Testing

Simon Robinson
Software Developer

@TechieSimon www.SimonRobinson.com

Overview

Problems involving testing:

- Avoiding external data
    - Solve with mocking
- Static methods
- Choosing data for tests

Version Check

This module additionally uses NUnit

- It is 100% applicable to NUnit 3.13 onwards

Testing Frameworks

nunit

xUnit.net

MS Test

All frameworks: Same principles

Test code will differ if not using NUnit

Principles apply to all tests (But this module will demo unit tests)

Avoiding External Data in Tests

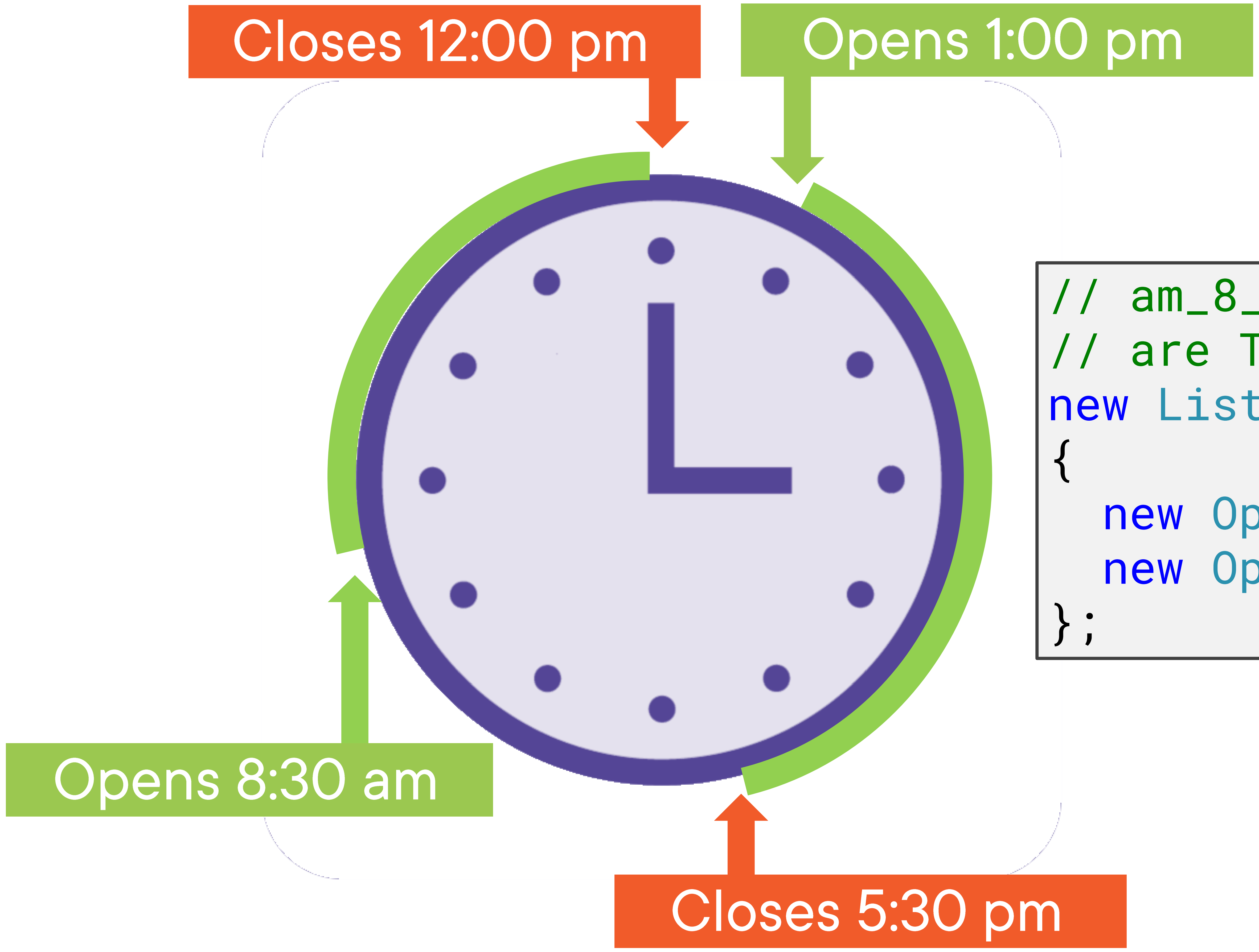
Demo

Code that stores opening times

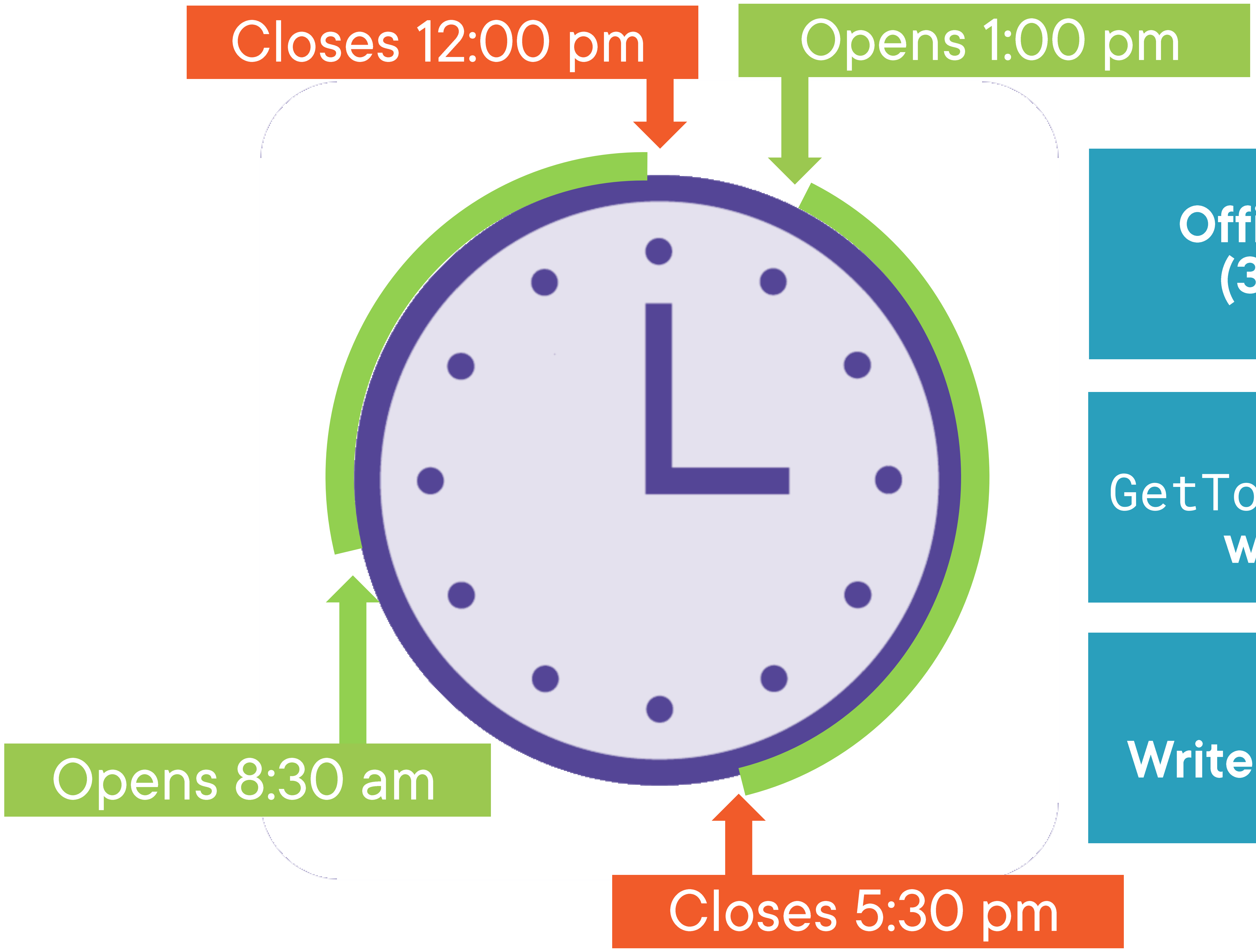
- (For example for some offices)

To test: Calculate how many hours the office is open for

- External data will prevent writing tests
- Use mocking to solve



```
// am_8_30 etc.
// are TimeOnly instances
new List<OpenPeriod>()
{
   new OpenPeriod(am_8_30, pm_12),
   new OpenPeriod(pm_1, pm_5_30)
};
```

