# Coping With Null References

Zoran Horvat
CTO at InterVenture GmbH
@zoranh75 www.codinghelmet.com

## The Problem of Null References

**Dealing with nulls complicates code**

- Remove nulls and code will be simpler

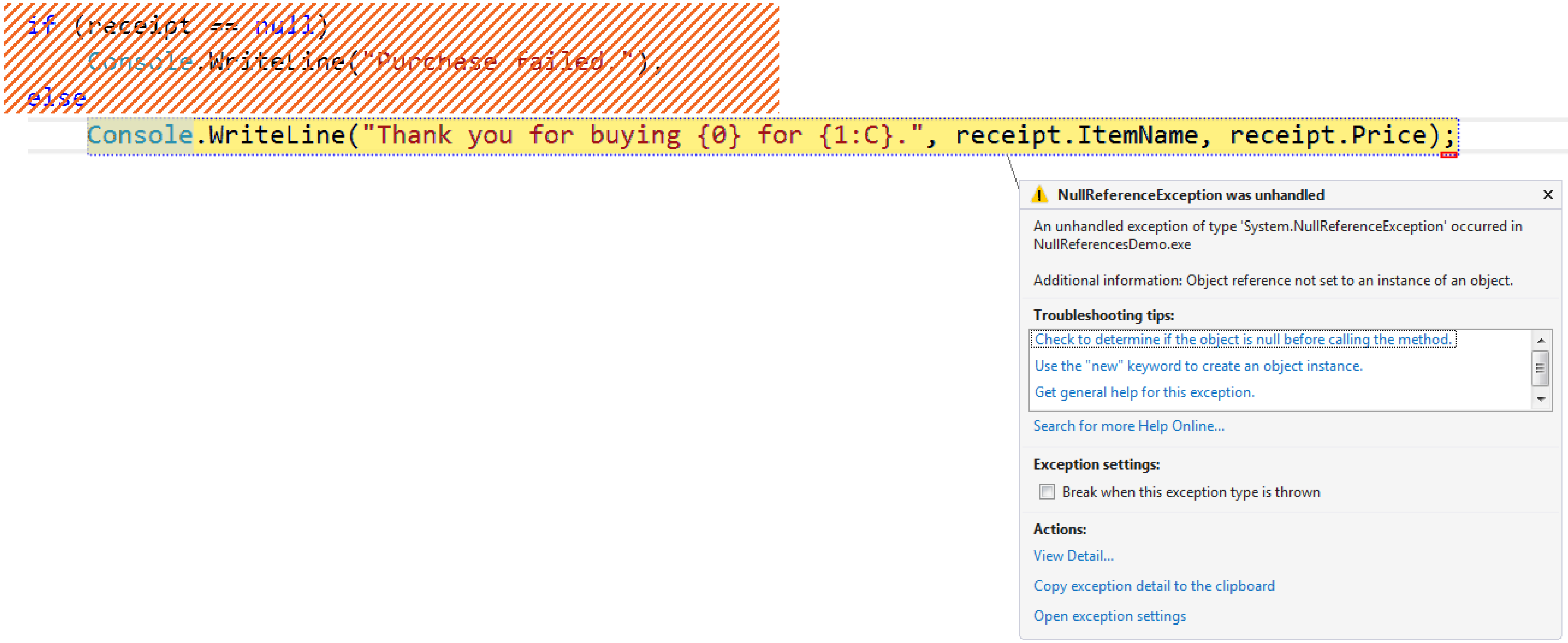
**We can remove null references**

- Different design patterns are meant to remove use of null references

## Guarding Against Null

```
if (receipt == null)
    Console.WriteLine("Purchase failed.");
else
    Console.WriteLine("Thank you for buying {0} for {1:C}.", receipt.ItemName, receipt.Price);
```

# Guarding Against Null



```
if (receipt == null)
    Console.WriteLine("Purchase failed.");
else
    Console.WriteLine("Thank you for buying {0} for {1:C}.", receipt.ItemName, receipt.Price);
```

**NullReferenceException was unhandled**

An unhandled exception of type 'System.NullReferenceException' occurred in NullReferencesDemo.exe

Additional information: Object reference not set to an instance of an object.

**Troubleshooting tips:**

Check to determine if the object is null before calling the method.
Use the "new" keyword to create an object instance.
Get general help for this exception.

