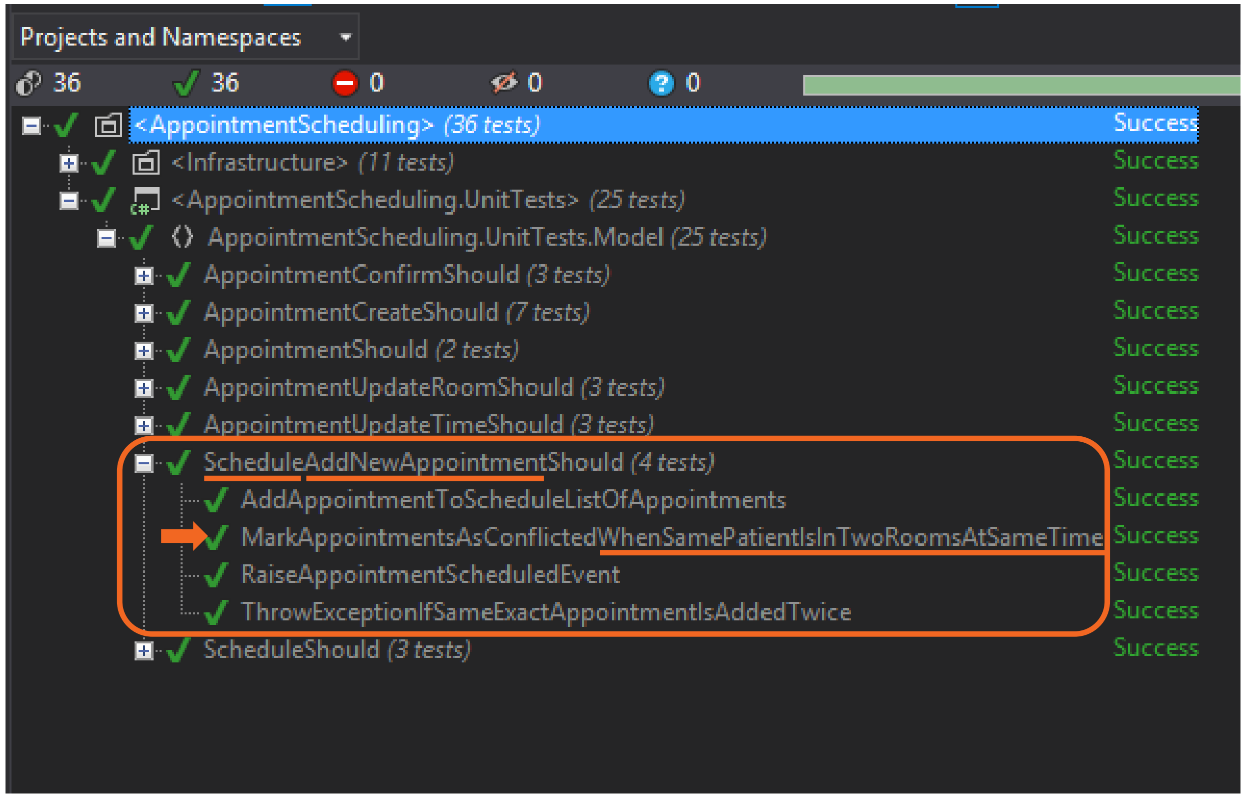
The Fundamentals of MSpec

Kevin Kuebler

@kevinkuebler

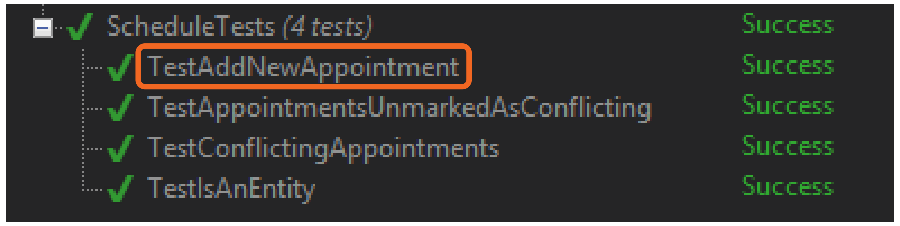
Testing Styles



```
Projects and Namespaces
36   ✓ 36   0   0   0
<AppointmentScheduling> (36 tests)                                      Success
  <Infrastructure> (11 tests)                                           Success
  <AppointmentScheduling.UnitTests> (25 tests)                          Success
    AppointmentScheduling.UnitTests.Model (25 tests)                    Success
      AppointmentConfirmShould (3 tests)                                Success
      AppointmentCreateShould (7 tests)                                 Success
      AppointmentShould (2 tests)                                       Success
      AppointmentUpdateRoomShould (3 tests)                             Success
      AppointmentUpdateTimeShould (3 tests)                             Success
      ScheduleAddNewAppointmentShould (4 tests)                         Success
        AddAppointmentToScheduleListOfAppointments                      Success
        MarkAppointmentsAsConflictedWhenSamePatientIsInTwoRoomsAtSameTime Success
        RaiseAppointmentScheduledEvent                                  Success
        ThrowExceptionIfSameExactAppointmentIsAddedTwice                Success
      ScheduleShould (3 tests)                                          Success
```

