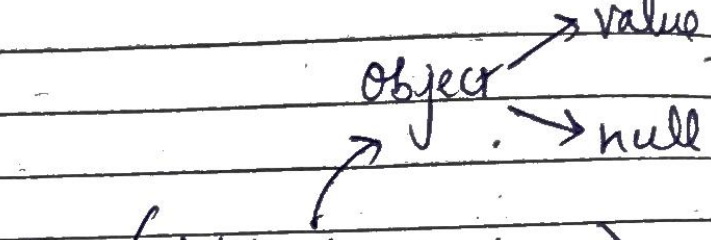
LLD of Handling NULL (NULL object Design Pattern)

Problem.




```
Void PrintVehicleDetails (Vehicle vehicle)
{
    // code
}
```

If the object is null than we can get error while executing this code.

Solution.

```
void PrintVehicleDetails (Vehicle vehicle)
{
    if (vehicle != null)
    {
        // execute code
    }
}
```

But this solution is not viable if we create thousands of objects as we have to write this condition for every object which is not a good practice.

Page No.
Date / /

Better Solution:

This solution avoids unnecessary checks that we putting in the code.

Problem in Simple language

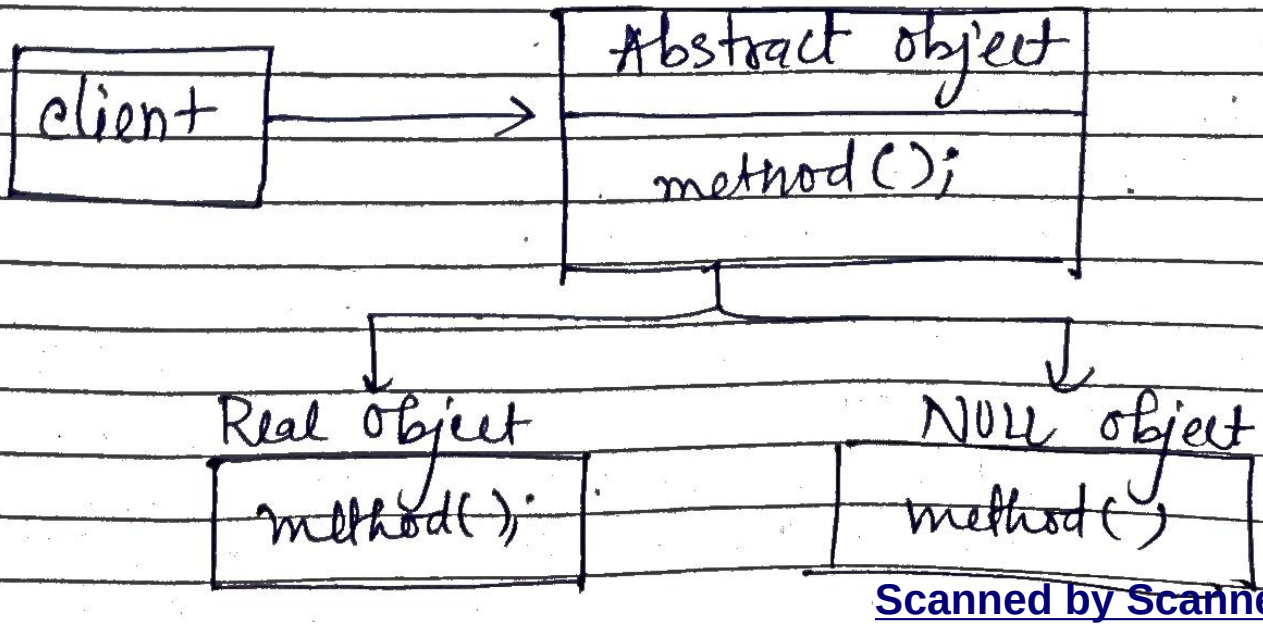
↳ How we can avoid != null condition in the code when an object is accessed.

