

A decorative graphic on the left side of the slide consisting of two overlapping parallelograms. The front one is blue and the back one is a light green. They are positioned diagonally, with the blue one partially covering the green one.

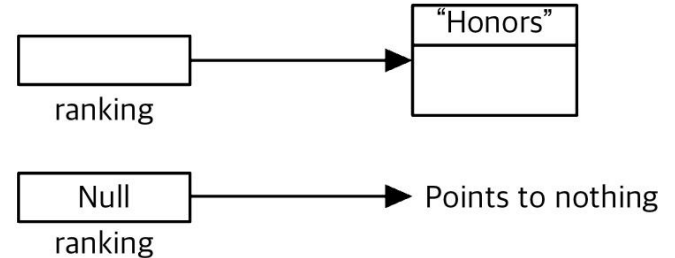
# Null Object Design Pattern

Brandon, Hannah & Angie

# Background: What does “null” mean?

- In simplest terms, null means “no value exists.”
- Commonly used when programmers do not have a memory location they want to assign a variable to.
- Without the use of null, a string may not correctly execute which can cause problems.

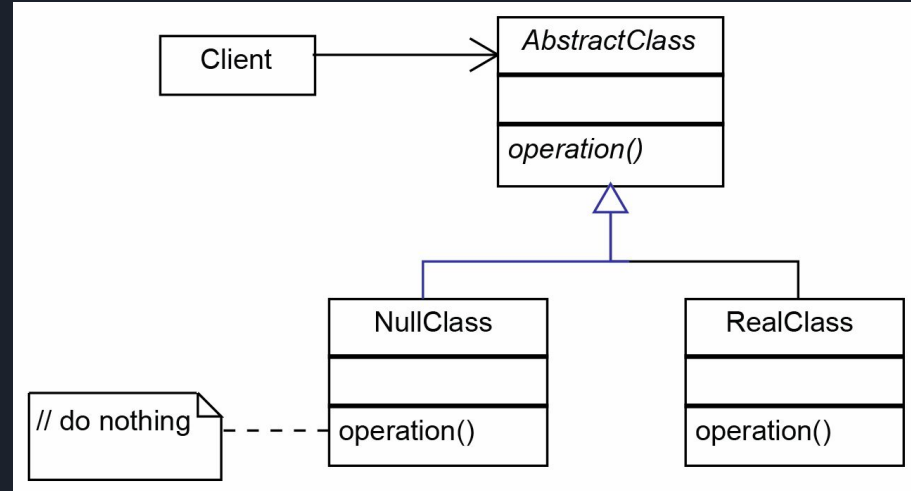
Student ranking = new student (“Honors”);  
Student ranking = null;



“Null” can be defined by the keyword *null* in most programming languages

# The Null Object Design Pattern

- The design pattern removes the need to check if an object's values is not equal to *null*.
- A null object class can be created so the object can be returned alternatively.
- Can be thought of as a default functionality.



# What is the Null Object Design Pattern Used For?

- In Object-Oriented software (Java, C#, and C++)
- Frameworks
- Libraries
- Applications dealing with unknown objects
- Handling optional parameters
- Default values
- Placeholders



Java Application Framework



JavaScript open source server environment.



# The Null Object Design Pattern Example

The following example is a program that tells students if they can enroll in honor courses. The requirement is that the student must be in the honors program and they must have a valid id. They are either in the honors program or not. New students may not have an id yet (a null student). We determine if a student can take an honors course by checking to see if their *rank* field is not equal to *null* and if their id is not equal to *null*.

```
ArrayList<Student> students = new ArrayList<>(); // Create a new array of students
```

```
// Add students to the list
```

```
students.add(new Student(id:"12345", rank:"Honors"));
```

```
students.add(new Student(id:null, rank:null));
```

```
students.add(new Student(id:"12346", rank:"Honors"));
```

```
students.add(new Student(id:"12347", rank:null));
```

```
students.add(new Student(id:null, rank:null));
```

```
for (int i = 0; i < students.size(); i++) {
```

```
    String currStudentId = students.get(i).getId(); // The current student's id
```

```
    String currStudentRank = students.get(i).getrank(); // The current student's rank
```

```
    // Check to see if the student is an honors student
```

```
    if (currStudentId != null && currStudentRank != null) {
```

```
        System.out.println("Hello, student " + currStudentId + ". You can take honor courses." + "\n");
```

```
    }
```

```
    else if (currStudentId != null && currStudentRank == null) {
```

```
        System.out.println("Hello, student " + currStudentId + ". You cannot take honor courses." + "\n");
```

```
    } else {
```

```
        students.get(i).setId(newId:"new student");
```

```
        System.out.println("Hello, " + students.get(i).getId()
```


```
            + ". You do not have an id and you cannot take honor courses." + "\n");
```

```
    }
```



# The Null Object Design Pattern Example

It would not necessarily be a bad design to use null checks in the previous program since the example only consists of a few students. However, if we had an entire database of students at a university or high school for example, we would have to check a student's rank and id every time before we could do anything with it.