Search for more Help Online...

**Exception settings:**

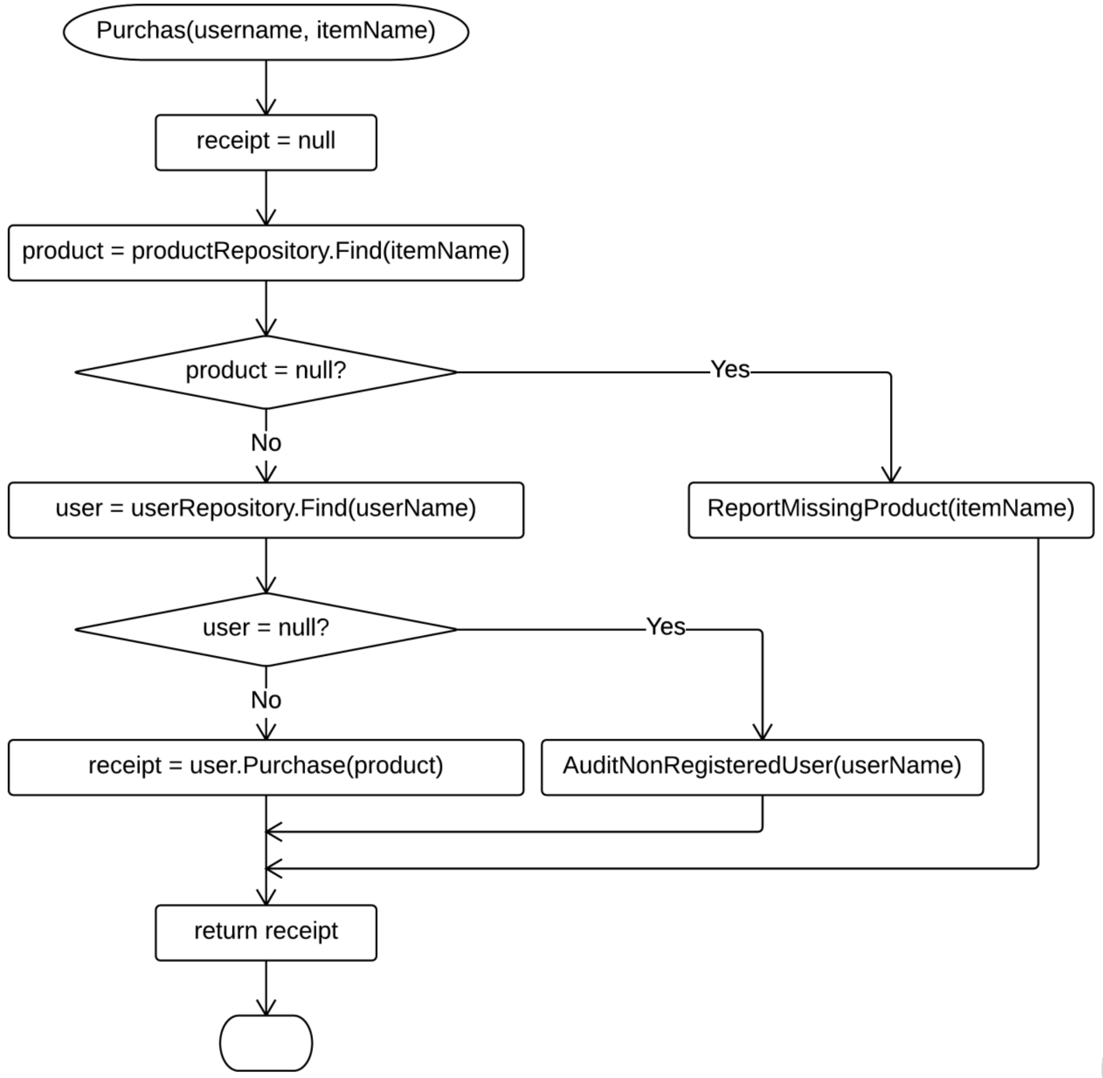
☐ Break when this exception type is thrown

