78385181)

[Construtores e blocos de inicialização 9](#_Toc78385182)

[Modificadores de Acesso 9](#_Toc78385183)

[Público 9](#_Toc78385184)

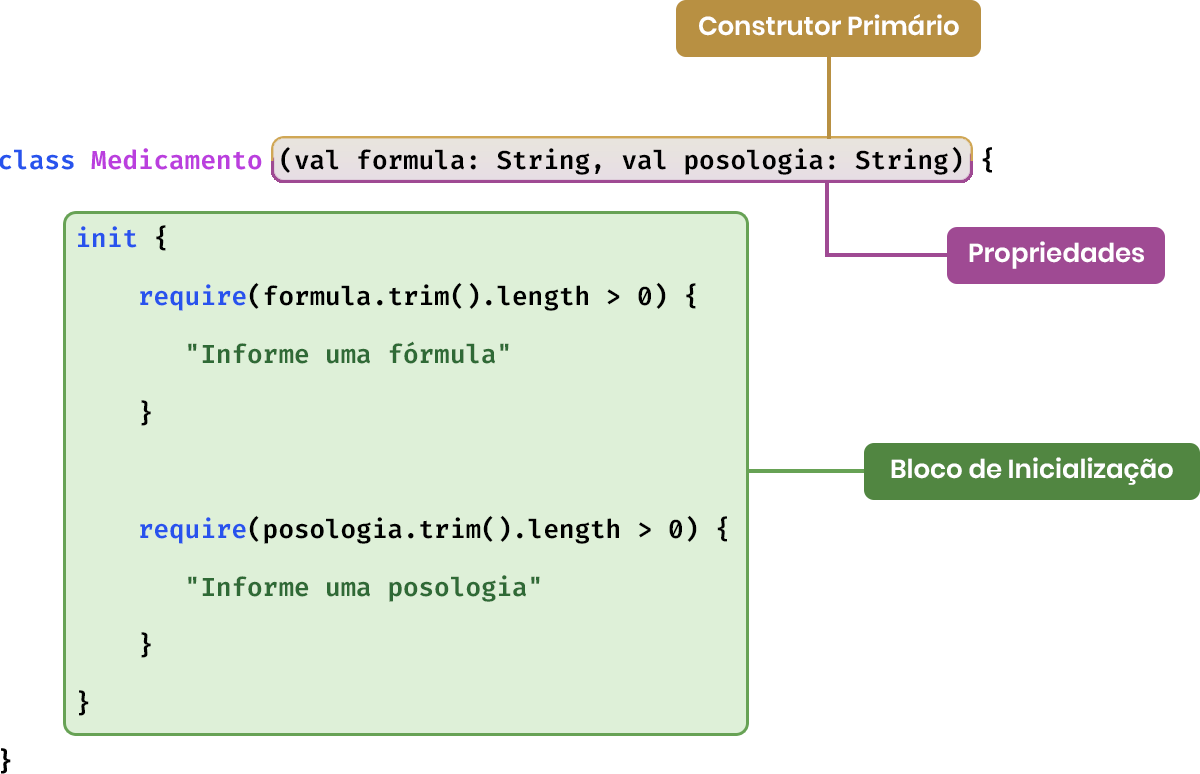
[Privado 9](#_Toc78385185)

[Protegido 10](#_Toc78385186)

[Interno 10](#_Toc78385187)

# Classes

Uma classe é um bloco de construção de software fundamental, encontrado na grande maioria das linguagens orientadas a objetos e, dessa forma, também presente em Kotlin. Elas são usadas para criar novos tipos de dados, utilizando uma estrutura composta por métodos e variáveis que podem possuir os seus próprios tipos.



## Declaração de classes

Em Kotlin, um arquivo pode conter diferentes declarações de classes, as quais são feitas utilizando a palavra reservada class.

**class Medicamento**

No exemplo acima, uma vez que a classe Medicamento não possui um corpo, podemos omitir as chaves que estariam na frente do seu nome.

Uma classe é uma unidade de programa que agrupa funções e dados para executar algumas tarefas relacionadas. Declaramos uma classe Kotlin usando a palavra-chave class — de modo semelhante ao Java.

**class Book**

O código anterior é a declaração de classe mais simples— acabamos de criar uma classe vazia chamada Book. Nós ainda podemos instanciar essa classe, mesmo que ela não contenha um corpo, usando seu construtor padrão.

