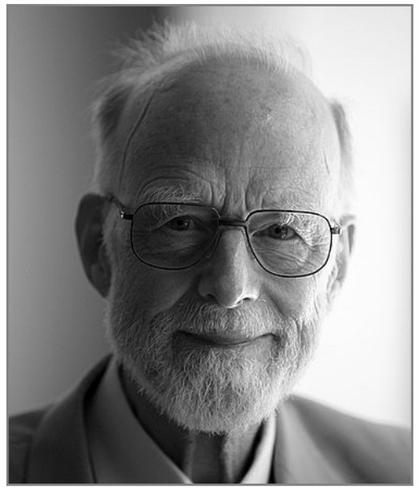
NULL

Fail Fast, Returning, Checking, OT

Yegor Bugayenko

Lecture #6 out of 8
90 minutes



“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, InfoQ, 2009

- Fail Fast vs Fail Safe
- Alternatives to Returning NULL
- Alternatives to Checking for NULL
- Alternatives to Storing NULL
- Object Thinking
- Spring Boot
- 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

Defaults

Fail Safe:

```
int size(File file) {
  if (!file.exists()) {
    return 0;
  }
  return file.length();
}
```

Fail Fast:

```
int size(File file) {
  if (!file.exists()) {
    throw new IllegalArgumentException(
      "The file is absent :("
    );
  }
  return file.length();
}
```

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

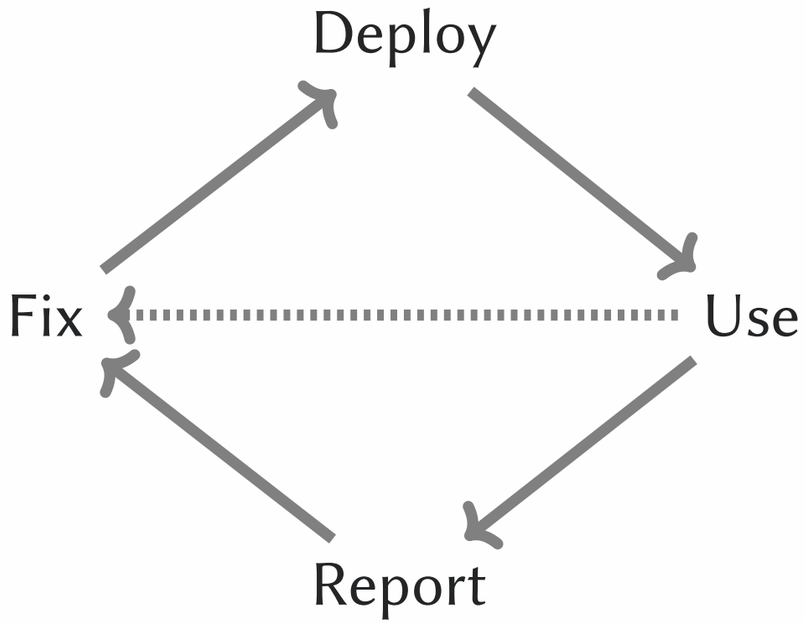
Exception swallowing

```
String read(File file) {
  try {
    return new String(
      Files.readBytes(file)
    );
  } catch (IOException e) {
    e.printStackTrace();
  }
}
```

```
String read(File file) {
  try {
    return new String(
      Files.readBytes(file));
  } catch (IOException e) {
    throw new IllegalStateException(
      String.format(
        "Can't read file %s", e.name()),
      e);
  }
}
```

The right snippet is escalating, while the left one is swallowing.

Software development lifecycle



Watch this video from DEVit'2016 conference:
Need It Robust?Make It Fragile!

Chapter #2:

Alternatives to Returning NULL

Returning NULL or raising an error?

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

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

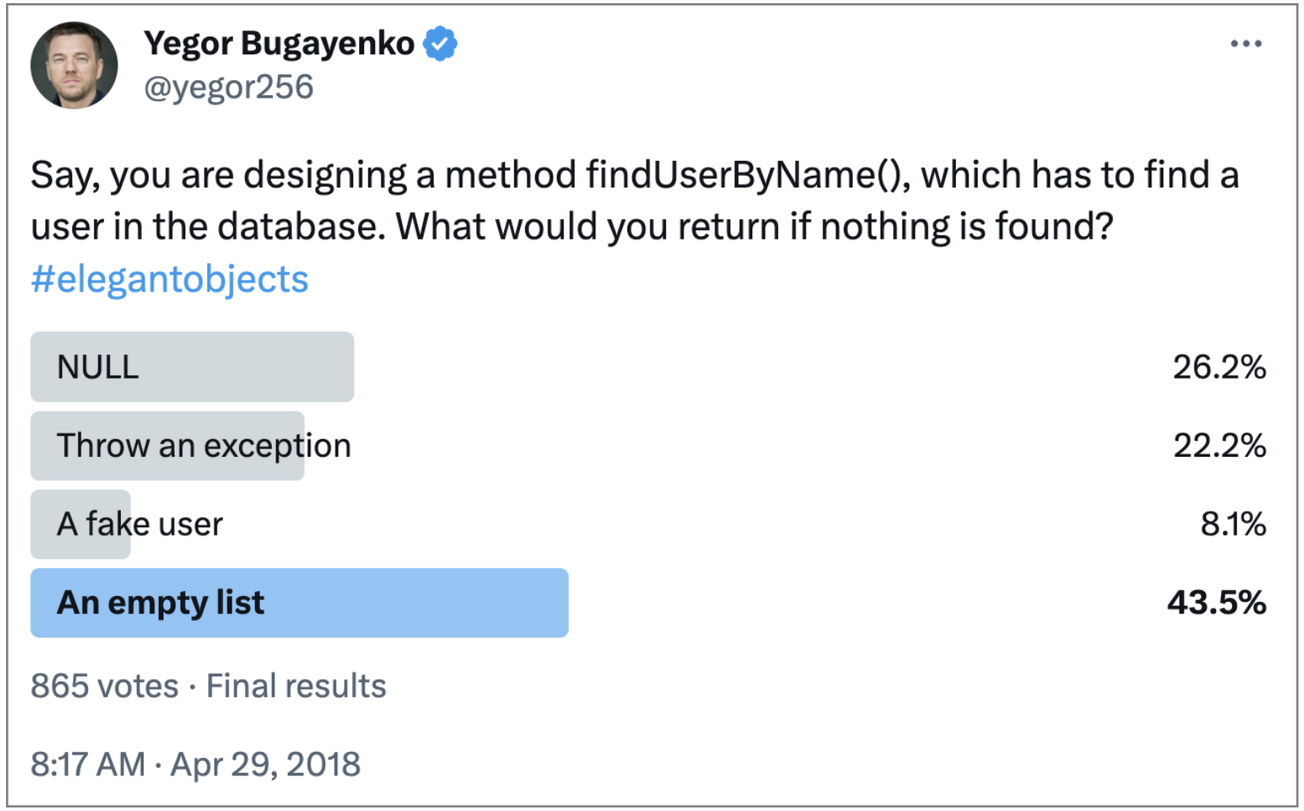
The right snippet is “Fail Fast,” that’s why more preferrable.

