# Kotlin

## Please Kno…

# Mau Bocanegra

- Android (7 años) & iOS Dev (3 años)
- IoT Rookie (https://distillery.com/blog/)
- Vegan

@MauBocanegra

linkedin.com/in/maubocanegra/


# ¿Por qué?

# ¿Por qué?

# ¿Por qué?

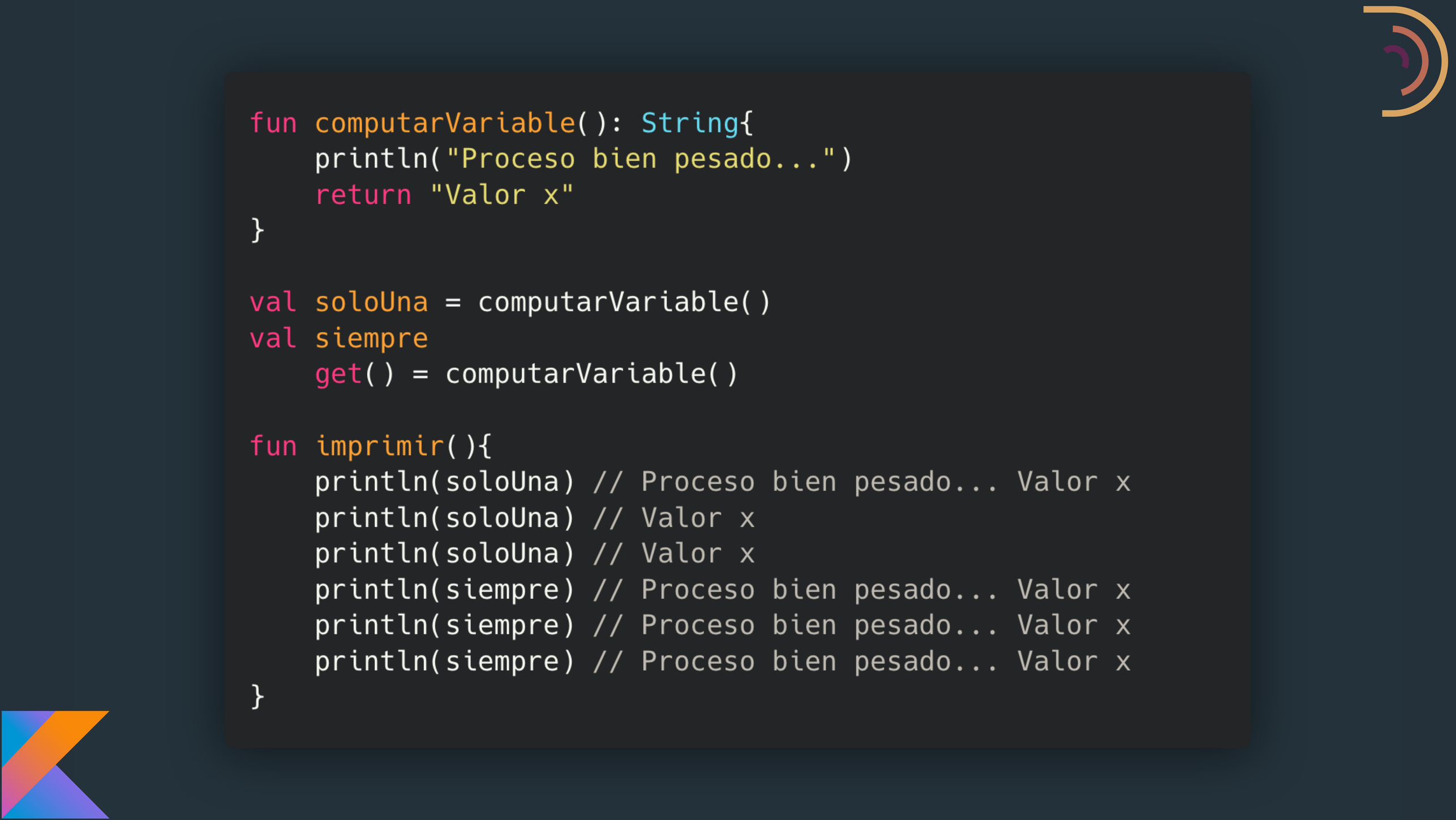
# ¿Y esas son…? 👀

- Getters
- ¿Inmutabilidad?
- Down-casting
- Exposure
- Legibilidad
- Class Shadowing

# Getters

```kotlin
fun computarVariable(): String{
    println("Proceso bien pesado...")
    return "Valor x"
}
```

```kotlin
fun computarVariable(): String{
    println("Proceso bien pesado...")
    return "Valor x"
}

val soloUna = computarVariable()
val siempre
    get() = computarVariable()
```