Should style test names

Much better than generic tests

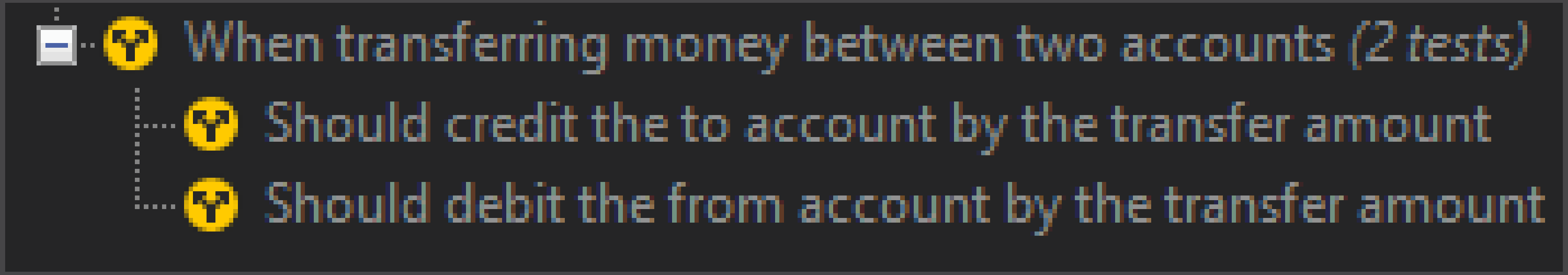


```
ScheduleTests (4 tests)                     Success
  TestAddNewAppointment                     Success
  TestAppointmentsUnmarkedAsConflicting     Success
  TestConflictingAppointments               Success
  TestIsAnEntity                            Success
```

Still implementation specific

MSpec Basics

```
WhenTransferringMoneyBetweenTwoAccounts
public class When_transferring_money_between_two_accounts
{
    It Should_debit_the_from_account_by_the_transfer_amount;
    It Should_credit_the_to_account_by_the_transfer_amount;
}
```



When transferring money between two accounts (2 tests)
- Should credit the to account by the transfer amount
- Should debit the from account by the transfer amount

What’s This **It** Thing Anyway?

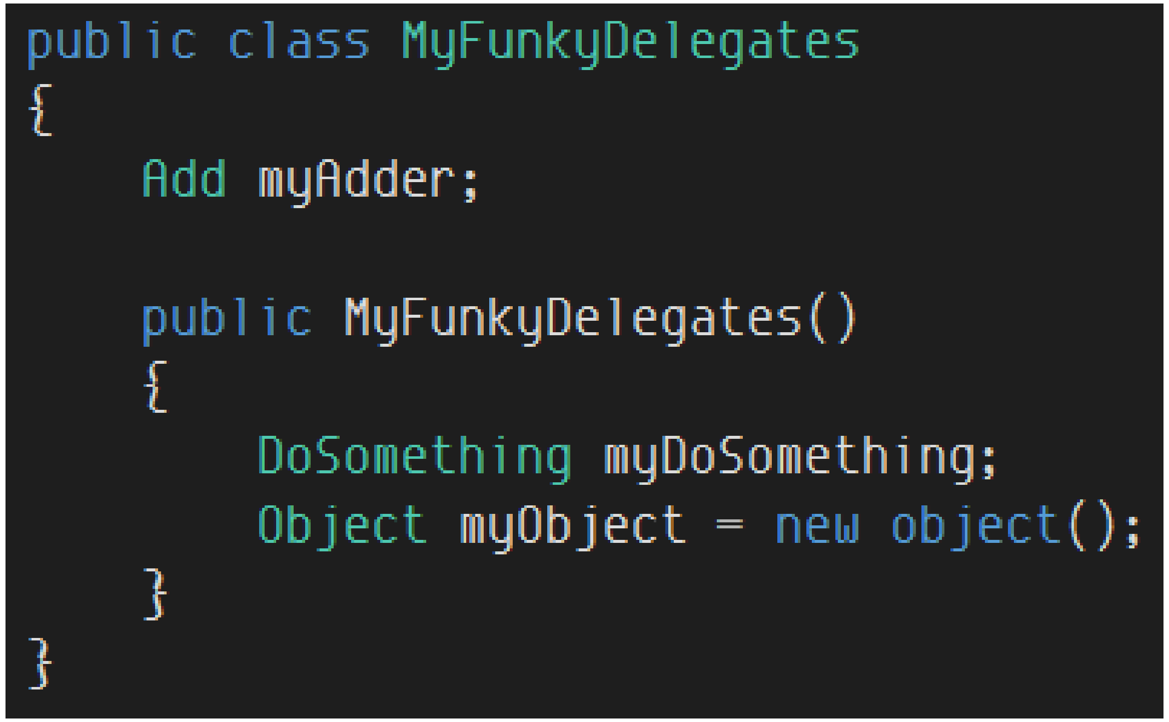
- **It** is a delegate type defined by MSpec
- Delegate: strongly-typed function pointer
- Defining a delegate is like defining a class with a single method

```
public delegate void DoSomething();

public delegate int Add(int first, int second);
```

