Null object: A behavioral design pattern with no behaviors

1. When component A uses component B, it typically assumes that B is non-null
   1. You inject B, not some Option<B> type
   2. You do not check for null on every call
2. There is no option of telling A not to use an instance of B
   1. Its use is hard coded
3. Thus, we build a no-op, non-functioning inheritor of B (or some interface that B implements) and pass it into A.
4. Null object: A no-op object that conforms to the required interface, satisfying a dependency requirement of some other object.

Null Object

1. Suppose you have a constructıor that takes an object but you are not dependent on that object so you want to pass null. This may cause null exceptions since the class probably uses that object so you create a null object that implements the same interface and does nothing.

Dynamic Null Object

1. If you do not want to create a dedicated Null class using proxy behavior.
2. Proxy is very costly. Be careful with it.

@SuppressWarnings("unchecked")

public static <T> T noOp(Class<T> itf)

{

return (T) Proxy.*newProxyInstance*(

itf.getClassLoader(),

new Class<?>[]{itf},

(proxy, method, args) ->

{

if (method.getReturnType().equals(Void.*TYPE*))

return null;

else

return method.getReturnType().getConstructor().newInstance();

});

}



Summary

1. Implement the required interface
2. Rewrite the methods with empty bodies
   1. If method is non-void, return default value for a given type
   2. If these values are ever used, you are in trouble
3. Supply an instance of Null Object in place of actual object
4. Cross your fingers
   1. Not the safest design pattern but sometimes needed. Eg: unit tests