**val book = Book()**

Como se pode observar no código acima, não usamos a palavra-chave new para instanciar essa classe — como é usual em outras linguagens de programação. new não é uma palavra-chave em Kotlin. Isso torna nosso código fonte conciso ao criar uma instância da classe. Mas lembre-se que instanciação de uma classe Kotlin em Java exigirá a palavra-chave new.

// In a Java file

**Book book = new Book()**

## Construtores

Um construtor é uma função especial da classe, utilizada na criação e inicialização dos objetos derivados dela.

Kotlin distingue os construtores de uma classe entre primários e secundários.

**O construtor primário de uma classe faz parte do seu cabeçalho e pode conter apenas uma lista de parâmetros, assim como apresentado abaixo.**

**class Medicamento(val formula: String, val posologia: String)**

Dado que o construtor primário não pode conter nenhum código, uma classe pode conter um ou mais blocos de inicialização, que são executados na ordem como são declarados no corpo da mesma.

Nesses blocos de inicialização, conforme visto no Código 1, podemos acessar quaisquer parâmetros que estejam presentes no construtor primário.

**class Medicamento(val formula: String, val posologia: String) {**

**init {**

**require(formula.trim().length > 0) {**

**"Informe uma fórmula"**

**}**

**require(posologia.trim().length > 0) {**

**"Informe uma posologia"**

**}**

**}**

**}**

Além do construtor primário, uma classe também pode declarar um ou mais construtores secundários.

Esses, por sua vez, sempre devem utilizar a palavra reservada constructor em suas declarações, mesmo quando um modificador de acesso for omitido ou nenhuma anotação for utilizada.

**class Medicamento {**

**constructor(formula: String, posologia: String)**

**}**

No caso de uma classe possuir mais de um construtor, sendo um deles primário, cada construtor secundário deve delegar ao construtor primário.

Quando ambos os construtores estiverem na mesma classe, a delegação de um construtor para o outro é feita com a palavra reservada this.

Podemos utilizar valores padrão do construtor delegado de uma classe. Para demonstrar isso vamos adicionar um terceiro parâmetro no construtor para a contraindicação do medicamento.

Considerando que a classe Medicamento possa ser inicializada sem um valor para essa propriedade, caso no qual o mesmo passará a ser uma mensagem padrão, poderíamos modelá-la da seguinte forma:

**class Medicamento(val formula: String, posologia: String, val contraindicacao: String) {**

**constructor(formula: String, posologia: String): this(formula, posologia,**

**"Este medicamento não é indicado para pessoas alérgicas a $formula")**

**}**

Dessa forma, na delegação do construtor this(formula, posologia, "Este medicamento não é indicado para pessoas alérgicas a $formula") informamos um valor padrão para a propriedade contraindicacao e assim a classe poderá ser instanciada com ou sem esse terceiro parâmetro.

**val medicamento1 = Medicamento("C8H9NO2", "...")**

**val medicamento2 = Medicamento("C8H9NO2", "...", "Minha contraindicação")**

No exemplo acima, o objeto medicamento1 será iniciado com o valor padrão para a propriedade contraindicacao, que é "Este medicamento não é indicado para pessoas alérgicas a $formula".

Para o objeto medicamento2, o valor da propriedade contraindicacao será aquele informado no construtor, que é "Minha contraindicação".

## Construtores e propriedades

Vamos ver como adicionar um construtor e propriedades para nossa classe. Mas primeiro, vamos ver uma classe típica em Java:

**/\* Java \*/**

**public class Book {**

**private String title;**

**private Long isbn;**

**public Book(String title, Long isbn) {**

**this.title = title;**

**this.isbn = isbn;**

**}**

**public String getTitle() {**

**return title;**

**}**

**public void setTitle(String title) {**

**this.title = title;**

**}**

**public Long getIsbn() {**

**return isbn;**

**}**

**public void setIsbn(Long isbn) {**

**this.isbn = isbn;**

**}**

**}**

Olhando para nossa classe de modelo Book acima, temos o seguinte:

dois campos: title e isbn

um construtor único

getters e setters para os dois campos (felizmente, o IntelliJ IDEA pode nos ajudar a gerar esses métodos)

Agora vamos ver como podemos escrever o código anterior em Kotlin em vez disso:

**/\* Kotlin \*/**

**class Book {**

**var title: String**

**var isbn: Long**

**constructor(title: String, isbn: Long) {**

**this.title = title**

**this.isbn = isbn**

**}**

**}**

Uma classe bem arranjada! Agora reduzimos o número de linhas de código de 20 para apenas 9.