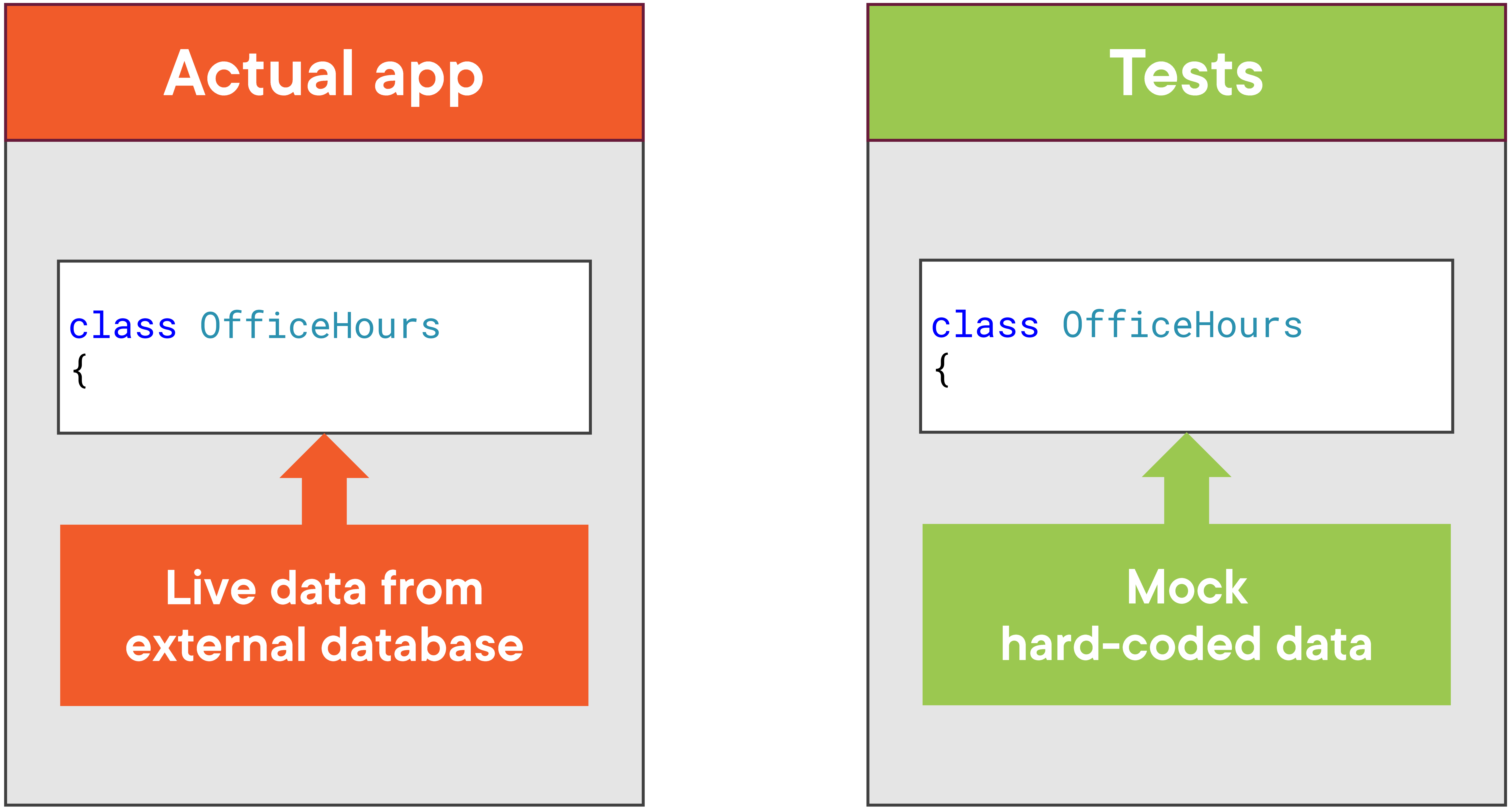


Office is open for 8 hours (3.5 hours am, 4.5 pm)

So you'd expect `GetTotalOpenHoursToday()` **would return 8 hours**

Our task: Write a unit test to verify that

The Principle of Mocking



```
class OfficeHours
{
```

```
class OfficeHours
{
```

