



The power of types

Gerardo Suárez | Kotlinconf Medellin 2020

<https://www.linkedin.com/in/gerardosuarezmejia/>

Disfrutas hacer debugging?

En un mundo ideal



En un mundo ideal

En el lenguaje correcto



En un mundo ideal

En el lenguaje correcto

Escribir código con bugs es imposible



En un mundo ideal

En el lenguaje correcto

Escribir código con bugs es imposible

Se generan errores de compilación!



Ningún lenguaje puede prevenirnos de escribir
bugs!

Halting Problem - Alan Turing

Pero podemos prevenirlos desde una etapa temprana

TYPES!

The type system is Kotlin's super power:

- Nullability
- 1st class functions
- Nothing



```
val user: String  
val phoneNo: String
```



Usar primitivos en vez de objetos pequeños para tareas simples



```
const val USER_ADMIN_ROLE = 1
```



Usar constantes para mantener información del código



```
val time: Long
```



```
val timeMs: Long
```



```
data class Time(val timeMs: Long)
```



```
data class Time(val value: Long, val unit: TimeUnit)
```


Regla de ORO

No pongas en el nombre lo que puede ir en el tipo!

Encoding the type `[..]` into the name `[..]` is brain damaged the compiler knows the types

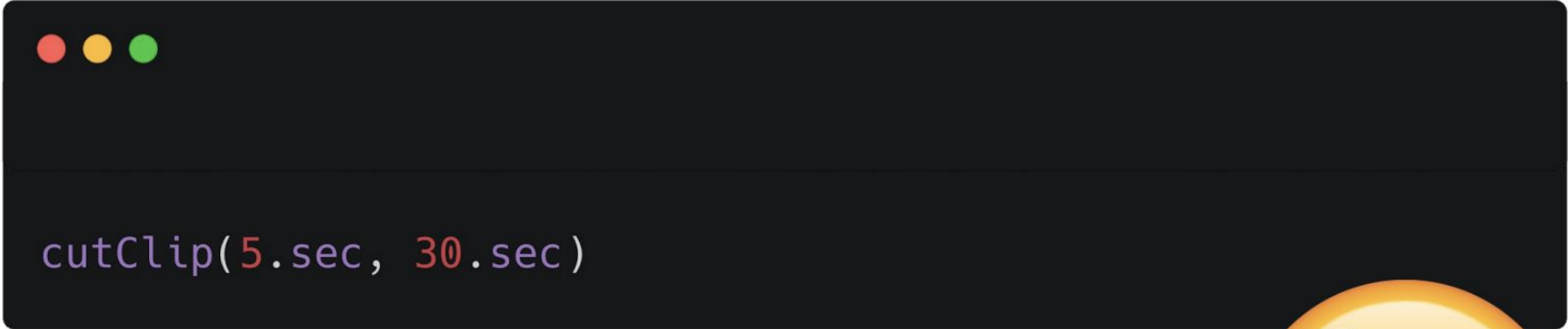
Linus Torvalds about Hungarian notation



```
suspend fun cutClip(start: Time, duration: Time): ClipId
```



```
suspend fun cutClip(start: Time, duration: Time): ClipId  
  
cutClip(5.sec, 30.sec)
```



```
cutClip(5.sec, 30.sec)
```





```
data class PointInTime(val value: Time)
data class TimeDelta(val value: Time)

suspend fun cutClip(start: PointInTime, duration: TimeDelta): ClipId
```



```
data class PointInTime(val value: Time)
data class TimeDelta(val value: Time)

suspend fun cutClip(start: PointInTime, duration: TimeDelta): ClipId
suspend fun cutClip(start: PointInTime, end: PointInTime): ClipId
```

En realidad:



```
package kotlin.time
```

```
@SinceKotlin("1.3")
```

```
@ExperimentalTime
```

```
inline class Duration(val value: Double)
```





```
fun priceInfo(regularPrice: Price): CharSequence  
fun discountPriceInfo(discountPrice: Price): CharSequence
```

No pongas el nombre donde puedes poner el type!



```
fun priceInfo(price: RegularPrice): CharSequence  
fun priceInfo(price: DiscountPrice): CharSequence
```



```
data class Comment(  
  
    val isReply: Boolean,  
  
)
```

2 tipos cosas en un solo lugar, comentarios y replies



```
data class Comment(  
  
    val isReply: Boolean,  
  
)
```

Booleans son smells

COSTOS

Allocation costs

Allocations are OK

Types are OK

Yes, even enums

Modern Android development: Android Jetpack, Kotlin, and more (Google I/O 2018)

Garbage Collection

The performance of the garbage collector is not determined by the number of dead objects but rather by the number of live ones

Analyzing Java Memory, Java enterprise
performance book

Short lived objects are cheap

Memory



```
val age: Int
```

El compilador lo traduce a *Int*



```
val age: Int?
```

El compilador lo traduce a *Integer*



```
class Age(val age: Int)
```

```
val age: Age?
```

Se crea un objeto y por dentro es un *Integer*



```
inline class Age(val age: Int)
```

```
val age: Age
```

El compilador lo traduce a *Int*

Conclusiones

- Tipos ayudan a mejorar el entendimiento del código - Readability - Clean Code
- Tipos ayudan a evitar bugs a través del compilador (safety, machine checked documentation)
- Tipos ayudan a la abstracción (modularity)

Revisa en tu código errores comunes:

- Información de los tipos en el nombre
- Booleans
- Primitivos

Permitamos que el compilador nos
ayude!