**Actions:**

View Detail...
Copy exception detail to the clipboard
Open exception settings

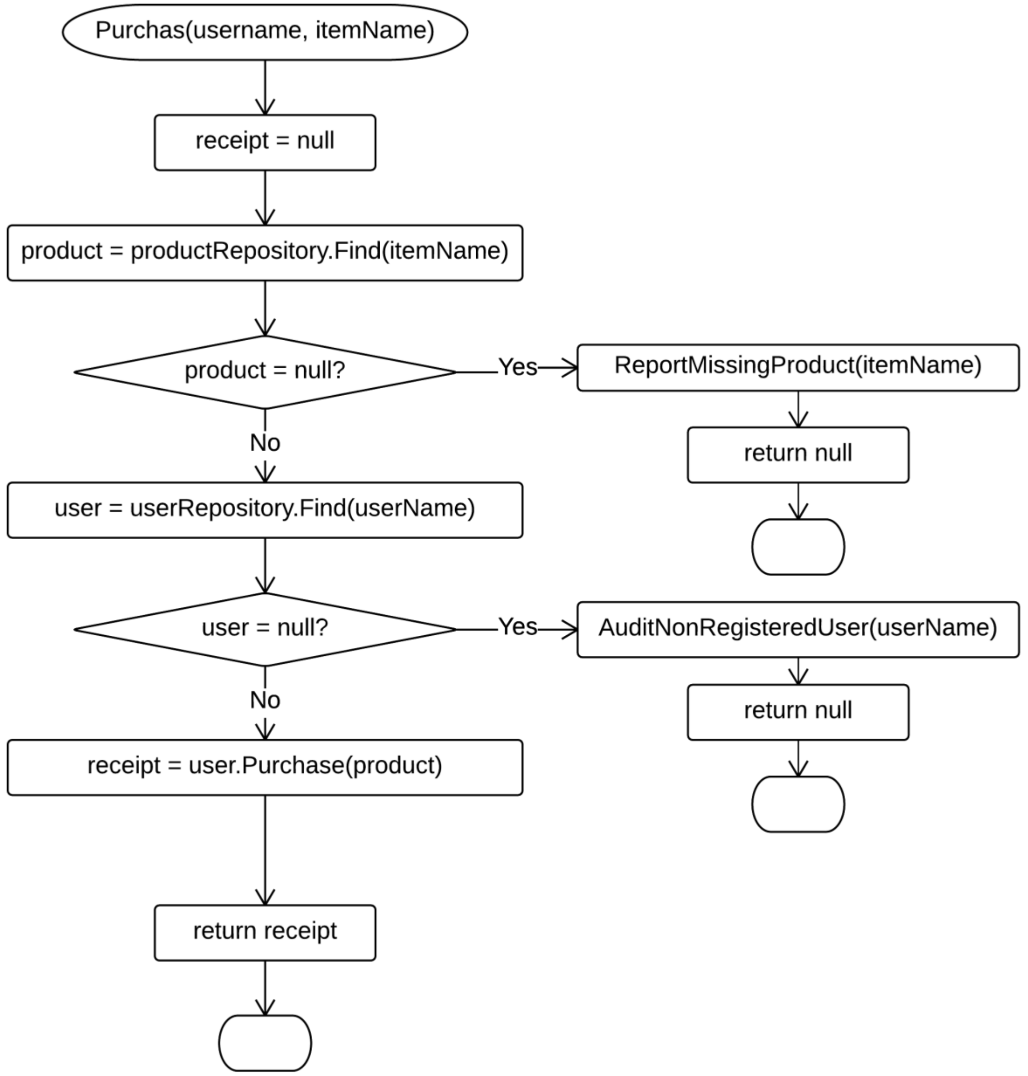
# Control Flow With Null

```
public Receipt Purchase(string username, string itemName)
{

    IProduct product = this.productRepository.Find(itemName);

    Receipt receipt = null;

    if (product != null)
    {

        IUser user = this.userRepository.Find(username);

        if (user != null)
        {
            receipt = user.Purchase(product);
        }
        else
        {
            this.AuditNonRegisteredUser(username);
        }

    }
    else
    {
        this.ReportMissingItem(itemName);
    }

    return receipt;

}
```


Purchas(username, itemName)
receipt = null
product = productRepository.Find(itemName)
product = null?
Yes
No
user = userRepository.Find(userName)
ReportMissingProduct(itemName)
user = null?
Yes
No
receipt = user.Purchase(product)
AuditNonRegisteredUser(userName)
return receipt