Demo

Apply mocking to the office hours test

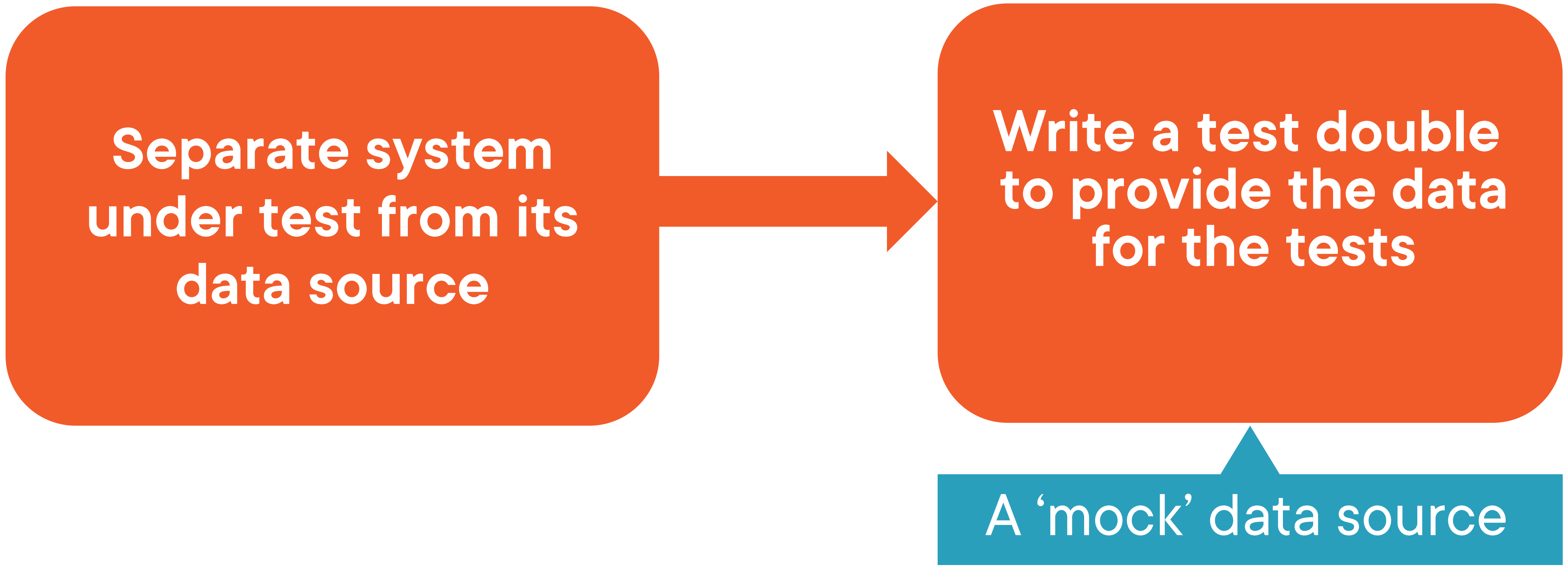
- Rewrite `OfficeHours` to facilitate mocking
- Create a test double class

Rewriting for Dependency Injection

```
// in a real app there would be code like this...
var officeHours = new OfficeHours();
```

```
//... which you must change to something like this
var dataSource = new HoursRepository();
var officeHours = new OfficeHours(dataSource);
```

Removing External Data: Wrap-up


Separate system under test from its data source
Write a test double to provide the data for the tests
A ‘mock’ data source

Sidebar 1: Mocking Libraries

There are open source libraries to help mocking

For example:

moq NSubstitute FakeItEasy

Using a library doesn't change the principles

- It just helps you create test double types

Sidebar 2: Terminology

The word ‘mock’ has two common usages

- Informally, describes the process you’ve seen: Using test doubles:
- More formally, only certain specific types of test double

Testing Code That Relies on Static Methods

Testing and Static Methods

Common belief:

Static methods
are bad for testing

More correctly:

