Eliminating Null Reference Exceptions

Jason Roberts
.NET MVP

@robertsjason dontcodetired.com

Overview

Introduce the Null Object pattern

Why use the Null Object pattern?

Overview of the pattern

Example code without the Null Object pattern in use

Implementing the Null Object pattern using interfaces

Refactoring to a base class and single null object instance

Considerations

Null Object Pattern

A software design pattern that uses object-orientation to remove or reduce the amount of null references and thus reduce the burden of repetitive null checking and handling code in specific parts of the codebase.

Why Null Objects?

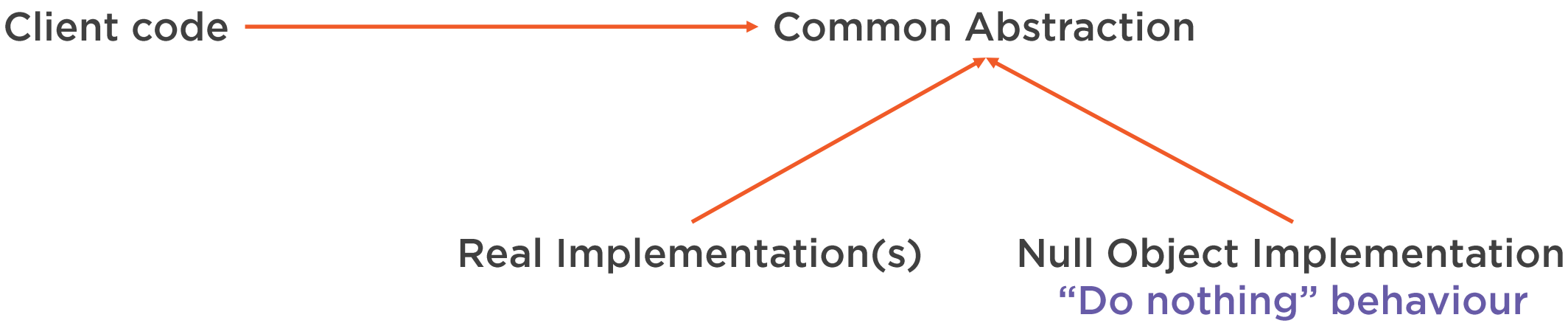
Happier users

- Reduce incidences of runtime null reference exceptions

Happier developers

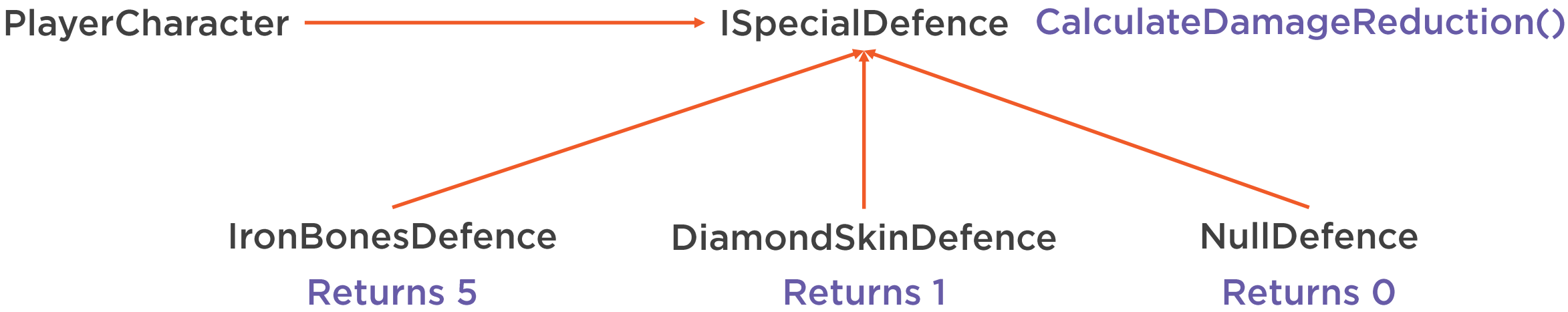
- Reduce the amount of repetitive null checking code we have to write
- Simplify production and test code
- Fewer branches in program control flow
- Reduce maintenance overheads: “do nothing” code exists in one centralized place

Pattern Overview



Client code can treat null objects and “real” objects the same way

Pattern Overview



Considerations

All developers need to be aware that the null object pattern is in use

All calling client code needs to agree on what “do nothing” behaviour is

Usually not used for error handling

Can hide exceptions and make fault finding harder

E.g. if a method should always return an instance of an object, throw exception and “fail fast”

Summary

- Introduced the Null Object pattern
- Reduce runtime null reference exceptions
- Reduce repetitive null checking code
- Overview of the pattern
- Example code without the Null Object pattern in use
- NullDefence : ISpecialDefence
- abstract class SpecialDefence
- SpecialDefence.Null
- Considerations

Next:

Understanding Non-Nullable Reference Types in C# 8