**Introduce Null Object**

**Motivation**

Some methods in a code can return null when they are executed. Having conditionals that check for null can lead to code being repeated many times throughout the code base. This makes the code longer and harder to read, since there is a lot of repeated code to read through, and it makes it hard to figure out what code does what function. Instead of checking for null with a conditional and executing the appropriate functionality, a class can be created that returns a null object that executes default behaviour. This ensures that the code can just call the methods in this null object instead of checking for null each time. `

**Methods**

1. Using the class to be refactored, create a subclass that will become a null object.
2. Create a isNull() method in both classes to return true for a null object
3. Find all places in the class where the code returns null and instead make it call a null object
4. Find all places in the rest of the code where null is checked and change it to call for the isNull() method.
5. Transfer the code that deals with non-null behaviour to methods in the null subclass.

**Sample code to refactor**

A business wants to keep track of the types of billing plans that is assigned to each of their customers. If a new customer joins, the program creates a new basic billing plan for that customer. For existing customers, the program returns the customer’s existing billing plan. The code that handles this is below.

**if** (customer == **null**) {

plan = BillingPlan.basic();

}

**else** {

plan = customer.getPlan();

}

**Risks**

* Requires adding new classes, which can complicate the code if many classes already exist.
* Increases maintenance due to the implementation of the null object using inheritance
* May cause duplication of the entire team is not made aware of the null object