Static methods
can make it harder
to remove
external dependencies

Demo

Two static methods

- One has an external dependency, one doesn’t

We’ll write tests for that code

- Solution will again require mocking

Demo

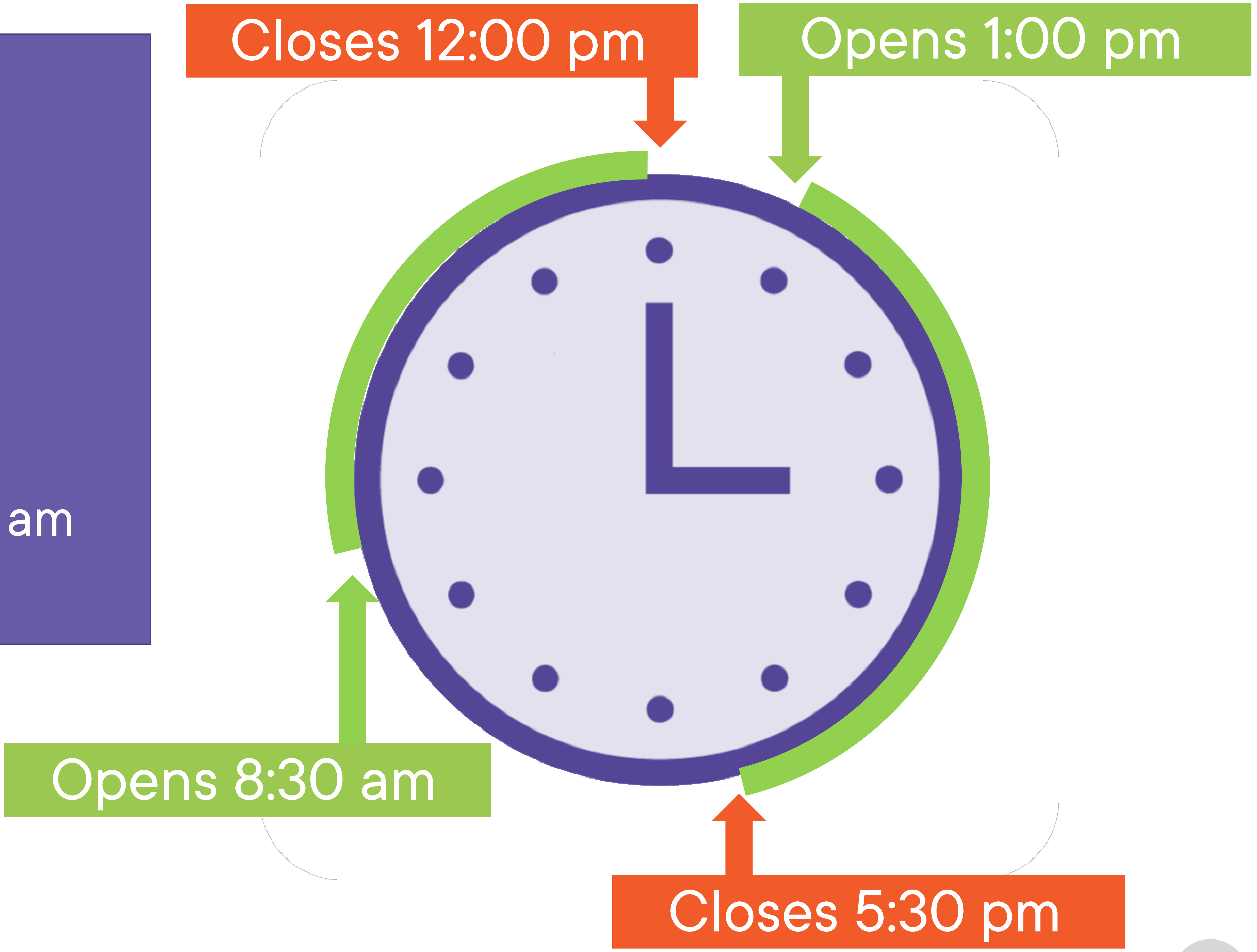
New requirement

- Display length of time until the office next opens
- We must test that calculation

If time now is 7:00 am:

Then it’s 1 hr 30 mins
until the office opens

Our Task: Write a test to verify
`GetTimeUntilNextOpen()`
returns 1 hr 30 mins if it’s 7:00 am



To Test a Method with a Static Dependency

1. Encapsulate dependency in a helper class/interface with no other features

```
public class TimeNowProvider : ITimeNowProvider
{
    public TimeOnly GetTimeNow() =>
                    TimeOnly.FromDateTime(DateTime.Now);
}
```