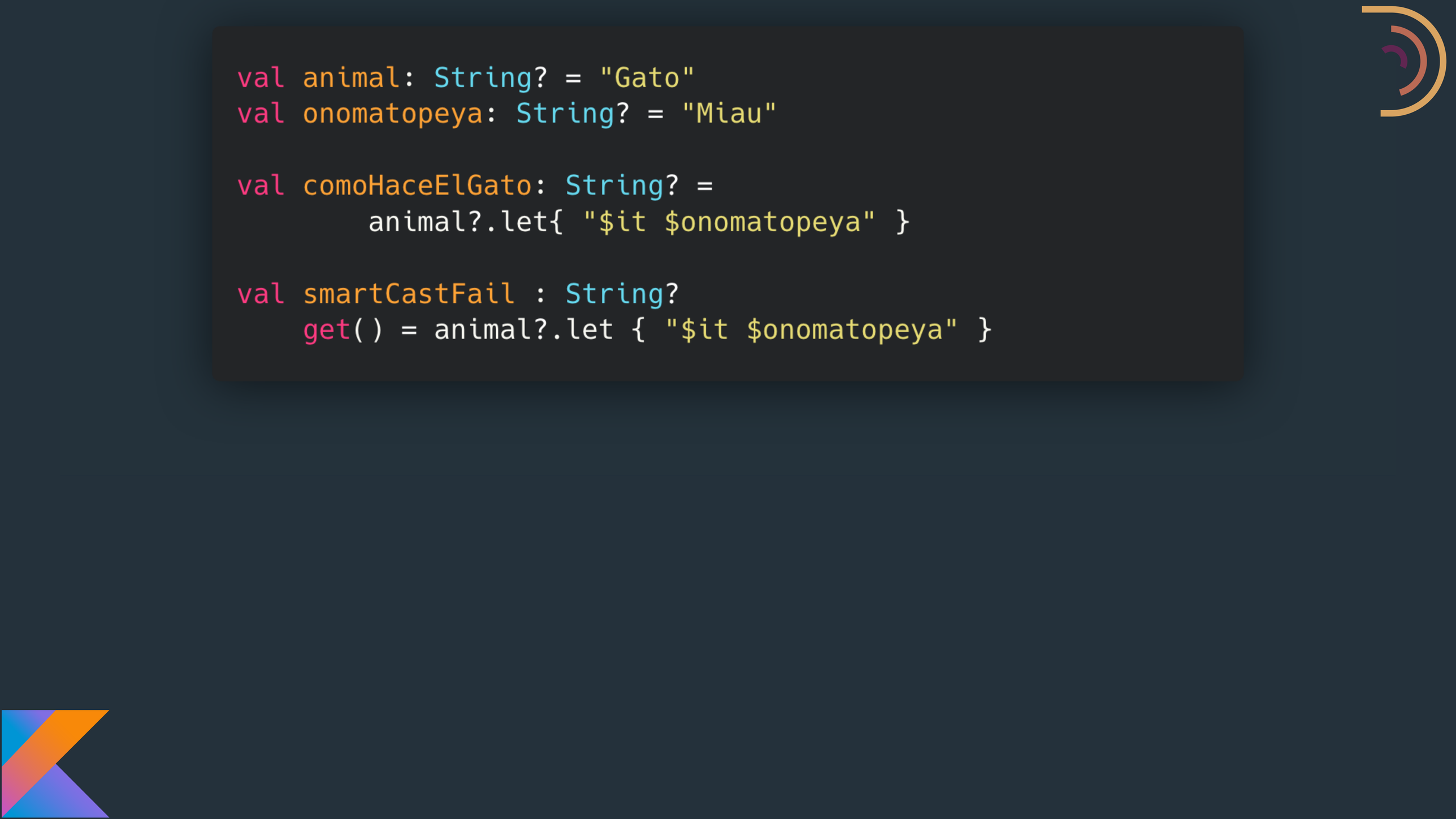


```
fun computarVariable(): String{
    println("Proceso bien pesado...")
    return "Valor x"
}

val soloUna = computarVariable()
val siempre
    get() = computarVariable()

fun imprimir(){
    println(soloUna) // Proceso bien pesado... Valor x
    println(soloUna) // Valor x
    println(soloUna) // Valor x
    println(siempre) // Proceso bien pesado... Valor x
    println(siempre) // Proceso bien pesado... Valor x
    println(siempre) // Proceso bien pesado... Valor x
}
```

# Inmutabilidad

```
val animal: String? = "Gato"
val onomatopeya: String? = "Miau"
```

```kotlin
val animal: String? = "Gato"
val onomatopeya: String? = "Miau"

val comoHaceElGato: String? =
        animal?.let{ "$it $onomatopeya" }
```



```kotlin
val animal: String? = "Gato"
val onomatopeya: String? = "Miau"

val comoHaceElGato: String? =
        animal?.let{ "$it $onomatopeya" }

val smartCastFail : String?
    get() = animal?.let { "$it $onomatopeya" }
```

```kotlin
val animal: String? = "Gato"
val onomatopeya: String? = "Miau"

val comoHaceElGato: String? =
        animal?.let{ "$it $onomatopeya" }

val smartCastFail : String?
    get() = animal?.let { "$it $onomatopeya" }

fun smartCastIntent(){
    if(smartCastFail != null){
        // println(smartCastFail.length)
        // Open or custom getter
    }

    if(comoHaceElGato != null ){
        println(comoHaceElGato.length)
    }
}
```

# Down-casting

Si sirve… ¡No lo toques!

# Down-casting (collections)

Si sirve… ¡No lo toques!

# Down-casting (collections)

Si sirve… ¡No lo toques!

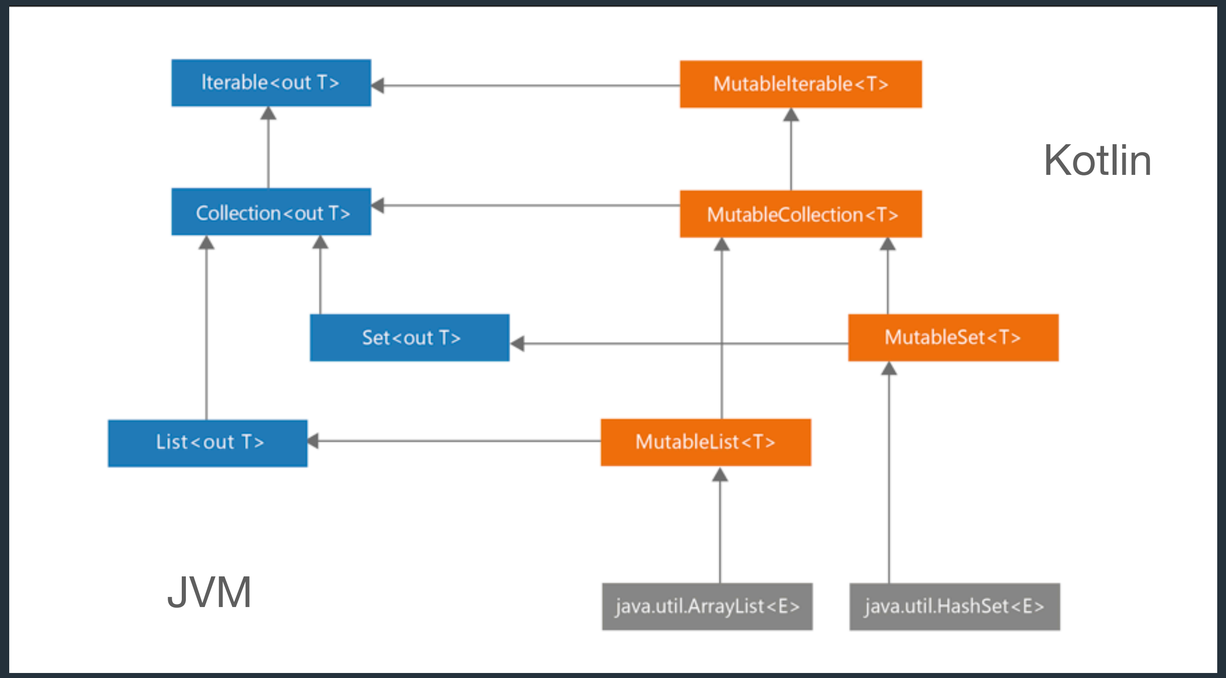
# Down-casting (collections)

Si sirve… ¡No lo toques!