- Declare instances of delegate types

Using Delegates

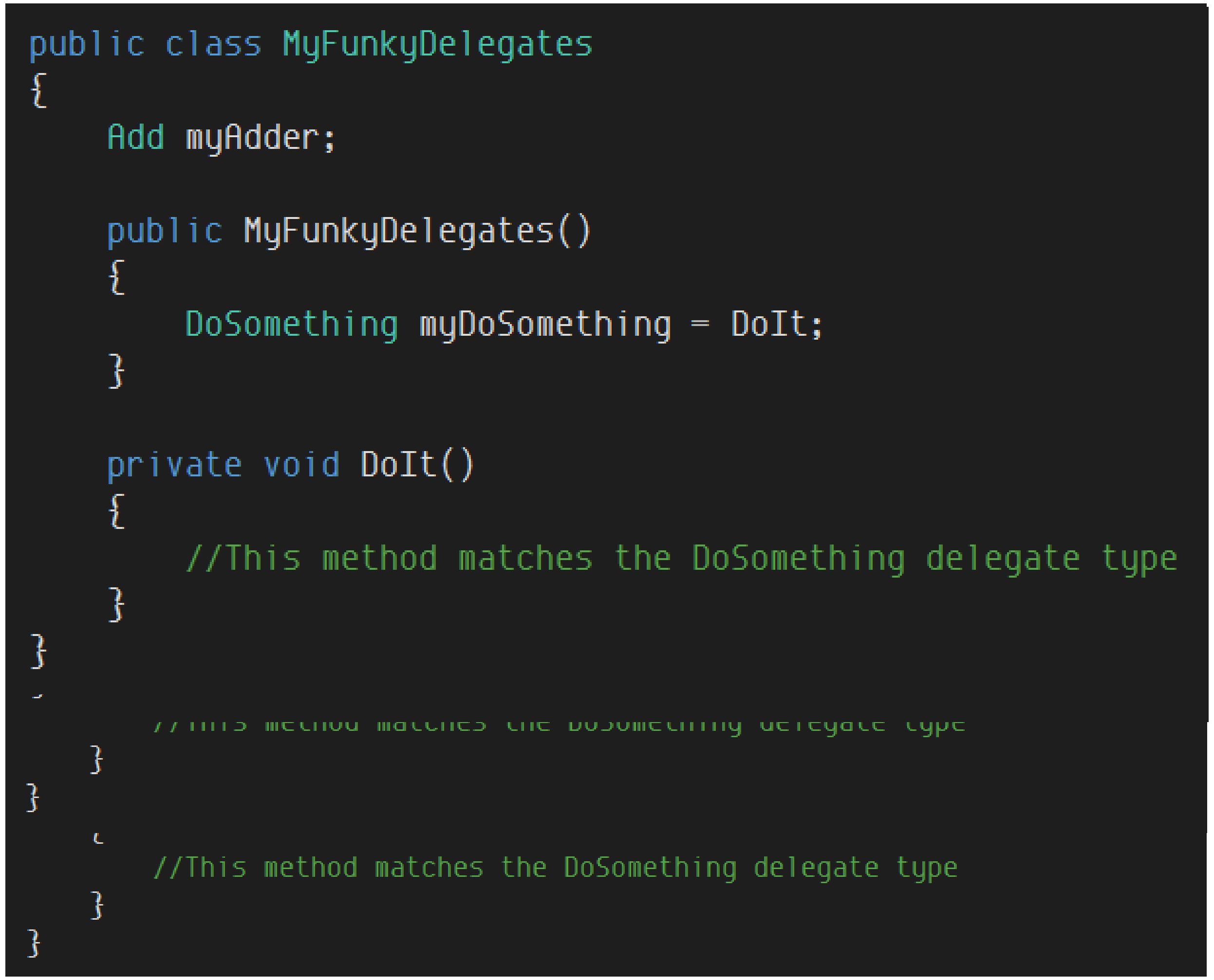


```
public class MyFunkyDelegates
{
    Add myAdder;

    public MyFunkyDelegates()
    {
        DoSomething myDoSomething;
        Object myObject = new object();
    }
}
```