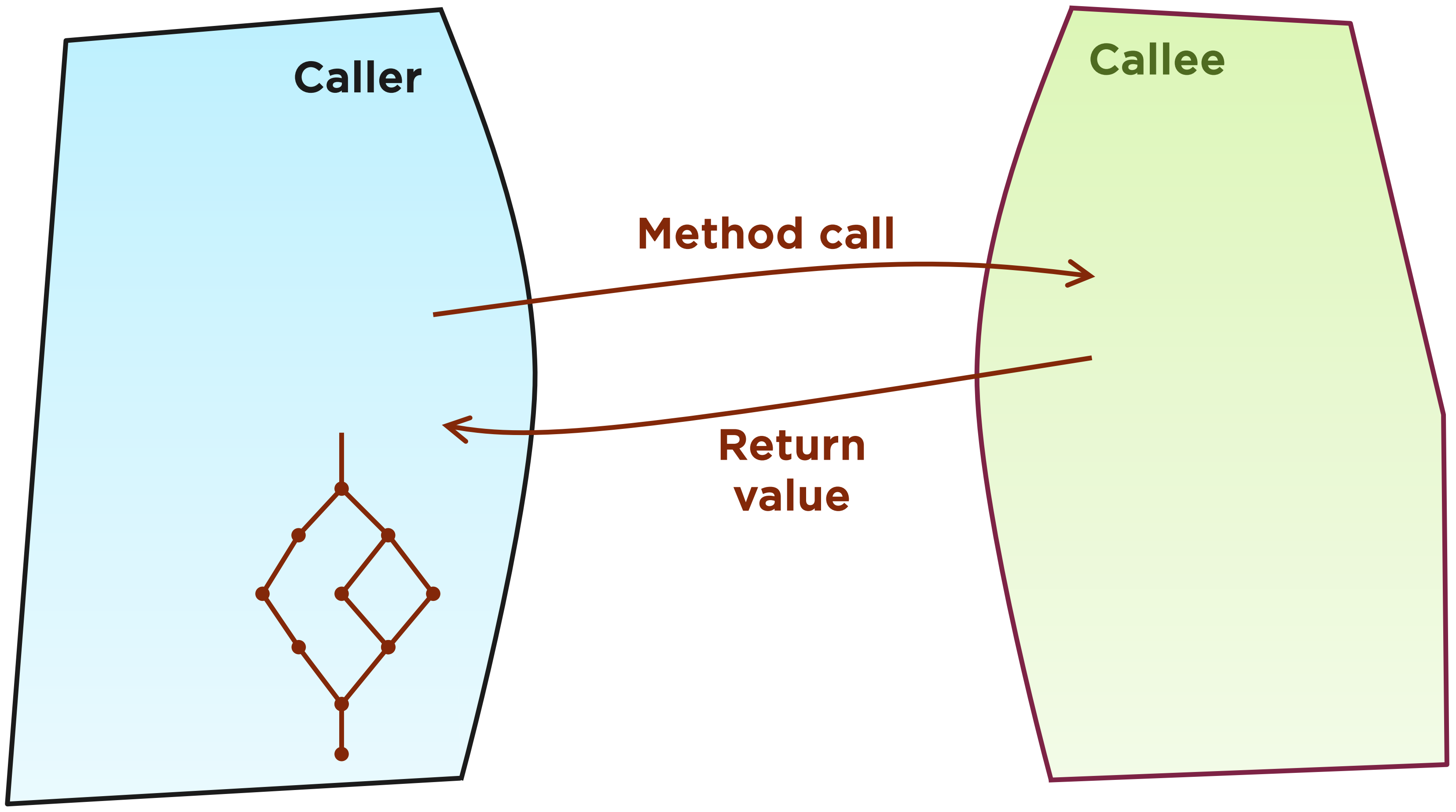
# Control Flow With Null

```
public Receipt Purchase(string username, string itemName)
{

    IProduct product = this.productRepository.Find(itemName);

    if (product == null)
    {
        this.ReportMissingItem(itemName);
        return null;
    }

    IUser user = this.userRepository.Find(username);

    if (user == null)
    {
        this.AuditNonRegisteredUser(username);
        return null;
    }

    return user.Purchase(product);

}
```

