Null Object

A major ’if’ killer ☺

Motivation

Automated testing of lengthy operation, but…

- To improve usability we show a progress bar ☹

Solution?

```
public void lengthyExecution() {
  ...
  progress.report(10);
  ..
  progress.report(50);
  ...
  progress.report(100);
  progress.end();
}
```

```
public void lengthyExecution() {
  ...
  if (progress != null )
    progress.report(10);
  ..
  if (progress != null )
    progress.report(50);
  ...
  if (progress != null ) {
    progress.report(100);
    progress.end();
  }
}
```

A Solution

The problem

- Representing ‘absence’ by null

Can be solved by

- Representing ‘absence’ with a special absence object

The null object…

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.";
  }

}
```

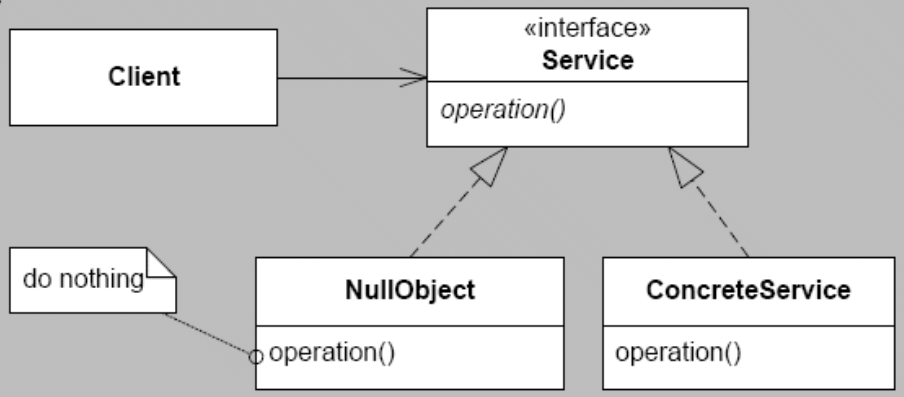
[27.1] Design Pattern: Null Object

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.*