

Options



The Billion-Dollar Mistake

"I call it my billion dollar mistake. It was the invention of the null reference in 1965. [...] But I couldn't resist the temptation to put in a null reference, simply because it was too easy to implement."

(Tony Hoare)

```
val string: String = null
println(string.length)
```

← method calls on null references result in NPEs and app crashes

```
Exception in thread "main" java.lang.NullPointerException
  at org.rtjvm.fundb.Main$.delayedEndpoint$org$rtjvm$Main$1(Main.scala:5)
  at org.rtjvm.fundb.Main$delayedInit$body.apply(Main.scala:3)
  ...
```

```
val string: String = null
if (string != null) {
  println(string.length)
}
```

← working with null values leads to spaghetti code

Options

An Option is a wrapper for a value that might be present or not.

```
sealed abstract class Option[+A]  
case class Some[+A](x: A) extends Option[A]  
case object None extends Option[Nothing]
```

- *Some* wraps a concrete value
- *None* is a singleton for absent values

Options are present in many places:

```
val map = Map("key" -> "value")  
map.get("key")    // Some(value)  
map.get("other") // None
```



map uses options on its basic get operation; prefer it over apply

```
val numbers = List(1, 2, 3)  
list.headOption    // Some(1)  
list.find(_ % 2 == 0) // Some(2)
```



lots of functions on all collections work with options

Options: Why and How

Use Options to stay away from the Boogeyman:

- avoid runtime crashes due to NPEs
- avoid an endless amount of null-related assertions

A functional way of dealing with absence

- *map, flatMap, filter*
- *orElse*
- others: *fold, collect, toList*

If you design a method to return a (some type) but may return null, return an Option[that type] instead.

Scala rocks