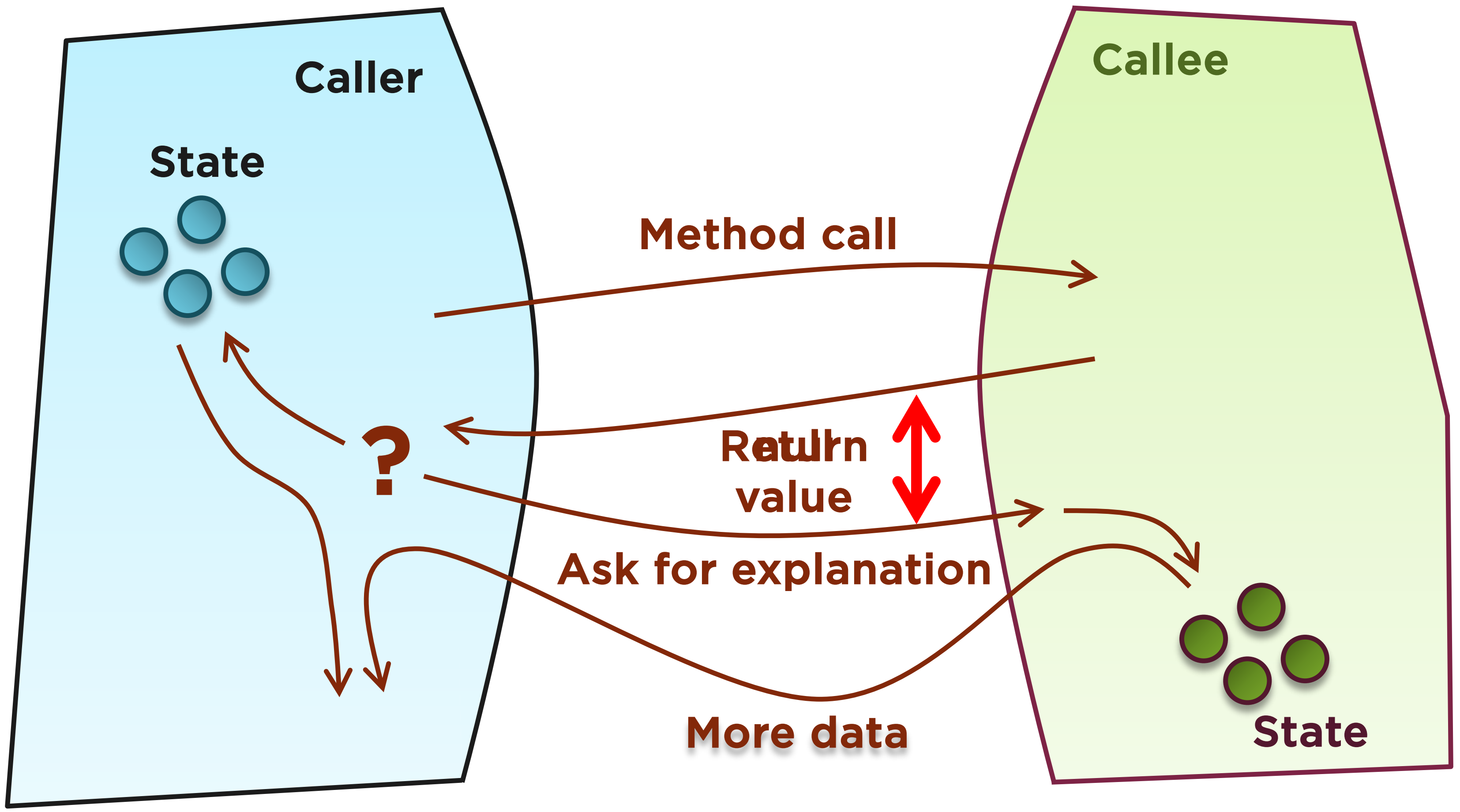


Purchas(username, itemName)
receipt = null
product = productRepository.Find(itemName)
product = null?
Yes
ReportMissingProduct(itemName)
return null
No
user = userRepository.Find(userName)
user = null?
Yes
AuditNonRegisteredUser(userName)
return null
No
receipt = user.Purchase(product)
return receipt