2. Use dependency injection to separate the dependency from the test method

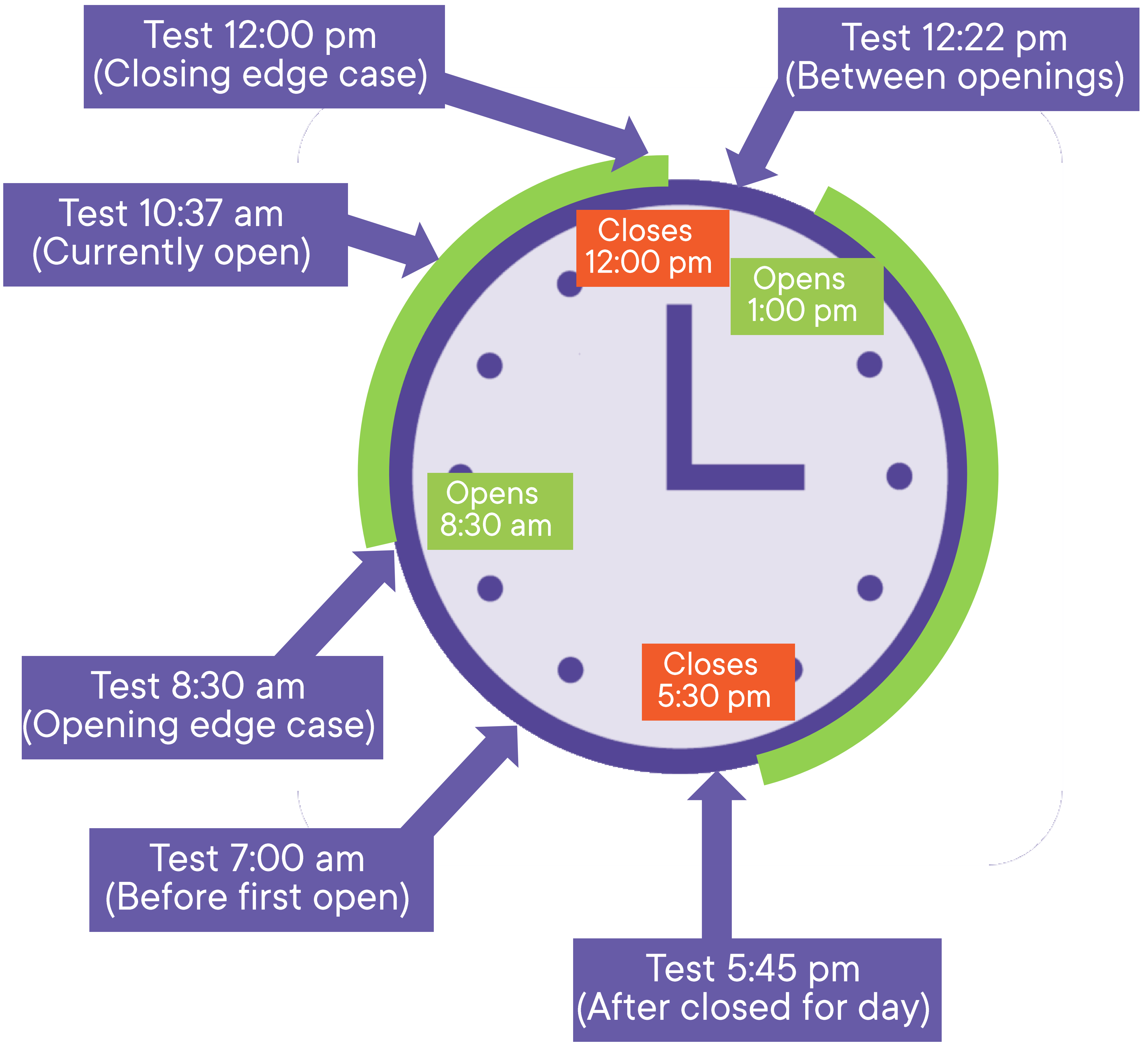
```
public TimeSpan GetTimeUntilNextOpen(
               ITimeNowProvider timeNowProvider)
{
```

3. Write a test double to replace the dependency in the tests

```
public class TimeNowProvider_TestDouble :
ITimeNowProvider
{
```

Choosing Test Data

To Test GetTimeUntilNextOpen()