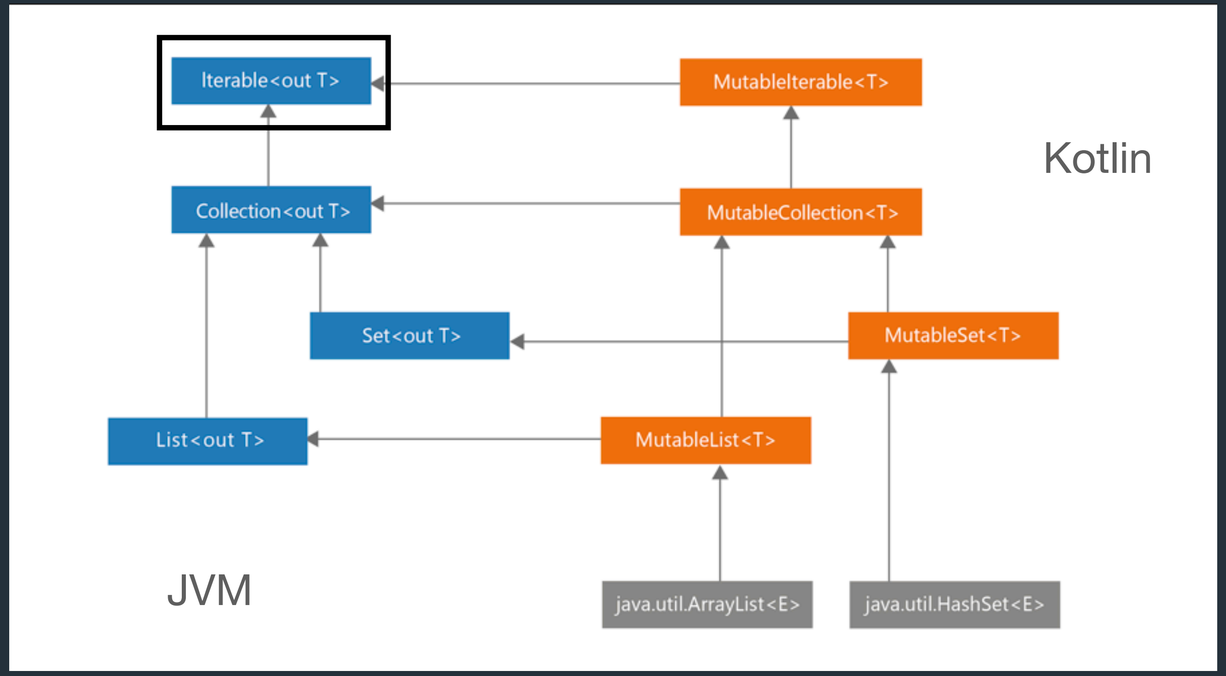
# Down-casting (collections)

Si sirve… ¡No lo toques!

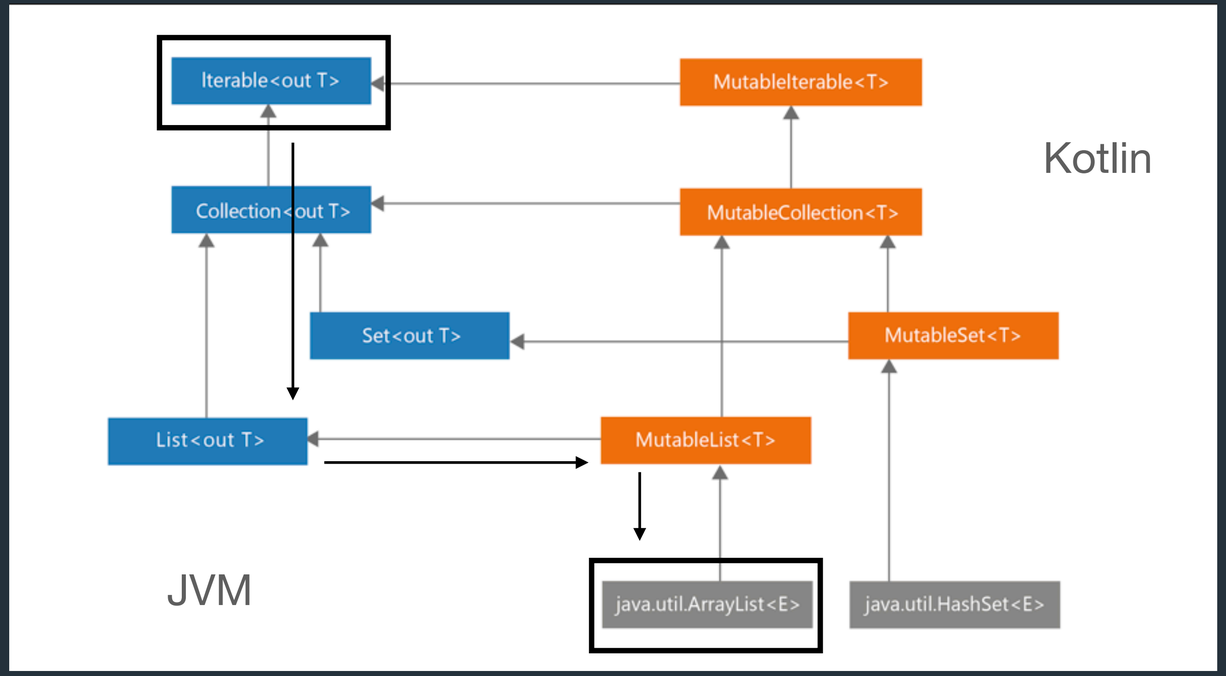
# Down-casting
## (Collections)



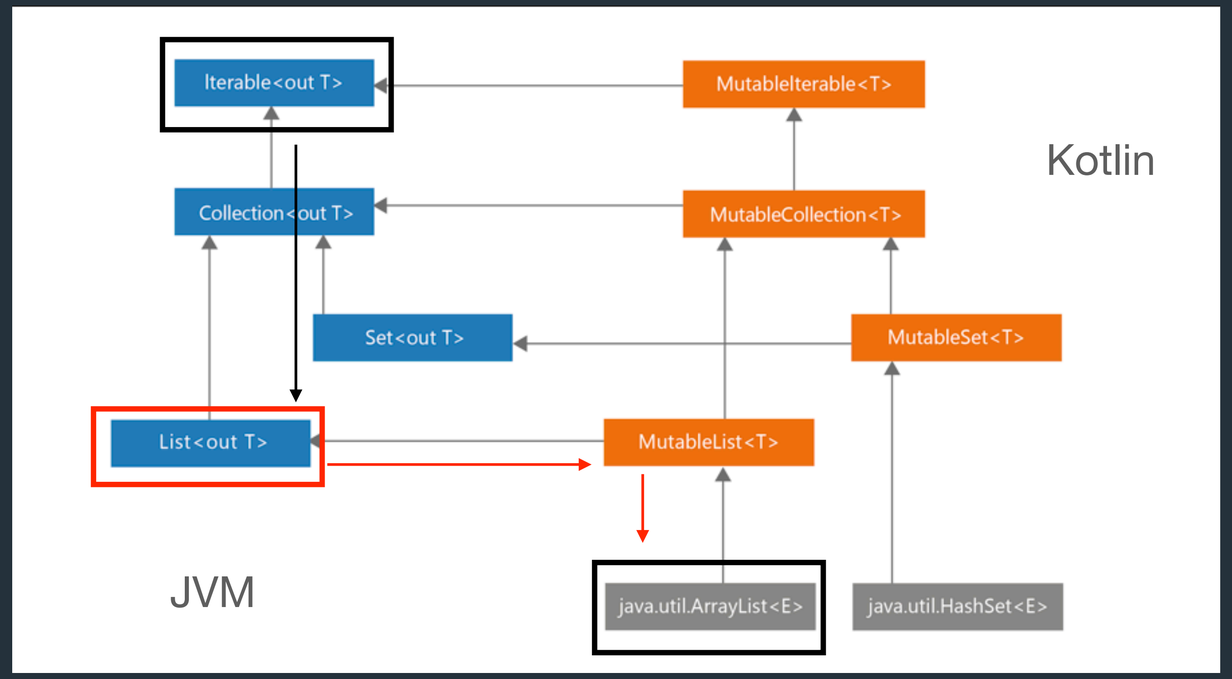
# Down-casting
## (Collections)