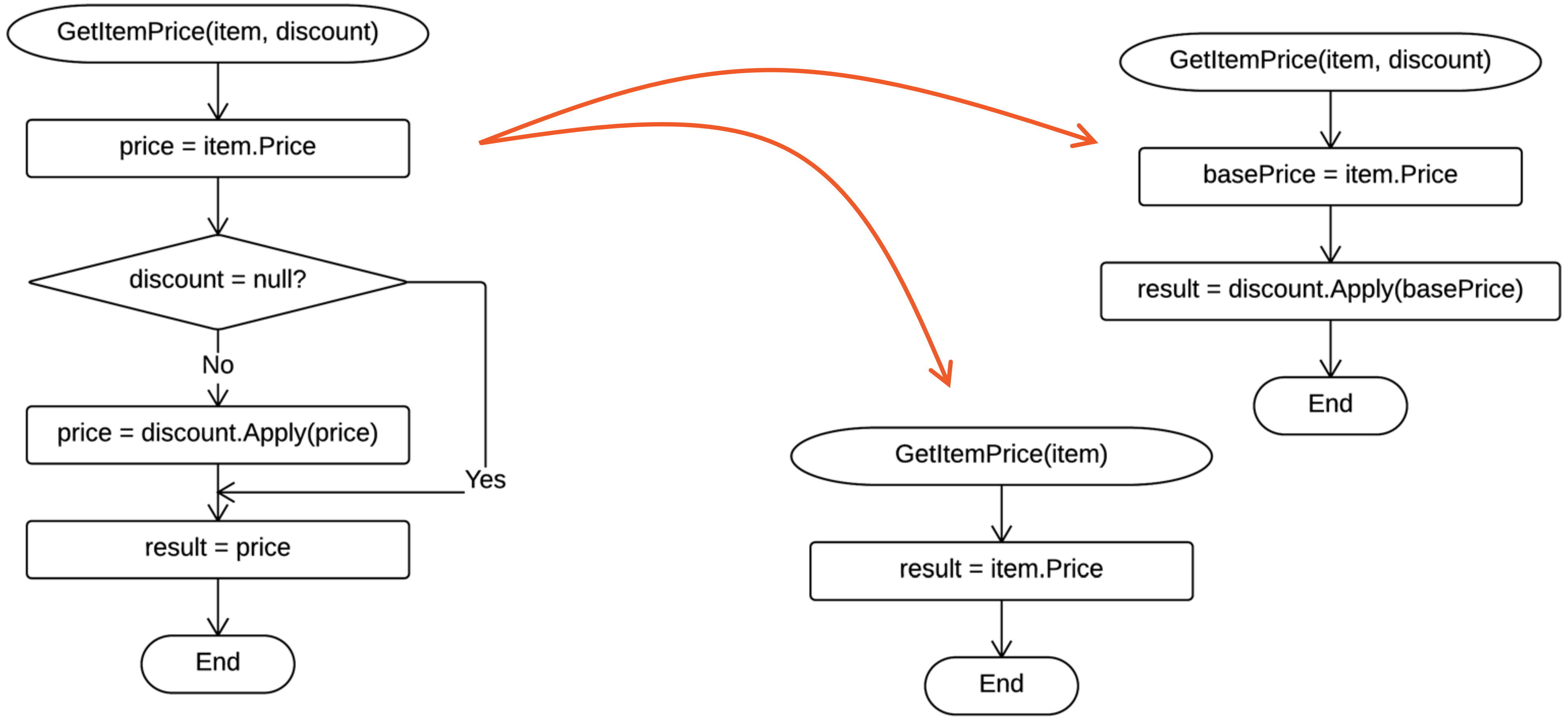
# Problems Caused by Nulls



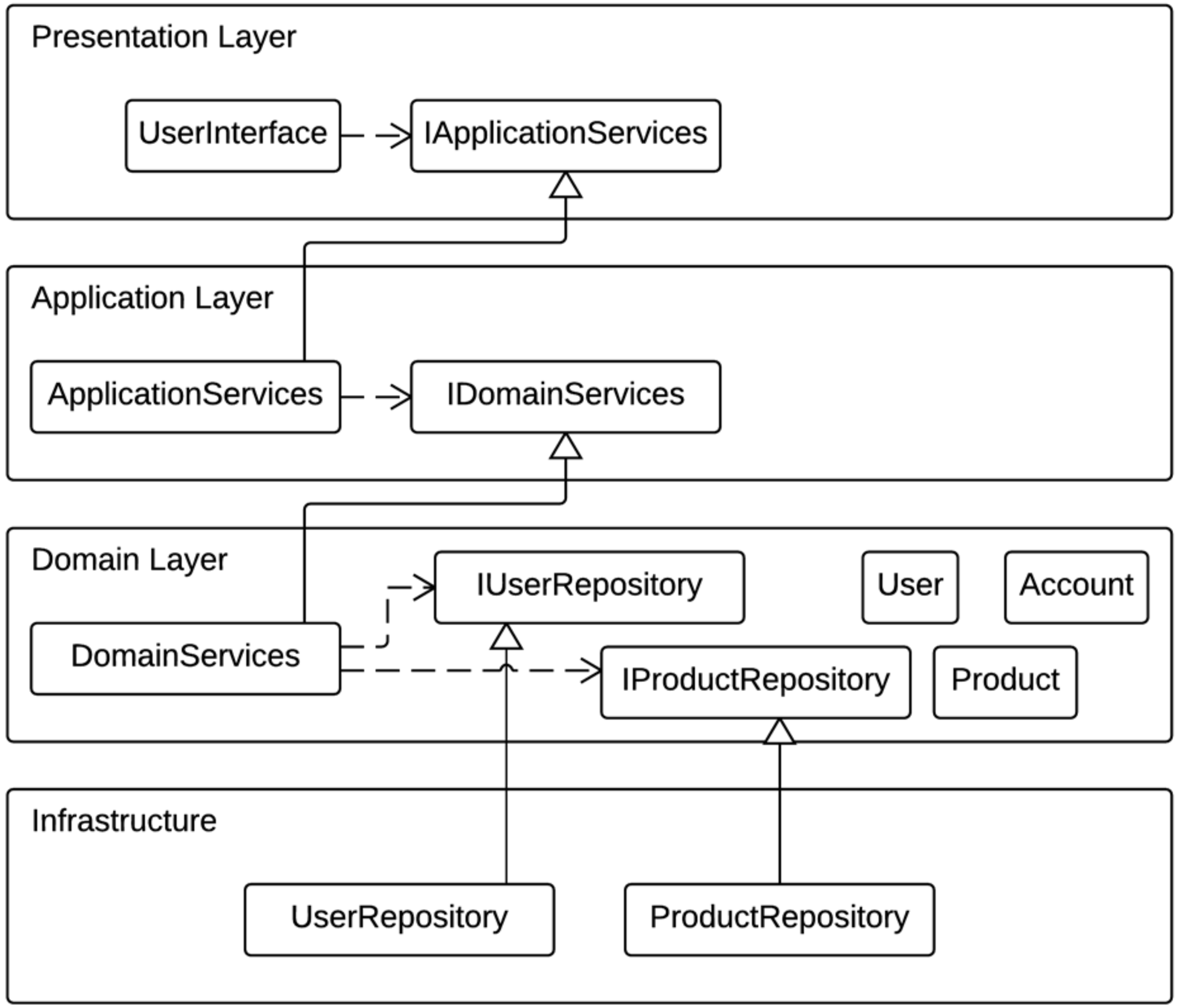
# Problems Caused by Nulls



# The Problem of Null



# Demo Application



## Identifying the Design Problem

**There were many if-then-else statements in code**

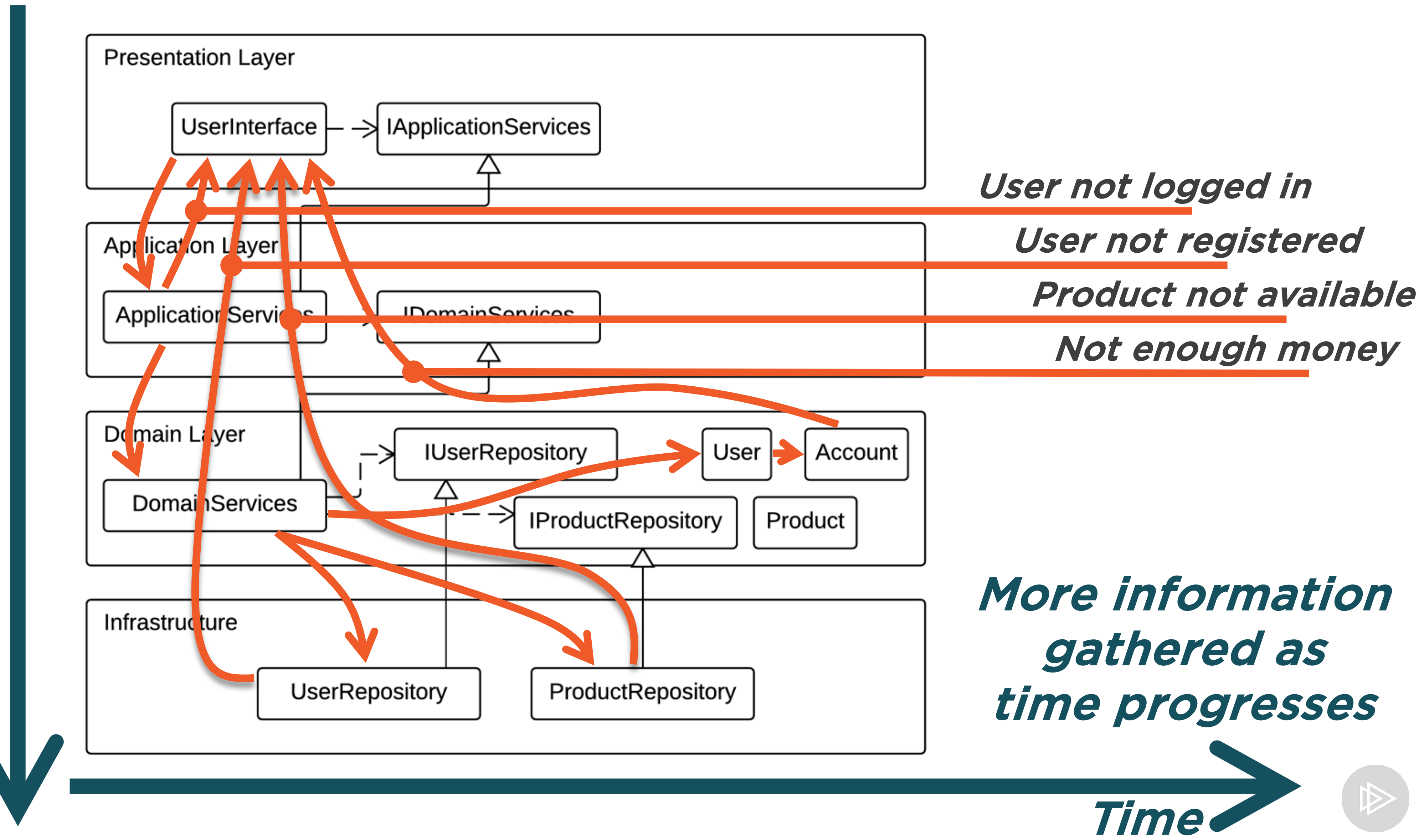
- These were guarding from null references returned from methods

**Information is lost in negative cases**

- Null reference carries no information about what went wrong

# Paths of the Failed Result

Any layer can detect error and stop execution



More information gathered as time progresses

## Refactoring to Design Patterns

**Fresh design is typically driven by requirements**

**When implementation emerges, we start noticing design issues**

- Too much branching on nulls
- Missing information when null was returned

**Then we start refactoring to mitigate the issues**

- Null Object pattern will eliminate branching on null references
- Special Case pattern will let us carry additional information in specific cases
- Option<T> functional type and map-reduce pattern

# Patterns That Help With Nulls

**Module 3 – Remove some null references**

- Introduce Null Object and Special Case design patterns

**Module 6 – Remove remaining null references**

- Replace nulls with Option<T> functional type

**Module 8 – Refactoring guard clause**

- Move branching logic out of domain classes

## Summary

**Two major ways of introducing null references**

- When passing an object into a method
- When returning an object from a method

**Consequences**

- Small gain from not investing into design
- Control flow becomes complicated

**Alternative is to provide proper objects instead of null references**

In the following module:
**Null Object and Special Case Design Patterns**