# Herança

Por padrão, tudo no Kotlin é final; portanto, precisamos usar a palavra-chave “open” na frente da declaração da classe para permitir a herança. Dê uma olhada no seguinte exemplo de herança.

open class ABC {

fun think () {

print("Hey!! i am thiking ")

}

}

class BCD: ABC(){

// herança acontece usando o construtor padrão

}

fun main() {

var a = BCD()

a.think()

}

O trecho de código acima produzirá a seguinte saída no navegador.

Hey!! i am thiking

Agora, e se quisermos substituir o método think () na classe filho. Então, precisamos considerar o exemplo a seguir, em que estamos criando duas classes e substituindo uma de suas funções na classe filho.

open class ABC {

open fun think () {

print("Hey!! i am thinking ")

}

}

class BCD: ABC() { // herança acontece usando o construtor padrão

override fun think() {

print("I Am from Child")

}

}

fun main() {

var a = BCD()

a.think()

}

O trecho de código acima chamará o método herdado da classe filho e produzirá a seguinte saída no navegador. Como o Java, o Kotlin também não permite múltiplas heranças.

I Am **from** Child

# Interface

No Kotlin , a interface funciona exatamente semelhante ao Java 8, o que significa que eles podem conter a implementação de métodos, bem como a declaração de métodos abstratos. Uma interface pode ser implementada por uma classe para usar sua funcionalidade definida.

A palavra-chave “interface” é usada para definir uma interface no Kotlin conforme mostrado no seguinte trecho de código.

interface ExampleInterface {

var myVar: String // abstract property

fun absMethod() // abstract method

fun sayHello() = "Hello there" // method with default implementation

}

No exemplo acima, criamos uma interface chamada "ExampleInterface" e, dentro dela, temos algumas propriedades e métodos abstratos todos juntos. Veja a função chamada "sayHello ()", que é um método implementado.

No exemplo a seguir, implementaremos a interface acima em uma classe.

interface ExampleInterface {

var myVar: Int // abstract property

fun absMethod():String // abstract method

fun hello() {

println("Hello there")

}

}

class InterfaceImp : ExampleInterface {

override var myVar: Int = 25

override fun absMethod() = "Happy Learning "

}

fun main() {

val obj = InterfaceImp()

println("My Variable Value is = ${obj.myVar}")

print("Calling hello(): ")

obj.hello()

print("Message from Caique ")

println(obj.absMethod())

}

O trecho de código acima produzirá a seguinte saída no navegador.

My Variable Value is = 25

Calling hello(): Hello there

Message from Caique Happy Learning

Como mencionado anteriormente, o Kotlin não suporta várias heranças, no entanto, a mesma coisa pode ser alcançada implementando mais de duas interfaces por vez.

No exemplo a seguir, criaremos duas interfaces e posteriormente implementaremos as duas interfaces em uma classe.

interface A {

fun printMe() {

println(" method of interface A")

}

}

interface B {

fun printMeToo() {

println("I am another Method from interface B")

}

}

// implements two interfaces A and B

class multipleInterfaceExample: A, B

fun main(){

val obj = multipleInterfaceExample()

obj.printMe()

obj.printMeToo()

}

No exemplo acima, criamos duas interfaces de amostra A, B e na classe denominada “multipleInterfaceExample” implementamos duas interfaces declaradas anteriormente. O trecho de código acima produzirá a seguinte saída no navegador.

method of interface A

I am another Method from interface B

# Kotlin - Controle de visibilidade

**O modificador de acesso** é usado para restringir o uso das variáveis, métodos e classe usados no aplicativo. Como outra linguagem de programação OOP, esse modificador é aplicável em vários locais, como no cabeçalho da classe ou na declaração do método. Existem quatro modificadores de acesso disponíveis no Kotlin .

## **Privado**

As classes, métodos e pacotes podem ser declarados com um modificador privado. Uma vez que qualquer coisa é declarada como privada, será acessível dentro de seu escopo imediato. Por exemplo, um pacote privado pode ser acessível dentro desse arquivo específico. Uma classe ou interface privada pode ser acessível apenas por seus membros de dados, etc.

**private** **class** **privateExample** {

**private** **val** i = 1

**private** **val** doSomething() {

}

}

No exemplo acima, a classe **“privateExample”** e a variável i podem ser acessíveis apenas no mesmo arquivo Kotlin , onde é mencionado, pois todos são declarados como privados no bloco de declaração.

## **Protegido**

Protected é outro modificador de acesso para o Kotlin , que atualmente não está disponível para declaração de nível superior, pois nenhum pacote pode ser protegido. Uma classe ou interface protegida é visível apenas para sua subclasse.

**class** **A**() {

**protected** **val** i = 1

}

**class** **B** : A() {

**fun** **getValue**() : Int {

**return** i

}

}

No exemplo acima, a variável **“i”** é declarada como protegida, portanto, é visível apenas para sua subclasse.

## **interno**

Internal é um modificador recém-adicionado, introduzido no Kotlin . Se algo estiver marcado como interno, esse campo específico estará no campo interno. Um pacote interno é visível apenas dentro do módulo em que é implementado. Uma interface de classe interna é visível apenas por outra classe presente dentro do mesmo pacote ou módulo. No exemplo a seguir, veremos como implementar um método interno.

**class** **internalExample** {

**internal** **val** i = 1

**internal** **fun** **doSomething**() {

}

}

No exemplo acima, o método chamado “doSomething” e a variável são mencionados como internos, portanto, esses dois campos podem ser acessíveis apenas dentro do pacote sob o qual é declarado.

## **Público**

O modificador público pode ser acessado de qualquer lugar no espaço de trabalho do projeto. Se nenhum modificador de acesso for especificado, por padrão, ele estará no escopo público. Em todos os nossos exemplos anteriores, não mencionamos nenhum modificador, portanto, todos eles estão no escopo público. A seguir, é apresentado um exemplo para entender mais sobre como declarar uma variável ou método público.

**class** **publicExample** {

**val** i = 1

**fun** **doSomething**() {

}

}

No exemplo acima, não mencionamos nenhum modificador, portanto, todos esses métodos e variáveis são por padrão públicos.