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+.


Yegor Bugayenko
@yegor256
Say, you are designing a method findUserByName(), which has to find a user in the database. What would you return if nothing is found?
#elegantobjects
NULL
26.2%
Throw an exception
22.2%
A fake user
8.1%
An empty list
43.5%
865 votes · Final results
8:17 AM · Apr 29, 2018

Returning a Fake Entity

```
Employee employee(int id) {
  if (em.existsInDb(id)) {
    return null;
  }
  return new PgEmployee(id);
}

e = employee(42);
print(e.id());
print(e.salary());
```

```
Employee employee(int id) {
  if (em.existsInDb(id)) {
    return FakeEmployee(id);
  }
  return new PgEmployee(id);
}

e = employee(42);
print(e.id());

print(e.salary());
```

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.

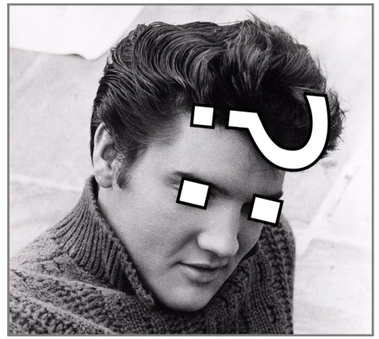
&. operator in Ruby

```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:

Alternatives to Storing NULL

Immutable objects

```
1 class Employee {
2   private String name = null;
3   void setName(String n) {
4     this.name = n;
5   }
6 }
7
8 e = new Employee();
9 e.setName("Jeff");
```

```
1  class Employee {
2    private final String name;
3    Employee(String n) {
4      this.name = n;
5    }
6    Employee withName(String n) {
7      return new Employee(n);
8    }
9  }
10
11 e1 = new Employee();
12 e2 = e1.withName("Jeff");
```

Chapter #5:

Object Thinking

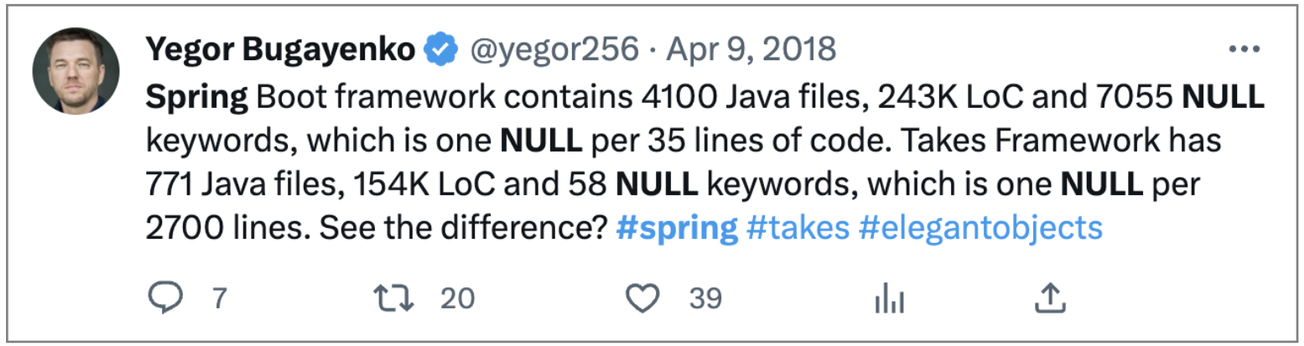
Pay respect to your objects!

```
d = getDepartment(42);
e = d.getEmployee("Jeff");
if (e != null) {
  printf("Hello, %s", e.name());
}
```

```
- Hello, is it the department no.42?
- Yes.
- Let me talk to your employee "Jeff".
- Hold the line please...
- Hello.
- Are you NULL?
```

Chapter #6:

Spring Boot



You can do you own analysis of existing Java open source GitHub repositories to see how often their developers use `null` keyword.

The Takes framework is here: yegor256/takes.

Chapter #7:

Read and Watch

Why NULL is Bad? by me (2014)

Throwing an Exception Without Proper Context Is a Bad Habit by me (2015)

One More Recipe Against NULL by me (2018)

Need Robust Software? Make It Fragile by me (2015)

What is Wrong About NULL in OOP? (Webinar #3) by me on YouTube (2015)