# Down-casting
## (Collections)

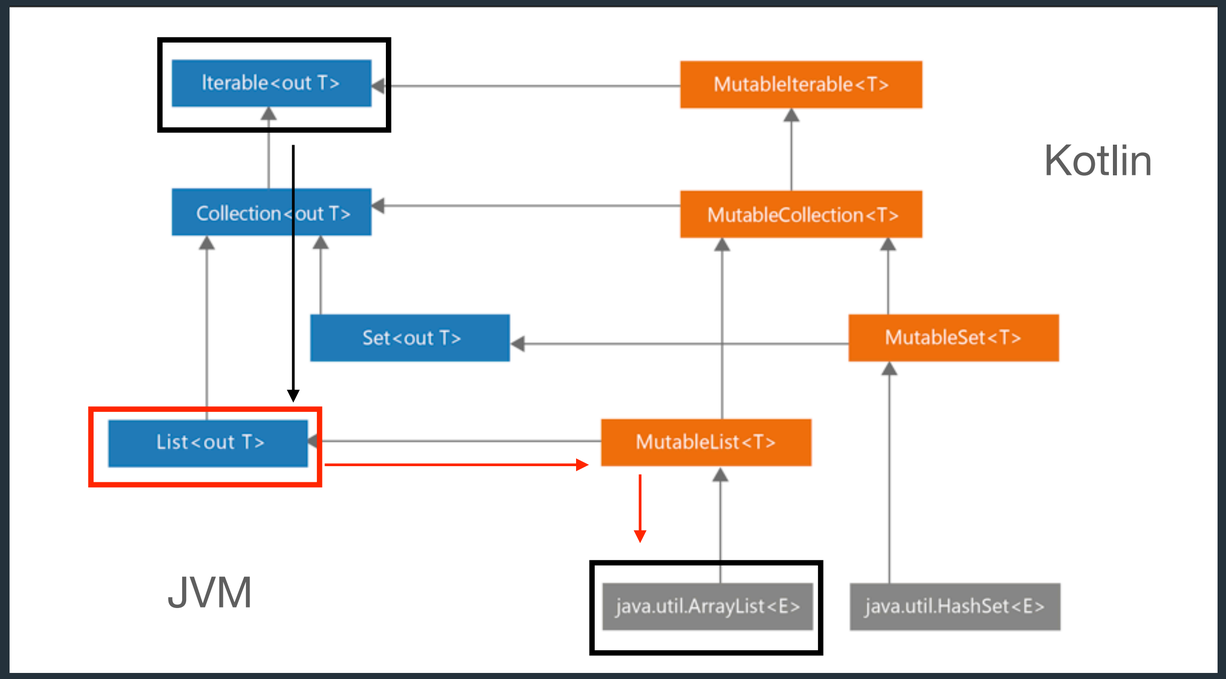


# Down-casting
## (Collections)



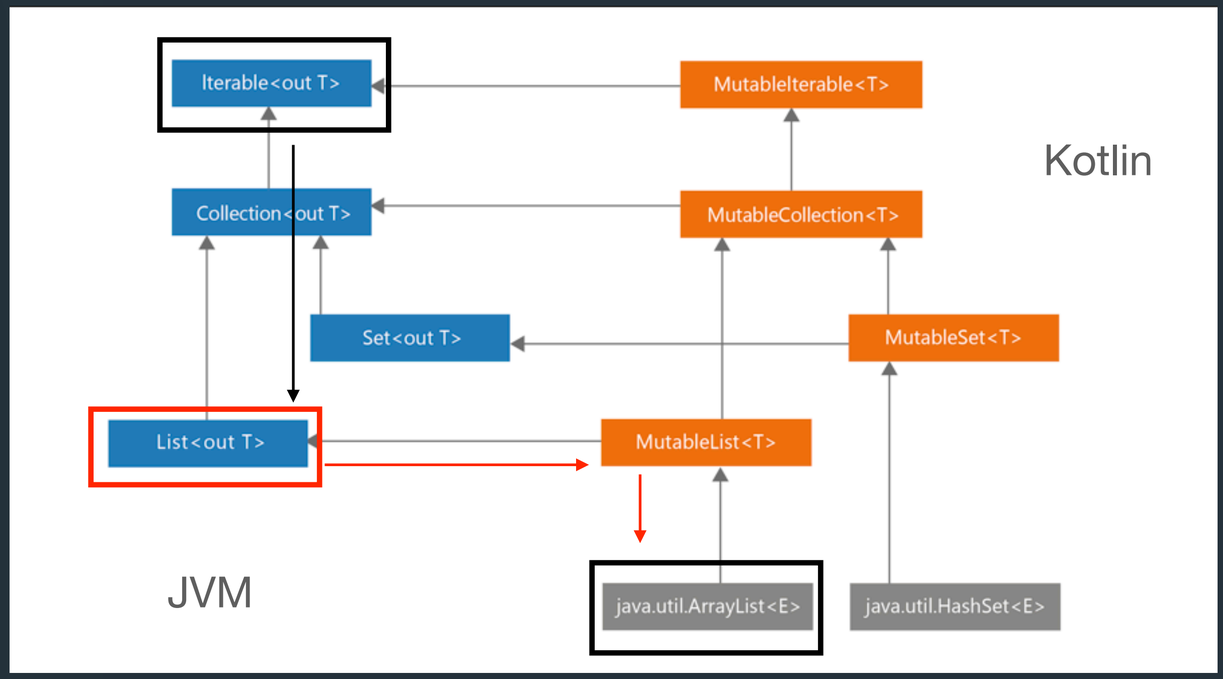
# Down-casting
## (Collections)

```
val listaSoloLectura = listOf("A", "B", "C")

fun hacerDownCasting() {
    //No, no, no, no y ¡NO!
    if (listaSoloLectura is MutableList) {
        listaSoloLectura.add("D")
    }
}
```



