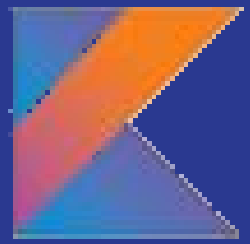
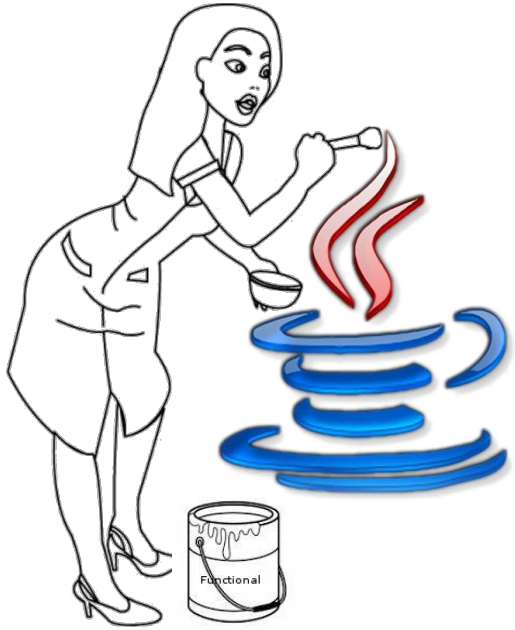


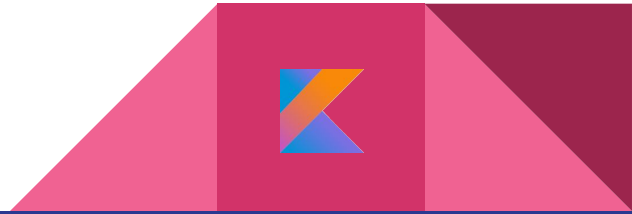
Kotlin

@malkomich


Functional

VENTAJAS

```java
public class JavaPerson {

    private String name;
    private String surname;
    private int age;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getSurname() {
        return surname;
    }

    public void setSurname(String surname) {
        this.surname = surname;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    @Override
    public String toString() {
        return "Person{" +
                "name='" + name + "'" +
                ", surname='" + surname + "'" +
                ", age=" + age +
                "}";
    }

}
```



VENTAJAS

```
public class JavaPerson {

    private String name;
    private String surname;
    private int age;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getSurname() {
        return surname;
    }

    public void setSurname(String surname) {
        this.surname = surname;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    @Override
    public String toString() {
        return "Person{" +
                "name='" + name + "'" +
                ", surname='" + surname + "'" +
                ", age=" + age +
                "}";
    }

}
```

VENTAJAS

```java
public class JavaPerson {

    private String name;
    private String surname;
    private int age;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getSurname() {
        return surname;
    }

    public void setSurname(String surname) {
        this.surname = surname;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    @Override
    public String toString() {
        return "Person{" +
                "name='" + name + "'" +
                ", surname='" + surname + "'" +
                ", age=" + age +
                "}";
    }

}
```

```kotlin
data class KotlinPerson(
        var name: String,
        var surname: String,
        var age: Integer)
```

Escribir más con menos código

VENTAJAS

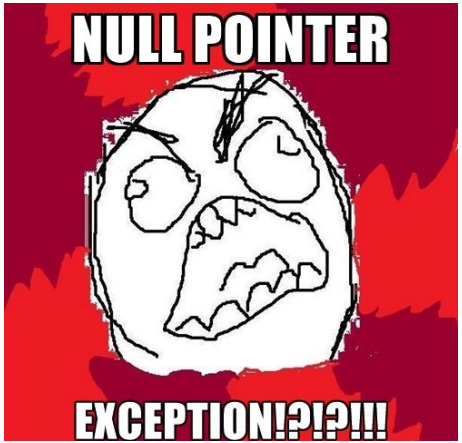
Expresividad

```java
public class JavaPerson {

    private String name;
    private String surname;
    private int age;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getSurname() {
        return surname;
    }

    public void setSurname(String surname) {
        this.surname = surname;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    @Override
    public String toString() {
        return "Person{" +
                "name='" + name + "'" +
                ", surname='" + surname + "'" +
                ", age=" + age +
                "}";
    }

}
```

```kotlin
data class KotlinPerson(
        var name: String,
        var surname: String,
        var age: Integer)
```

VENTAJAS

Expresividad



VENTAJAS

Kotlin es *Null Safe*

VENTAJAS

NullPointerException pasa a ser un error en tiempo de compilación:

```
var notNullPerson: KotlinPerson = null
```

```
var person: KotlinPerson? = null
```