## Implementation of the Null Object Design Pattern

Instead of using null checks, we can create a new class which will be the default version of a student (if their id is null).



```
for (int i = 0; i < students.size(); i++) {  
  
    String currStudentId = students.get(i).getId(); // The current student's id  
    String currStudentRank = students.get(i).getrank(); // The current student's rank  
  
    if (currStudentId == null) {  
        System.out.println(new NullStudent()); // Return a default student with no id or rank  
    }  
  
    else if (currStudentRank == null) {  
        System.out.println("Hello, " + currStudentId + ". You cannot take honor courses.");  
    }  
    else {  
        System.out.println("Hello, " + currStudentId + ". You can take honor courses.");  
    }  
}
```

# NullStudent Class

```
1 public class NullStudent {  
2  
3     String id;  
4     String rank;  
5  
6     public NullStudent() {  
7         this.id = "new student";  
8         this.rank = "no rank";  
9     }  
10  
11     @Override  
12     public String toString() {  
13         return "Hello, " + id + ". You have no rank.";  
14     }  
15  
16 }  
17
```



## The Problem Solving Aspect

The null object allows the object's reference to be optionally null. Making the null check to do nothing or use some default value. This allows us to treat a collaborator that does nothing the same way as a collaborator that provides behavior. For example the previously added new students, who we are still treat as students, even though they don't have a rank or ID to reference. With null we can still determine if they can still take an honors course. That way we can make sure we can go through all students fairly without missing one.



## The Pros

- Makes the client code simple. Originally the client does not know if it is dealing with a real or a null object. But null objects can be used in place of real objects when the object is expected to do nothing.
- Reduces the need for explicit null checks in client
- code. This makes the code look much cleaner.
- Since null objects are not actually null, they won't throw `NullPointerException`s, so methods can call on them without this being a problem.
- Null objects act like real objects to the same interface. This allows them to be used interchangeably. This makes it easier to swap out objects without affecting the client code.



## The Cons

- It's not easy to implement if various clients do not agree how the null object should “do nothing.” This can happen if your Abstract Object is not defined well.
- A new Null Object class may be needed for every new Abstract Object class, which can be a lot of classes.
- The Null Object Pattern is really only effective in scenarios where there's a clear default behavior for missing objects. Another way to deal with null may be more appropriate in cases where null represents an error condition.



# The limitations of the Null Design Pattern

- Null does not always accurately represent an absconded of a value. It can make it challenging to distinguish between an intentional null object and one that occurs due to an error.
- May not always be applicable, especially when the absconded of a value could be represented in a less complex way to prevent confusions in the code.
- It may also cause confusions in large projects that have multiple different software developers working, creating inconsistent usage of null objects and references.



## Real World Uses

A function may retrieve a list of files in a folder and perform some action on each. In case a folder is empty, we do not want the response to be to throw an exception or return a null reference rather than a list because this will cause to be an over complicated design and take way to long for a code that just wants to verify whether there is a list in that folder.



## Real World Uses

Instead, by returning a null object, we verify there is a list in the folder and since conditions of finding a list has been met (even though it's empty) we move on. Quick and simple. It verified it exists and does nothing to it and that's why null object is useful.





## Real World Uses

Null object is also really useful in databases. If you've ever taken databases and used SQL, you'll realise how much null is used so we can filter objects that meet a specific condition, but still have empty values that we just have the system purposely ignore. And we can filter out the ones with empty values if need be by stating the correct conditions.



# Sources

[Null object Design Pattern - GeeksforGeeks](#)

[Introduction to the Null Object Pattern | Baeldung](#)

[Working with Design Patterns: Null Object | Developer.com](#)

[Design Patterns: Null Object. Avoid conditional complexity using this... | by Carlos Caballero | Better Programming](#)

[https://en.wikipedia.org/wiki/Null\\_object\\_pattern](https://en.wikipedia.org/wiki/Null_object_pattern)

[https://sourcemaking.com/design\\_patterns/null\\_object](https://sourcemaking.com/design_patterns/null_object)