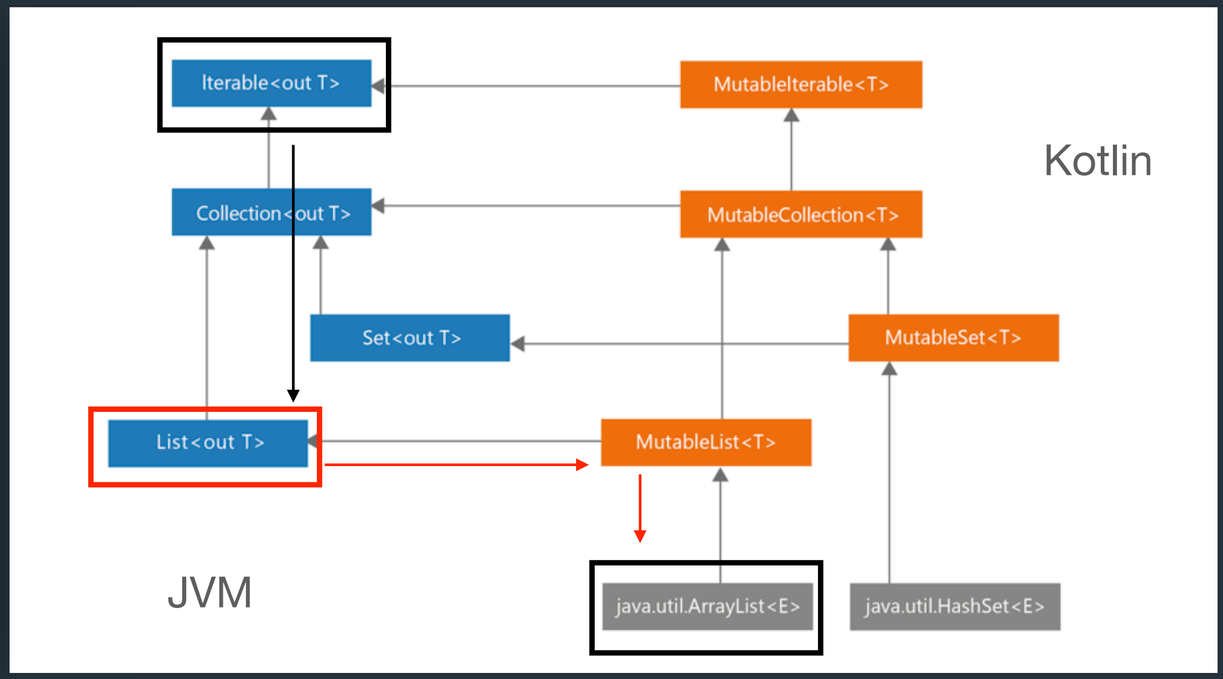
# Down-casting
## (Collections)

```
val listaSoloLectura = listOf("A", "B", "C")

fun hacerDownCasting() {
    //No, no, no, no y ¡NO!
    if (listaSoloLectura is MutableList) {
        listaSoloLectura.add("D")
    }
}

/**
* JVM
*
“Exception in thread “main” java.lang.UnsupportedOperationException
 at java.util.AbstractList.add(AbstractList.java:148)
 at java.util.AbstractList.add(AbstractList.java:108)”
**/
```



# Down-casting
## (Collections)

```kotlin
val listaSoloLectura = listOf("A", "B", "C")

fun sinHacerDownCasting() {
    //Así si :D
    val listaLecturaEscritura = listaSoloLectura.toMutableList()
    listaLecturaEscritura.add("D")
    //copia mutable
}

/**
* JVM
*

 NO EXCEPTION ^_^

**/
```




#KHE

# DefensiveCopying
## Copiado de defensa

Chiste local mexicano

# DefensiveCopying
## Copiado de defensa

( Coff coff )

[let] es para un scope local

Chiste local mexicano

# DefensiveCopying
## Copiado de defensa

```
data class Usuario(
        val nombre: String,
        val apellido: String
)

fun copiadoDefensivo() {
    var usuario = Usuario("Mau", "Bocanegra")
    usuario = usuario.copy(nombre = "Mauro")
    println(usuario) // User(nombre=Mauro apellido=Bocanegra)
}
```

# DefensiveCopying
## Copiado de defensa

# DefensiveCopying
## Copiado de defensa

```
class RepositorioUsuarios{
    private val usuariosDB : MutableMap<Int, String> = mutableMapOf()
}
```

# DefensiveCopying
## Copiado de defensa

```
class RepositorioUsuarios{
    private val usuariosDB : MutableMap<Int, String> = mutableMapOf()

  fun obtenerTodos(): MutableMap<Int, String> (el anterior){
    return usuariosDB
  }
}
```

# DefensiveCopying
## Copiado de defensa

```
class RepositorioUsuarios{
    private val usuariosDB : MutableMap<Int, String> = mutableMapOf()

  fun obtenerTodos(): MutableMap<Int, String> (el anterior){
    return usuariosDB
  }
}

fun agregarUsuarioMalamente(){
    val repoUsuarios = RepositorioUsuarios()
    val usuarios = repoUsuarios.obtenerDB()
    usuarios[4] = "asdf"
    println(repoUsuarios.obtenerDB()) // [4] = asdf :(
}
```

# DefensiveCopying
## Copiado de defensa

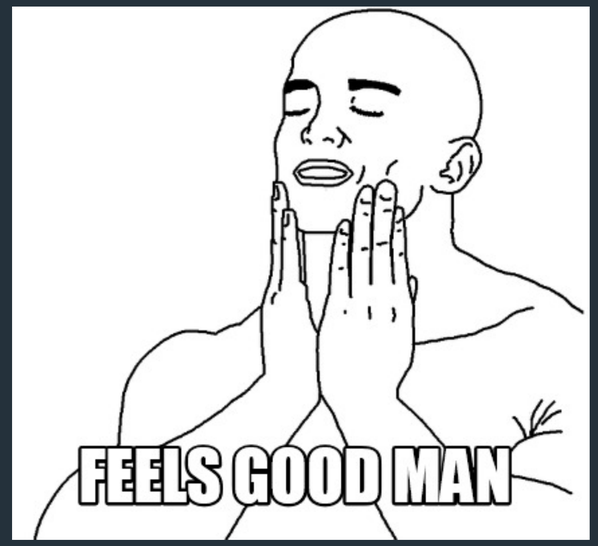
```
class RepositorioUsuarios{
    private val usuariosDB : MutableMap<Int, String> = mutableMapOf()

  fun obtenerCopiaDef(): MutableMap<Int, String> (el anterior){
    return usuariosDB.toMutableMap()
  }

  fun obtenerSuperTipo(): Map<Int, String> (el anterior){
    return usuariosDB
  }
}
```

# DefensiveCopying
## Copiado de defensa

```
class RepositorioUsuarios{
    private val usuariosDB : MutableMap<Int, String> = mutableMapOf()

  fun obtenerCopiaDef(): MutableMap<Int, String> (el anterior){
    return usuariosDB.toMutableMap()
  }
}
```

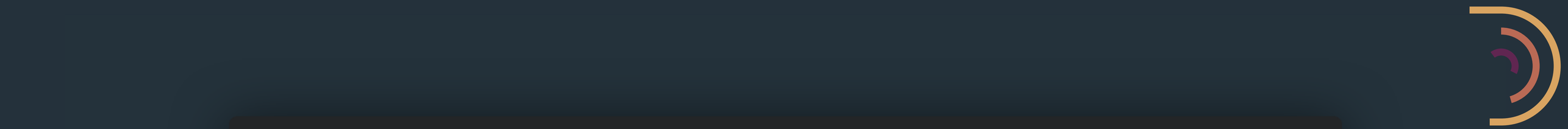
¿Conciso?

