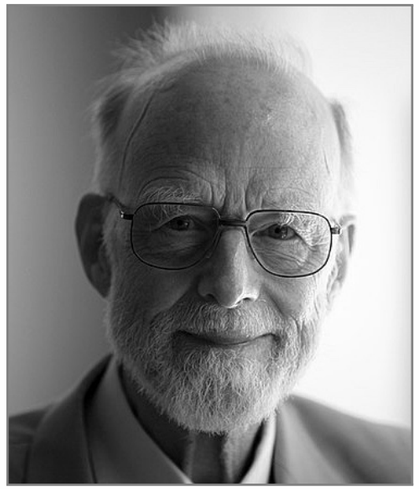
NULL

Yegor Bugayenko

Lecture #6 out of 8
90 minutes

All visual and text materials presented in this slidedeck are either originally made by the author or taken from public Internet sources, such as website. Copyright belongs to their respected authors.



“I call it my billion-dollar mistake. It was the invention of the null reference in 1965. At that time, I was designing the first comprehensive type system for references in an object oriented language (ALGOL W). My goal was to ensure that all use of references should be absolutely safe, with checking performed automatically by the compiler. But I couldn’t resist the temptation to put in a null reference, simply because it was so easy to implement. This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years.”

— Tony Hoare, 2009

- Fail Fast vs Fail Safe
- Alternatives to Returning NULL
- Alternatives to Checking for NULL
- Read and Watch

Chapter #1:

Fail Fast vs Fail Safe

“Over time, more and more errors will fail fast, and you’ll see the cost of debugging decrease and the quality of your system improve”

— James Shore, IEEE Software, 2002

Calculating file size

Fail Safe:

```
1 int size(File file) {
2   if (!file.exists()) {
3     return 0;
4   }
5   return file.length();
6 }
```

Fail Fast:

```
1 int size(File file) {
2   if (!file.exists()) {
3     throw new IllegalArgumentException(
4       "The file is absent :("
5     );
6   }
7   return file.length();
8 }
```

The right snippet is more fragile, leading to more errors in runtime, but eventually ... leading to less bugs.

Chapter #2:

Alternatives to Returning NULL

Returning NULL or raising an error?

```
1 String nameOfEmployee(int id) {
2   if (em.existsInDb(id)) {
3     return null;
4   }
5   return em.readFromDb(id);
6 }
```

```
1 String nameOfEmployee(int id) {
2   if (em.existsInDb(id)) {
3     throw new EmployeeNotFound(id);
4   }
5   return em.readFromDb(id);
6 }
```

The right snippet is “Fail Fast,” that’s why more preferrable.

Returning a List or a NULL?

```
String nameOfEmployee(int id) {
  if (em.existsInDb(id)) {
    return null;
  }
  return em.readFromDb(id);
}
```

```
List<String> nameOfEmployee(int id) {
  List<String> names =
    new ArrayList<>(0);
  if (em.existsInDb(id)) {
    return names;
  }
  names.add(em.readFromDb(id));
  return names;
}
```

There are mo elegant alternatives in most languages, like `Optional` in Java 8+.

Chapter #3:

Alternatives to Checking for NULL

null-coalescing operator in C#

```
int? sizeOf(File f) {
  if (f.exists()) {
    return null;
  }
  return f.size();
}
int? s = sizeOf(f);
if (s == null) {
  s = 0;
}
```

```
int? sizeOf(File f) {
  if (f.exists()) {
    return null;
  }
  return f.size();
}
int s = sizeOf(f) ?? 0;
```

Both snippets are bad design, though. They are workarounds.

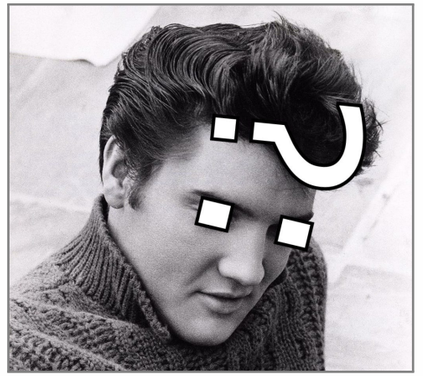
&. operator in Ruby

```ruby
def employee(id)
  unless db.exists?(id)
    return nil
  end
  return db.get(id)
end
e = employee(42)
puts e.name unless e.nil?
```

```ruby
def employee(id)
  unless db.exists?(id)
    return nil
  end
  return db.get(id)
end
puts employee(42)&.name
```

Actually, the snippets produce different output when the employee is not found. How are they different?

NULL-awareness in Kotlin



```
1 var a: String = "abc"
2 a = null // compilation error
3
4 var b: String? = "abc"
5 b = null // no error here
6
7 println(b?.length) // prints what?
8 println(b?.length ?: -1) // Elvis operator
```

Chapter #4:

Read and Watch

Why NULL is Bad? by me

One More Recipe Against NULL by me

Need Robust Software? Make It Fragile

What is Wrong About NULL in OOP? (Webinar #3) by me on YouTube