VENTAJAS

NullPointerException pasa a ser un error en tiempo de compilación:

```
person.print()
```

```
person?.print()
```

Sólo se ejecuta si *person* no es NULL

VENTAJAS

- Expresividad
- Null Safe

También podemos usar el *Elvis operator*, de forma similar al operador ternario en Java:

```
val name = person?.name ?: "empty"
```

```
String name;
if(person != null) {
    name = person.getName();
} else {
    name = "empty";
}
```

VENTAJAS

- Expresividad
- Null Safe

VENTAJAS

- Expresividad
- Null Safe

VENTAJAS

- Expresividad
- Null Safe

Extend unknown classes functionality:

```
fun Fragment.toast(message: CharSequence, duration: Int = Toast.LENGTH_SHORT) {
    Toast.makeText(getActivity(), message, duration).show()
}
```

VENTAJAS

- Expresividad
- Null Safe

Extend unknown classes functionality:

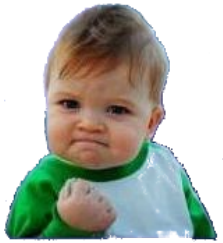
```
fragment.toast("Hello world")
```

```
fun Fragment.toast(message: CharSequence, duration: Int = Toast.LENGTH_SHORT) {
    Toast.makeText(getActivity(), message, duration).show()
}
```

VENTAJAS

- Expresividad
- Null Safe
- Extension functions

```
fragment.toast("Hello world")
```



VENTAJAS

Expresividad
Null Safe
Extension functions

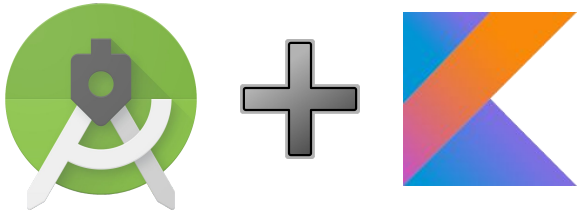
Expresividad

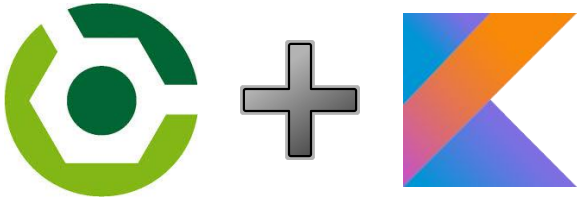
Null Safe

Extension functions

REQUISITOS

PLUGINS:





SOBRE LAS VARIABLES

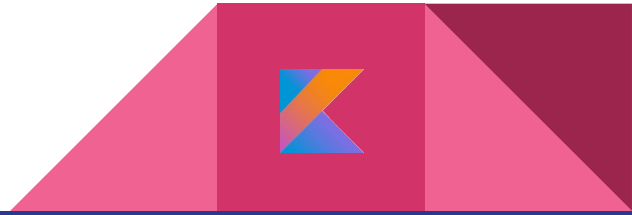
El tipado sigue siendo estático, como en Java,

pero...

permite inferencia de tipos

```
val name = "Juan Carlos"
val name: String = "Juan Carlos"
```

- **Tipado estático**
- **Inferencia de tipos**



SOBRE LAS VARIABLES

En Kotlin, las variables se declaran de dos formas:

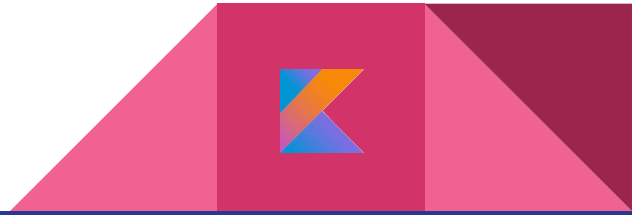
MUTABLE

```
var name: String = "Juan Carlos"
```

INMUTABLE

```
val name = "Juan Carlos"
```

- **Tipado estático**
- **Inferencia de tipos**
- **Nueva sintaxis, misma función**



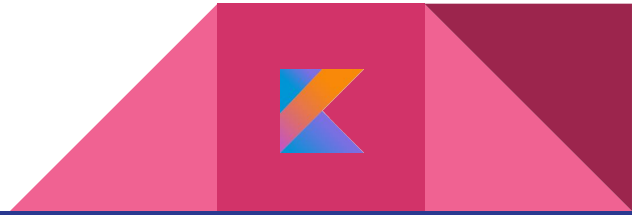
SOBRE LAS VARIABLES

Kotlin quiere que por defecto usemos variables inmutables, por lo que añade a las clases del dominio (*data class*) una nueva función:

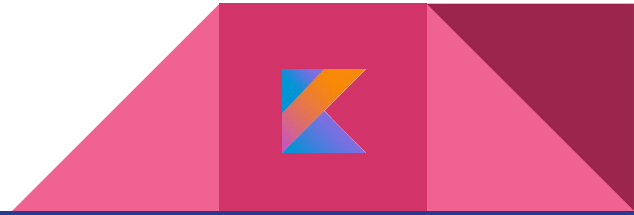
COPY()

```
val forecast: KotlinForecast = KotlinForecast(Date(), 21.5f, "OK")

val forecastCopy1 = forecast.copy()
val forecastCopy2 = forecast.copy(temperature = 29f, details = "TOO HOT")
```

- **Tipado estático**
- **Inferencia de tipos**
- **Nueva sintaxis, misma función**
- **copy(), una manera más simple de construir instancias similares**



MÁS NOVEDADES...



Semicolon inference

- ~~new~~ Object() → Object()
- object.getX() → object.x

Inline functions

```
override fun getItemCount(): Int = items.size
```

Binding en Strings:

```
val name = "Juan Carlos"
println("Hola, mi nombre es $name")
println("El producto de 20x5 es ${20*5}")
```

~~Sobrecarga de métodos~~ → valores por defecto

```
fun method1(mandatoryParam: String, optionalParam: Int = 1) {
    // DO WHATEVER
}
```

~~Setter, Getter~~ → Parámetros en definición de la clase

Public visibility by default

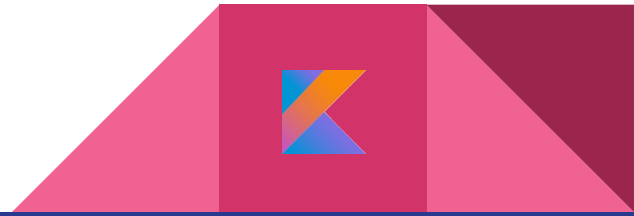
~~Static properties/functions~~ → companion object

```
companion object {
    private val CONSTANT = 0

    public fun staticMethod(param: String) : Int = CONSTANT
}
```

~~Primitive types~~

~~Casting~~ → Explicit conversions

== ⟺ equals()

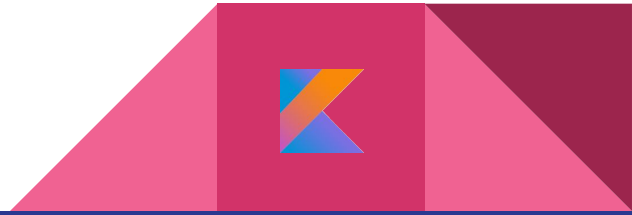


Remember... all I'm offering is the truth. Nothing more.

Pero...¿y si quiero elegir ambos?

INTEROPERABILIDAD CON JAVA:

- **Calling Java code from Kotlin** ✓
- **Calling Kotlin code from Java** ✓
- **Projects with both Java & Kotlin code** ✓

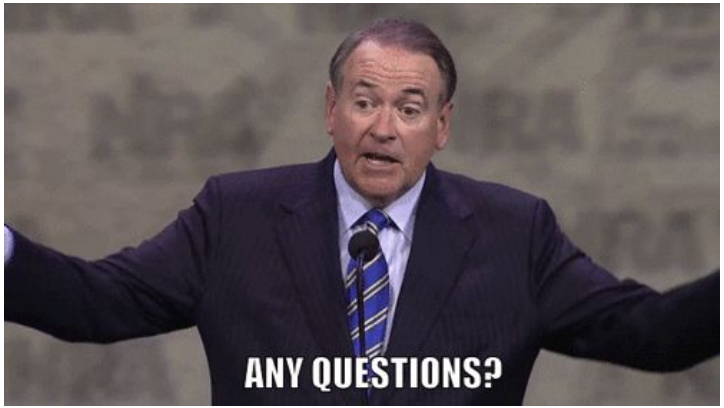


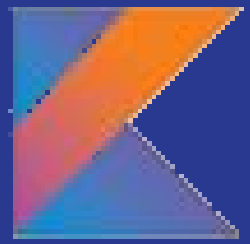
Pero...¿y si quiero elegir ambos?

INTEROPERABILIDAD CON JAVA:

REFERENCE: http://kotlinlang.org/docs/reference/

Para trastear con ejemplos: http://try.kotlinlang.org/





Kotlin

@malkomich