~~¿Conciso?~~

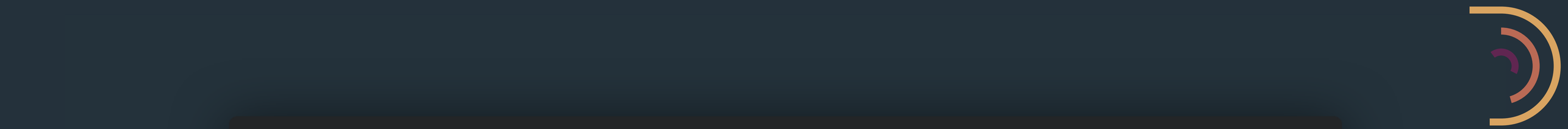



# Legible

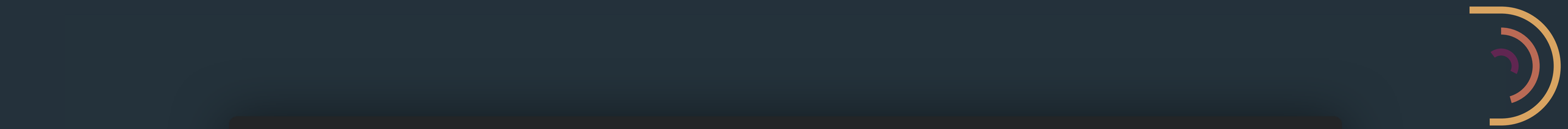
```
fun infUsr(u: Persona?, v: View){
    u?.takeIf { it.esAdulto }
            ?.let(v::mostrarDatos)
            ?: v.mostrarError()
```



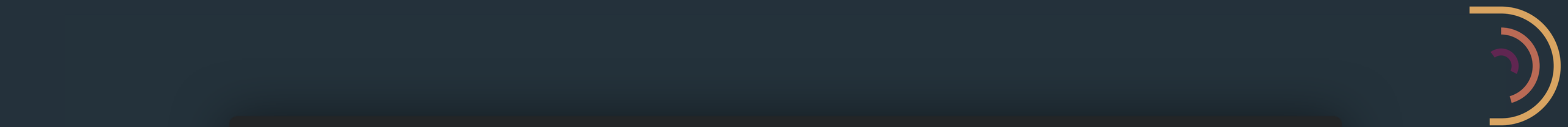
```
fun infUsr(u: Persona?, v: View){
    u?.takeIf { it.esAdulto }
            ?.let(v::mostrarDatos)
            ?: v.mostrarError()
```



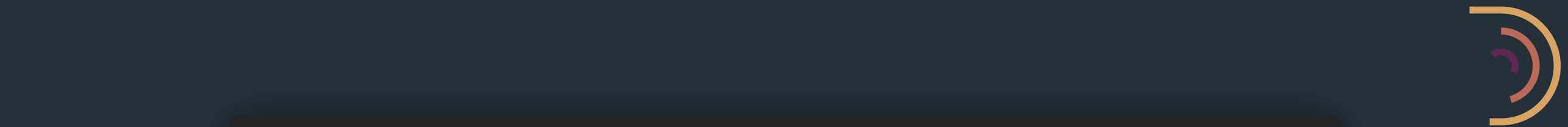
```
fun infUsr(u: Persona?, v: View){
    u?.takeIf { it.esAdulto }
            ?.let(v::mostrarDatos)
            ?: v.mostrarError()
```



```
fun infUsr(u: Persona?, v: View){
    u?.takeIf { it.esAdulto }
            ?.let(v::mostrarDatos)
            ?: v.mostrarError()
```



```
fun infUsr(u: Persona?, v: View){
    u?.takeIf { it.esAdulto }
            ?.let(v::mostrarDatos)
            ?: v.mostrarError()
```



```
fun infUsr(u: Persona?, v: View){
    u?.takeIf { it.esAdulto }
            ?.let(v::mostrarDatos)
            ?: v.mostrarError()
```

```kotlin
fun infUsr(u: Persona?, v: View){
    u?.takeIf { it.esAdulto }
            ?.let(v::mostrarDatos)
            ?: v.mostrarError()

fun mostrarInfoUsuario(usuario : Persona?, view: View){
    if(usuario!=null && usuario.esAdulto){
        view.mostrarDatos(usuario)
    } else{
        view.mostrarError()
    }
}
```

```
fun infUsr(u: Persona?, v: View){
    u?.takeIf { it.esAdulto }
            ?.let(v::mostrarDatos)
            ?: v.mostrarError()

fun mostrarInfoUsuario(usuario : Persona?, view: View){
    if(usuario!=null && usuario.esAdulto){
        view.mostrarDatos(usuario)
    } else{
        view.mostrarError()
    }
}

// en infUsr mostrarError() se ejecuta si usuario==null
// mientras que en mostraInfoUsuario, no
```

# Desde las sombras

# Desde las sombras

```
interface Figura
class Cuadrado: Figura
class Circulo: Figura
```

# Desde las sombras

```
interface Figura
class Cuadrado: Figura
class Circulo: Figura

class BaseDatosFiguras<T: Figura>{
    fun <T: Figura> agregarFiguraConAristas(figura: T) {
    //...
    }
}
```

# Desde las sombras

```
interface Figura
class Cuadrado: Figura
class Circulo: Figura

class BaseDatosFiguras<T: Figura>{
    fun <T: Figura> agregarFiguraConAristas(figura: T) {
    //...
    }
}

fun agregandoAlgunasFiguras(){

}
```

# Desde las sombras

```
interface Figura
class Cuadrado: Figura
class Circulo: Figura

class BaseDatosFiguras<T: Figura>{
    fun <T: Figura> agregarFiguraConAristas(figura: T) {
    //...
    }
}

fun agregandoAlgunasFiguras(){
    val bdFiguras = BaseDatosFiguras<Figura>()
}
```