A função constructor() é chamada de construtor secundário em Kotlin. Esse construtor é equivalente ao construtor Java que chamamos ao instanciar uma classe.

O construtor primário age de forma diferente do construtor secundário quanto a geração de propriedades para a classe na qual eles são declarados.

Essa abordagem impede que as propriedades de uma classe variem de acordo com os parâmetros declarados nos construtores, uma vez que a geração de propriedades é limitada ao construtor primário de uma classe.

Uma coisa incrível é que os getters e setters para essas propriedades são gerados automaticamente para nós por debaixo dos panos pelo compilador Kotlin. Observe que nós não especificamos quaisquer modificadores de visibilidade para essas propriedades — por padrão, elas são públicas. Em outras palavras, elas podem ser acessadas de qualquer lugar.

Vejamos uma outra versão da mesma classe em Kotlin:

**class Book constructor(title: String, isbn: Long) {**

**var title: String**

**var isbn: Long**

**init {**

**this.title = title**

**this.isbn = isbn**

**}**

**}**

Nesse código nós removemos o construtor secundário. Em vez disso, nós declaramos um construtor no cabeçalho da classe chamado de construtor primário. Um construtor primário não tem qualquer lugar para colocar um bloco de código, então utilizamos o modificador init para inicializar os parâmetros de entrada do construtor primário. Observe que o bloco de código init é executado imediatamente quando a instância da classe é criada.

Como você pode ver, nosso código ainda tem um monte de clichês. Vamos reduzi-lo ainda mais:

**class Book constructor(var title: String, var isbn: Long)**

Nossa classe Book agora é apenas uma linha de código. Isso é muito legal! Observe que na lista de parâmetros do construtor primário, definimos nossas propriedades mutáveis: title e isbn diretamente dentro do construtor primário com a palavra-chave var.

Podemos também adicionar valores padrão para qualquer uma das propriedades de classe dentro do construtor.

**class Book constructor(var title: String = "default value", var isbn: Long)**

Na verdade, nós também podemos omitir a palavra-chave constructor, mas somente se ele não tem nenhum modificador de visibilidade (public, private ou protected) ou quaisquer anotações.

**class Book (var title: String = "default value", var isbn: Long)**

Agora podemos criar uma instância de classe como esta:

**val book = Book("A Song of Ice and Fire", 9780007477159)**

**val book2 = Book(1234) // uses the title property's default value**

**println(book.title)**

## Imutabilidade dos parâmetros dos construtores secundários

Conforme demonstrado anteriormente, os parâmetros do construtor secundário são declarados sem utilizar as palavras reservadas val ou var.

**class Medicamento {**

**constructor(formula: String, posologia: String)**

**}**

Caso uma tentativa de fazer o contrário seja feita, digamos declarando a fórmula como val formula: String, um erro de compilação será emitido com a seguinte mensagem

**Kotlin: 'val' on secondary constructor parameter is not allowed**

Em Kotlin, tanto os parâmetros de funções quanto os de construtores são imutáveis por definição, o que elimina a necessidade de se utilizar val ou var nesse caso.

Em adendo, a utilização de val ou var em construtores é restrita ao construtor primário de uma classe para definir a mutabilidade das suas propriedades, será imutável ou não, dado que apenas ele pode gerar propriedades para uma classe.

Por exemplo, digamos que a fórmula de um medicamento nunca possa alterada, uma vez que seja definida no momento da criação de uma instância, mas que a sua posologia depende de uma regra externa a essa classe. Dessa forma, podemos modelar a classe Medicamento conforme apresentado a seguir:

**class Medicamento(val formula: String, var posologia: String)**

Tendo sido o parâmetro formula declarado imutável, a propriedade homônima da classe Medicamento também será somente leitura. Ao tentar atribuir valor a ela a partir de uma instância o seguinte erro será apresentado:

**Error:(8, 5) Kotlin: Val cannot be reassigned**

O mesmo erro não ocorrerá ao atribuir valor a propriedade posologia a partir de uma instância da classe Medicamento, uma vez que o parâmetro posologia, presente no construtor primário, foi declarado mutável.

