AARHUS UNIVERSITET

Software Engineering and Architecture

Pattern Catalog: Null Object

AARHUS UNIVERSITET

Motivation

- Implementation of lengthy operation, but…
  - To improve usability we show a progress bar ☺


```
public  void lengthyExecution(ProgressBar progress) {
  // ... do stuff
  progress.report( i: 10);
  // ... do stuff
  progress.report( i: 50);
  // ... do stuff
  progress.report( i: 100);
  progress.end();
}
```

- Uhum, Automated test?
  - Popping up progress bars?
    - No, we do not want that, so?
  - Absense = Null Reference?
    - progress = null;

AARHUS UNIVERSITET

The Null Pointer

- Null pointer = **absence of an object**
  - [in Machine code, it is address 0L in memory]

History [edit]

In 2009, Sir Tony Hoare stated[14] that he invented the null reference in 1965 as part of the ALGOL W language. In that 2009 reference Hoare describes his invention as a "billion-dollar mistake":

> 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.

AARHUS UNIVERSITET

Motivation

- Ups – Null pointer exceptions ☹
- Solution
  - Avoid it with *guards*

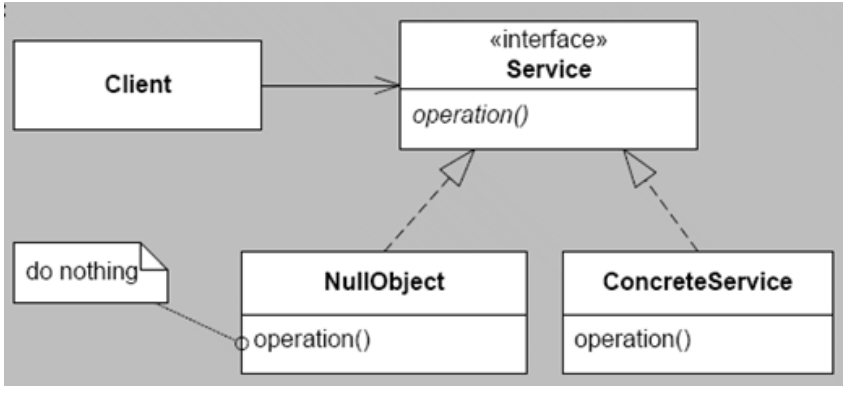
```
public  void lengthyExecution(ProgressBar progress) {
  // ... do stuff
  progress.report( i: 10);
  // ... do stuff
  progress.report( i: 50);
  // ... do stuff
  progress.report( i: 100);
  progress.end();
}
```

```
public void lengthyExecution(ProgressBar progress) {
  // ... do stuff
  if (progress != null) {
    progress.report( i: 10);
  }
  // ... do stuff
  if (progress != null) {
    progress.report( i: 50);
  }
  // ... do stuff
  if (progress != null) {
    progress.report( i: 100);
    progress.end();
  }
}
```

- No, no, no ☹
  - Low analyzability…

A Solution

- The problem
  - Representing ‘absence’ by null
- Can be solved by
  - Representing ‘absence’ with a special “absence object”:
  - The null object…
- ③-①-②:
  - ‘Having or not having object’ is variable
  - Express ‘the object’ via an interface (‘service’ in the UML above)
  - Delegate to it
    - Allows a NullObject implementation to be injected…



Thus

- Instead of calling
  - lenghtyExecution(null);
- … we call with a null object
  - lengthyExecution(new NullProgressBar());

```
public class NullProgressBar implements ProgressBar {
  @Override  5 usages
  public void report(int i) {}
  @Override  1 usage
  public void end() {}
}
```

AARHUS UNIVERSITET

Example: Net4Care

- TDD of AppServer which stores documents in XDS.b (database system)
  - SOAP messages containing ebXML payload sent to web service on remote machine that connects a MicroSoft SQLServer 2008!

- But but, I am only interested in testing internal stuff...

```
/** Null object implementation of XDSRepository that does absolutely
 * nothing. All return values are null.
 */
public class NullXDSRepository implements XDSRepository {

  @Override
  public void provideAndRegisterDocument(RegistryEntry metadata,
                                         Document xmlDocument) {}
  @Override
  public List<Document> retrieveDocumentSet(XDSQuery query){
    return null;
  }

  @Override
  public List<String> retrieveDocumentSetAsXMLString(XDSQuery query) {
    return null;
  }

  public void connect() {}

  public void disconnect() {}

  public void utterlyEmptyAllContentsOfTheDatabase() {
  }

  public String toString() {
    return "NullXDSRepository.";
  }
}
```

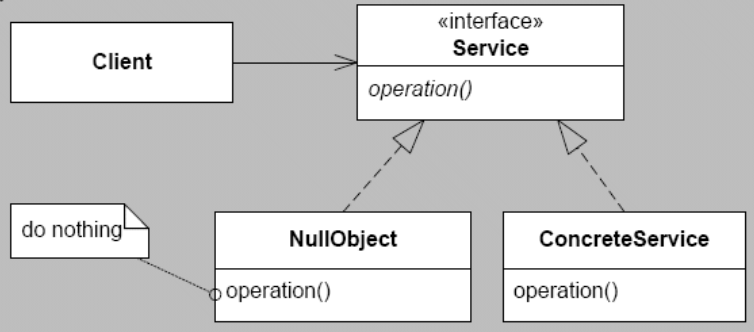
[27.1] Design Pattern: Null Object

Intent Define a no-operation object to represent null.

Problem The absence of an object, or the absence of behavior, is often represented by a reference being null but it leads to numerous checks to ensure that no method is invoked on null. It is easy to forget such checks.

Solution You create a Null Object class whose methods have no behavior, and use an instance of this class instead of using the null type. Thereby there is no need for null checking before invoking methods.

Structure:



Roles **Service** defines the interface of some abstraction while **ConcreteService** is an implementation of it. **NullObject** is an implementation whose methods do nothing.

Cost - Benefit It reduces code size and increases reliability because a lot of *testing for null is avoided.* If an interface is not already used it *requires additional refactoring to use.*

AARHUS UNIVERSITET

Modern Languages…

… tries to tackle it at language level

AARHUS UNIVERSITET

Paths to avoid the issue…

- Lots of work to handle it better
  - **Kotlin** – you can explicitly state that a reference cannot be null

```
val nullable: String?
val nonNullable: String
```

  - You cannot call a method on a nullable

```
val nullable: String? = "FooBar"
nullable.toLowerCase()
```

  - Must use the ‘null safe operator’ ?.
    - nullable**?.**toLowerCase()
    - (Often called the ‘elvis’ operator)
  - Also in C# and Groovy

AARHUS UNIVERSITET

Java: Optional<T>

- Java 8 introduced **Optional** which is a wrapper or a kind of proxy
  - Optional has an inner object which may be null
  - … and methods to handle the inner object
- Our example

```
progress.report( i: 10);
```

```
var optionalProgress = Optional.ofNullable(progress);
optionalProgress.ifPresent(bar -> bar.report( i: 10));
```

- Hmmm – compare analyzability to the elvis operator
  - progress?.report(10);