# Desde las sombras

```
interface Figura
class Cuadrado: Figura
class Circulo: Figura

class BaseDatosFiguras<T: Figura>{
    fun <T: Figura> agregarFiguraConAristas(figura: T) {
        // ...
    }
}

fun agregandoAlgunasFiguras(){
    val bdFiguras = BaseDatosFiguras<Cuadrado>()
    bdFiguras.agregarFiguraConAristas(Cuadrado())
}
```

# Desde las sombras

```
interface Figura
class Cuadrado: Figura
class Circulo: Figura

class BaseDatosFiguras<T: Figura>{
    fun <T: Figura> agregarFiguraConAristas(figura: T) {
        // ...
    }
}

fun agregandoAlgunasFiguras(){
    val bdFiguras = BaseDatosFiguras<Cuadrado>()
    bdFiguras.agregarFiguraConAristas(Cuadrado())
    bdFiguras.agregarFiguraConAristas(Circulo())
}
```

# Desde las sombras

```
interface Figura
class Cuadrado: Figura
class Circulo: Figura

class BaseDatosFiguras<T: Figura>{
    fun <T: Figura> agregarFiguraConAristas(figura: T) {
        // ...
    }
}

fun agregandoAlgunasFiguras(){
    val bdFiguras = BaseDatosFiguras<Cuadrado>()
    bdFiguras.agregarFiguraConAristas(Cuadrado())
    bdFiguras.agregarFiguraConAristas(Circulo())
}
```

# Desde las sombras

```
interface Figura
class Cuadrado: Figura
class Circulo: Figura

class BaseDatosFiguras<T: Figura>{
    fun <T: Figura> agregarFiguraConAristas(figura: T) {
        // ...
    }
}

fun agregandoAlgunasFiguras(){
    val bdFiguras = BaseDatosFiguras<Cuadrado>()
    bdFiguras.agregarFiguraConAristas(Cuadrado())
    bdFiguras.agregarFiguraConAristas(Circulo())
}
```

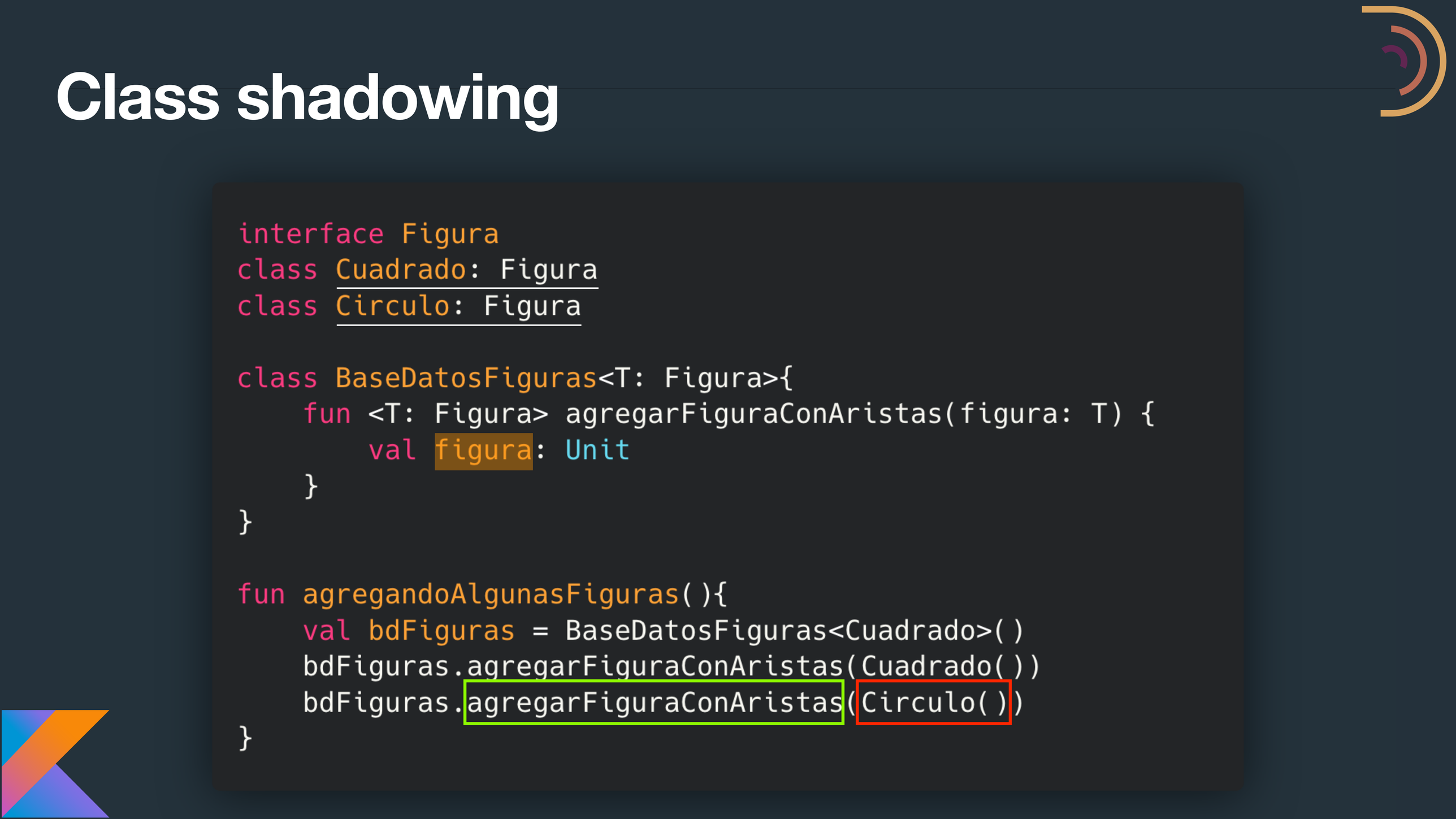
# Class shadowing

```
interface Figura
class Cuadrado: Figura
class Circulo: Figura

class BaseDatosFiguras<T: Figura>{
    fun <T: Figura> agregarFiguraConAristas(figura: T) {
        // ...
    }
}

fun agregandoAlgunasFiguras(){
    val bdFiguras = BaseDatosFiguras<Cuadrado>()
    bdFiguras.agregarFiguraConAristas(Cuadrado())
    bdFiguras.agregarFiguraConAristas(Circulo())
}
```

# Class shadowing

```
interface Figura
class Cuadrado: Figura
class Circulo: Figura

class BaseDatosFiguras<T: Figura>{
    fun <T: Figura> agregarFiguraConAristas(figura: T) {
        val figura: Unit
    }
}

fun agregandoAlgunasFiguras(){
    val bdFiguras = BaseDatosFiguras<Cuadrado>()
    bdFiguras.agregarFiguraConAristas(Cuadrado())
    bdFiguras.agregarFiguraConAristas(Circulo())
}
```

