* In Null Object pattern, a null object replaces check of NULL object instance.
* This way we no longer need to deal with special handling of null references.
* It can also be used to provide default / donothing behaviour in case data is not available.
* A Null Object can be a special case of the State pattern.

In Null Object pattern, we create an abstract class specifying various operations to be done, concrete classes extending this class and a null object class providing do nothing implemention of this class and will be used seemlessly where we need to check null value.

public abstract class Vehicle {  
 abstract int getWheels() ;  
}

public class Car extends Vehicle{  
  
 @Override  
 int getWheels() {  
 return 4 ;  
 }  
}

public class Bike extends Vehicle{  
 @Override  
 int getWheels() {  
 return 2;  
 }  
}

public class NullVehicle extends Vehicle{  
 @Override  
 int getWheels() {  
 return 0; *// defaulted or do nothing behaviour* }  
}

public class VehicleFactory {  
  
 public static Vehicle getVehicle(String type ){  
 if(type.equals("car")) {  
 return new Car();  
 } else if (type.equals("bike")) {  
 return new Bike();  
 }  
 return new NullVehicle();  
 }  
}

public class NullObjectPattern {  
 public static void main(String[] args) {  
  
 Vehicle vehicle = VehicleFactory.*getVehicle*("something else") ;  
 *// This will return the NullVehicle Instance* System.*out*.println(vehicle.getWheels());  
 }  
}

Output :

Process finished with exit code 0