## Instanciando uma classe

Para criar uma instância de uma classe usamos seu nome e construtor. Em Koltin não utilizamos a palavra-chave new, como mostra o Código 10.

**class Medicamento(val formula: String, val posologia: String) {**

**}**

**val medicamento = Medicamento("C8H9NO2", "...")**

Aqui devemos observar que se uma classe possui um construtor, sendo ele primário ou secundário, o mesmo deve ser invocado. Por exemplo, a classe Medicamento possui um construtor secundário, o que torna obrigatório instanciá-la da forma Medicamento("C8H9NO2", "...").

**class Medicamento {**

**constructor(formula: String, posologia: String)**

**}**

Caso isso não seja feito e a classe seja instanciada como Medicamento(), sem que os argumentos sejam fornecidos para o construtor, teremos um erro de compilação com a mensagem

**Error:(10, 35) Kotlin: No value passed for parameter 'formula'**

**Error:(10, 35) Kotlin: No value passed for parameter 'posologia',**

onde formula e posologia são propriedades da classe.

## Funções membro

Funções membro são funções declaradas dentro de classes. As regras que aprendemos anteriormente para a escrita de funções também se aplicam aqui com uma exceção, funções membro podem utilizar a palavra-chave this para referenciar a instância atual.

Funções membro sempre devem ser invocadas a partir de instâncias da classe.

**class Medicamento(val formula: String, val posologia: String) {**

**fun contem(formula: String) = this.formula.contains(formula, ignoreCase = true)**

**}**

**fun main() {**

**val medicamento = Medicamento("C8H9NO2", "...")**

**if (medicamento.contem("C8H9NO2")) {**

**println("Este medicamento contém paracetamol")**

**}**

**}**

Observe que na Linha 3 utilizamos this para diferenciar entre a propriedade e o parâmetro formula.

## Construtores e blocos de inicialização

A delegação de um construtor secundário para o construtor primário ocorrerá como sendo a primeira instrução no construtor secundário. Isso quer dizer que, uma vez que eles passam a fazer parte do construtor primário, cada bloco de inicialização será executado antes de qualquer construtor secundário.

Fundamentalmente, caso uma classe não possua nenhum construtor e não seja abstrata, um construtor público vazio lhe será atribuído. Supondo que nessa mesma classe não exista um construtor primário declarado, a delegação ocorrerá de forma implícita, como demonstrado no Código 16.

**class Medicamento {**

**val formula: String**

**constructor(formula: String, posologia: String) {**

**this.formula = formula**

**println("Construtor secundário")**

**}**

**init {**

**println("Bloco de inicialização")**

**}**

**}**

No exemplo acima, não importando a ordem da declaração do construtor secundário e do bloco de inicialização, as mensagens exibidas serão “Bloco de inicialização” e “Construtor secundário”, nessa ordem.

# Modificadores de Acesso

Modificadores de visibilidade nos ajudam a restringir a acessibilidade de nossa API para o público. Nós podemos fornecer modificadores de visibilidade diferentes para nossas classes, interfaces, objetos, métodos ou propriedades. Kotlin nos fornece quatro modificadores de visibilidade:

## Público

Este é o padrão, e qualquer classe, função, propriedade, interface ou objeto que possui este modificador pode ser acessada de qualquer lugar.

## Privado

Uma função de nível superior, interface ou classe que é declarada como private pode ser acessada apenas dentro do mesmo arquivo.

Qualquer função ou propriedade que é declarada private dentro de uma classe, interface ou objeto só pode ser visível para outros membros da mesma classe, objeto ou interface.

class Account {

private val amount: Double = 0.0

}

## Protegido

O modificador protected só pode ser aplicado a propriedades ou funções dentro de uma classe, interface ou objeto — não pode ser aplicada a funções de nível superior, classes ou interfaces.Propriedades ou funções com este modificador são acessíveis apenas dentro da classe que o define e em qualquer subclasse.

## Interno

Em um projeto que tem um módulo (Gradle ou Maven), uma classe, objeto, interface ou a função especificada com o modificador internal declarado dentro desse módulo é acessível unicamente dentro desse módulo.

internal class Account {

val amount: Double = 0.0

}