Assume you don't know how the method is implemented

Think: What other input data might require different logic

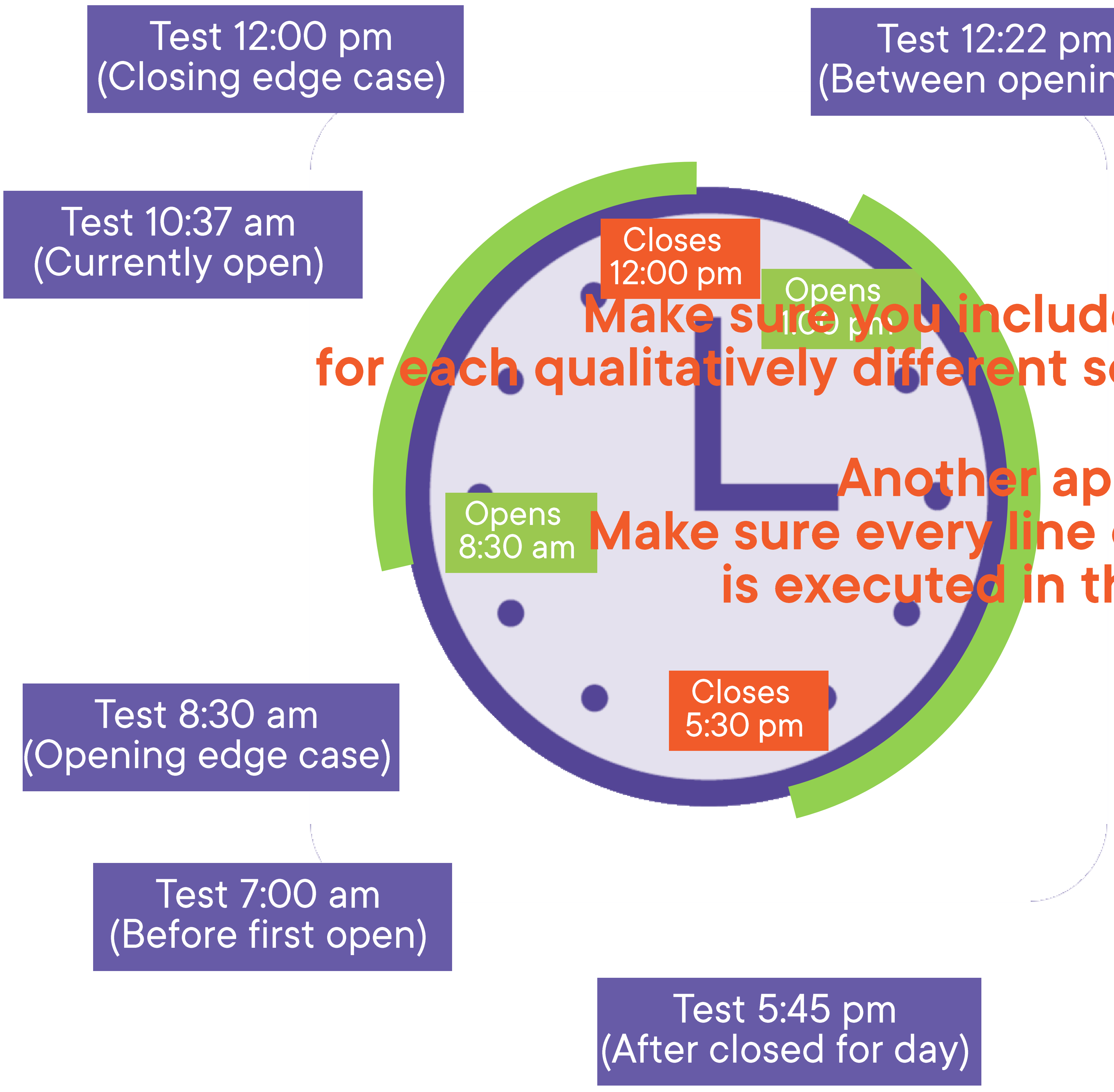
Use test data with different features (On the hour vs. random minute)

Include edge cases

Include invalid data

- Not possible here: All inputs are valid

To Test GetTimeUntilNextOpen()



Make sure you include a test for each qualitatively different scenario

Another approach: Make sure every line of code is executed in the tests

Time to test	Expected answer
	1 hr 30 mins
	zero
	zero
	1 hr
	38 mins
	14 hrs 45 mins (open tomorrow)

Demo

Rewrite test to use all the data values

- Use a data-driven test

Summary

If a class depends on external data:

- Separate the data with dependency injection
- Write a test double (‘mock’) type to replace the dependency in tests

Static dependencies:

- Wrap static method in an interface instance
- Then use the same technique
- Methods with no dependencies can be tested ‘as-is’

Summary

Test data values

- Cover range of situations that method might need to cope with
- Include edge case
- Include invalid data