# Class shadowing



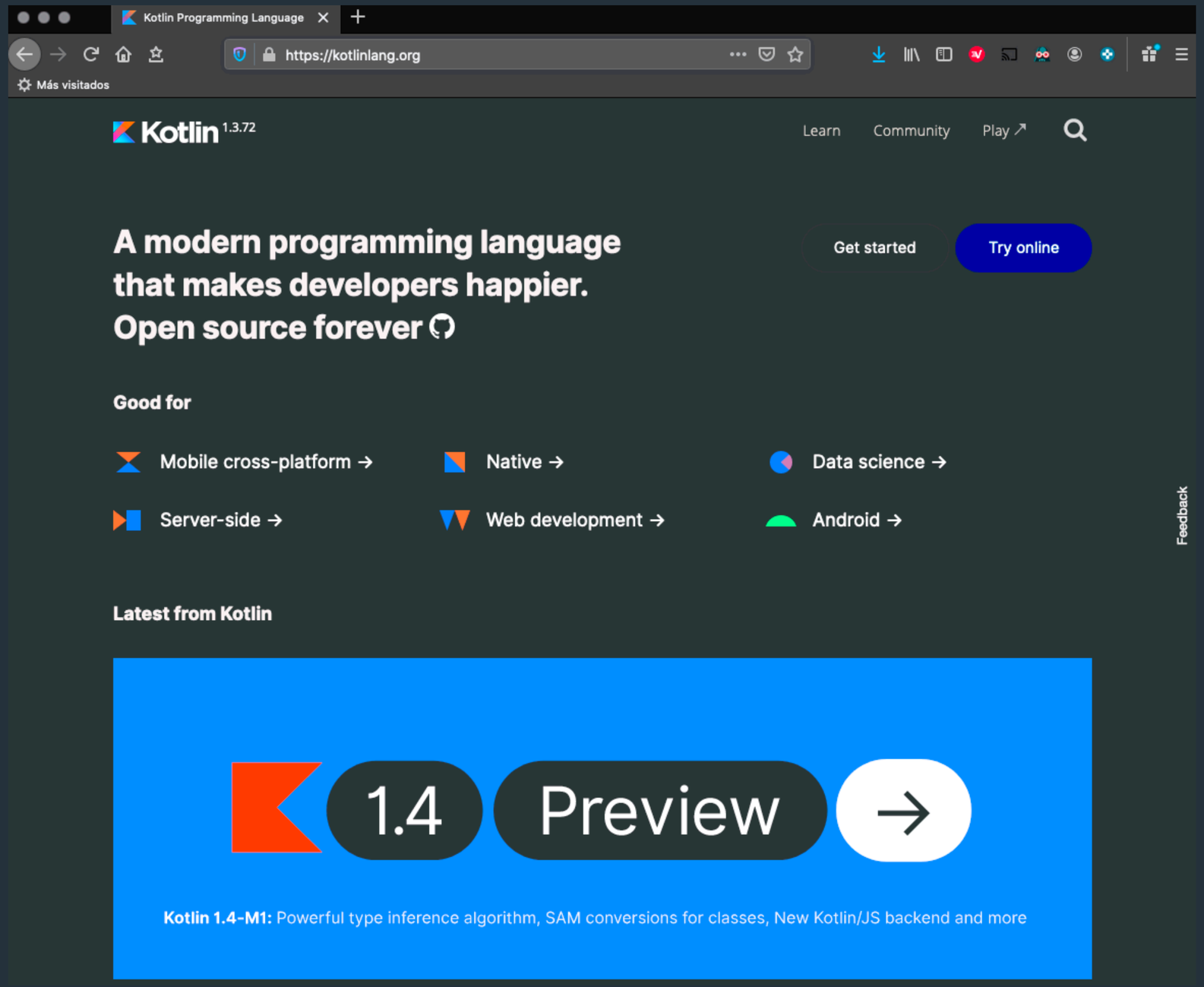
```
interface Figura
class Cuadrado: Figura
class Circulo: Figura

class BaseDatosFiguras<T: Figura>{
    fun <T: Figura> agregarFiguraConAristas(figura: T) {
        val figura: Unit
    }
}

fun agregandoAlgunasFiguras(){
    val bdFiguras = BaseDatosFiguras<Cuadrado>()
    bdFiguras.agregarFiguraConAristas(Cuadrado())
    bdFiguras.agregarFiguraConAristas(Circulo())
}
```

# Class shadowing

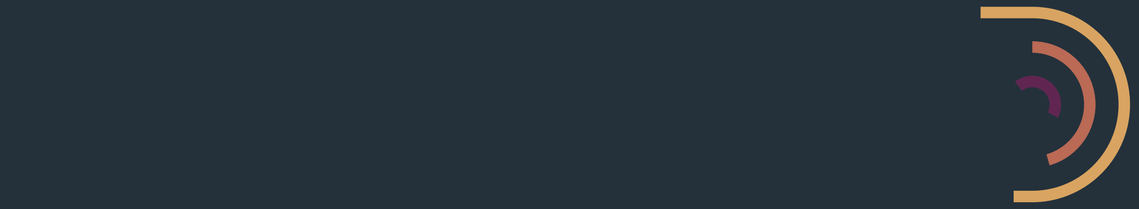
```
interface Figura
class Cuadrado: Figura
class Circulo: Figura

class BaseDatosFiguras<T: Figura>{
    fun <T: Figura> agregarFiguraConAristas(figura: T) {
        // ...
    }
}

fun agregandoAlgunasFiguras(){
    val bdFiguras = BaseDatosFiguras<Cuadrado>()
    bdFiguras.agregarFiguraConAristas(Cuadrado())
    bdFiguras.agregarFiguraConAristas(Circulo())
}
```

# Class shadowing

```
interface Figura
class Cuadrado: Figura
class Circulo: Figura

class BaseDatosFigurasConAristas<T: Figura>{
    fun <T: Figura> agregarFiguraConAristas(figura: T) {
        // ...
    }
}

fun agregandoAlgunasFiguras(){
    val bdFiguras = BaseDatosFigurasConAristas<Cuadrado>()
    bdFiguras.agregarFiguraConAristas(Cuadrado())
    bdFiguras.agregarFiguraConAristas(Circulo())
}
```


Kotlin Programming Language
https://kotlinlang.org
Más visitados
Kotlin 1.3.72
Learn
Community
Play ↗
A modern programming language that makes developers happier.
Open source forever
Get started
Try online
Good for
Mobile cross-platform →
Native →
Data science →
Server-side →
Web development →
Android →
Feedback
Latest from Kotlin
1.4
Preview
Kotlin 1.4-M1: Powerful type inference algorithm, SAM conversions for classes, New Kotlin/JS backend and more

# ¿Comentarios?
# ¿Preguntas?
# ¿Correcciones?



# ¡GRACIAS! :D

@MauBocanegra

linkedin.com/in/maubocanegra/