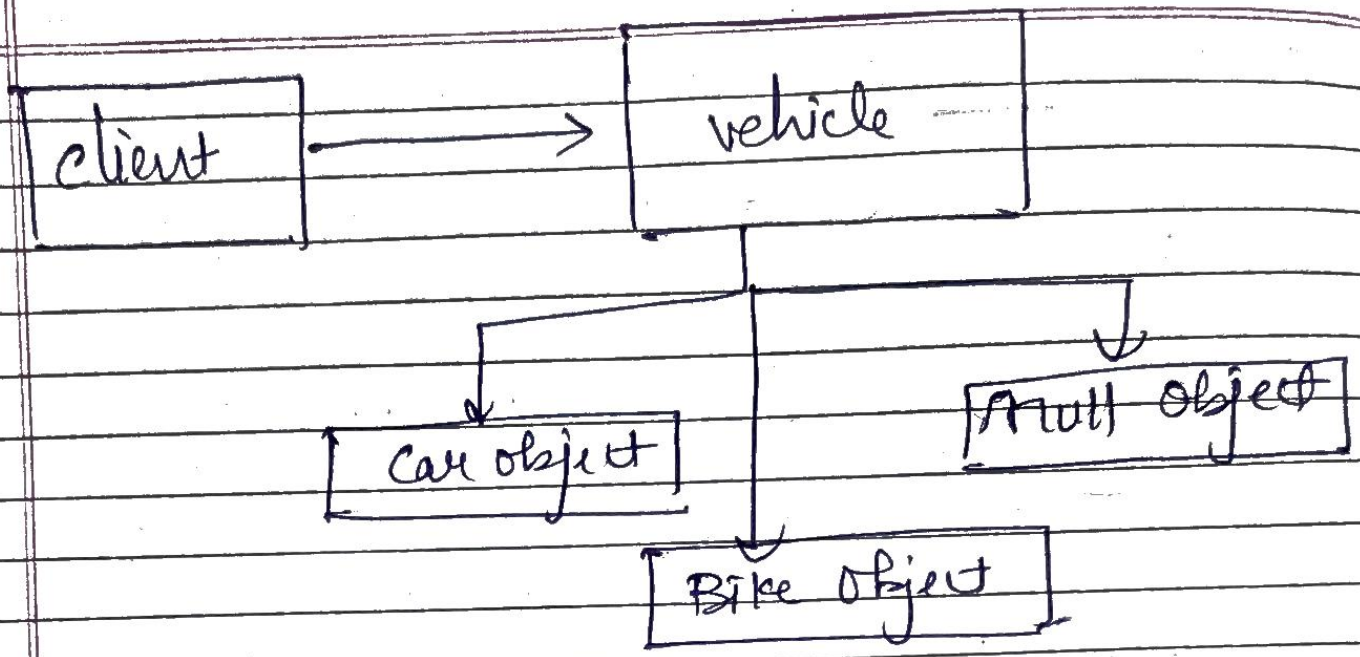
- Solution : Null object Design Pattern

This states that

- A null object replace NULL return type.
- No need to put If check for checking null
- Null object reflects do nothing or Default Behavior.

UML





Now vehicle class can return car object, Bike obj or Null obj but not Null value.

```
Public Interface Vehicle
{
    int getTankCapacity();
    int getSeatingCapacity();
}

Public class Car implements Vehicle
{
    @override
    public int getTankCapacity()
    {   return 40;
    }
    @override
    Public int getSeatingCapacity()
    {   return 5;
    }
}
```

Page No.
Date / /

```
Public class NullVehicle implement Vehicle
{
    @override
int getTankCapacity()
    {
        return 0;
    }
    @override
int getSeatingCapacity()
    {
        return 0;
    }
}
```

Now we create a factory of vehicle

```
Public Class VehicleFactory
{

Static Vehicle getVehicleObject(String type)
{
    if("car".equals(type)
    {
        return new Car();
    }
    return new NullVehicle();
}
```

** Instead of returning null we are returning null Object

Page No.
Date / /

This pattern states that Replace null value with null object that contains the Default behavior of object