- Delegates can be:
  - Local variables
  - Instance variables
  - Parameters
  - etc.
- What do you assign to a delegate?
  - Named method
  - Anonymous method
  - Lambda expression

Using Delegates



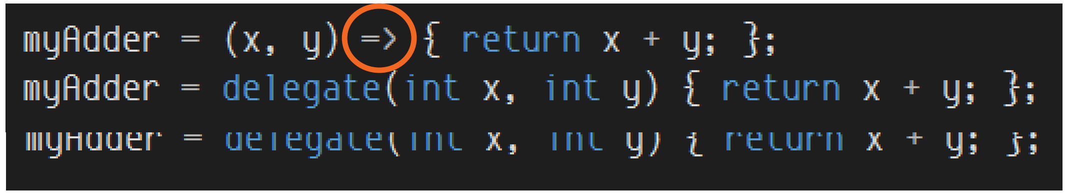
```
public class MyFunkyDelegates
{
    Add myAdder;

    public MyFunkyDelegates()
    {
        DoSomething myDoSomething = DoIt;
    }

    private void DoIt()
    {
        //This method matches the DoSomething delegate type
    }
}
```

```
        //This method matches the DoSomething delegate type
    }
}
    {
        //This method matches the DoSomething delegate type
    }
}
```

- Assign a delegate instance to:
  - Named method/function
  - Anonymous method/function
  - Lambda expression
- Invoking a delegate instance:
  - Invoke method
  - Normal function syntax



```
myAdder = (x, y) => { return x + y; };
myAdder = delegate(int x, int y) { return x + y; };
```

```
myAdder = delegate(int x, int y) { return x + y; };
```

Anatomy of a Specification

```
public class When_transferring_money_between_two_accounts
{
    It Should_debit_the_from_account_by_the_transfer_amount;
}
```

No need for [TestFixture]

No need for [Test]

The MSpec runner will discover and invoke all of the It delegates in your class.

Anatomy of a Specification

```
public class When_transferring_money_between_two_accounts
{
    It Should_debit_the_from_account_by_the_transfer_amount = () =>
    {
        var fromAccount = new Account(1000);
        var toAccount = new Account(2000);
        var transferManager = new Transfer();
        transferManager.TransferFunds(fromAccount, toAccount, 250);
        fromAccount.Balance.ShouldEqual(750);
    }
}
```

Anatomy of a Specification

```
public class When_transferring_money_between_two_accounts
{
    It Should_debit_the_from_account_by_the_transfer_amount = () =>
    {
        //Arrange
        var fromAccount = new Account(1000);
        var toAccount = new Account(2000);
        var transferManager = new Transfer();
        //Act
        transferManager.TransferFunds(fromAccount, toAccount, 250);
        //Assert
        fromAccount.Balance.ShouldEqual(750);
    }
}
```

Anatomy of a Specification

```
public class When_transferring_money_between_two_accounts
{
   It Should_debit_the_from_account_by_the_transfer_amount = () =>
   {
      var fromAccount = new Account(1000);
      var toAccount = new Account(2000);
      var transferManager = new Transfer ();
      transferManager.TransferFunds(fromAccount, toAccount, 250);
      fromAccount.Balance.ShouldEqual(750);
   }
   It Should_credit_the_to_account_by_the_transfer_amount = () =>
   { //Same arrange and act steps }
}
```

Anatomy of a Specification

```
public class When_transferring_money_between_two_accounts
{
   static Account FromAccount;
   static Account ToAccount;
   static Transfer TransferManager;

   Establish context = () =>
   {
      FromAccount = new Account(1000);
      ToAccount = new Account(2000);
      TransferManager = new Transfer();
   };
}
```

Anatomy of a Specification

```
public class When_transferring_money_between_two_accounts
{
    static Account FromAccount;
    static Account ToAccount;
    static Transfer TransferManager;

    Establish context = () =>
    {
        FromAccount = new Account(1000);
        ToAccount = new Account(2000);
        TransferManager = new Transfer();
    };
}
```

Establish delegate like [TestFixtureSetup] attribute – invoked once before any **It** delegates in the class are invoked

Anatomy of a Specification

```
public class When_transferring_money_between_two_accounts
{
    static Account FromAccount;
    static Account ToAccount;
    static Transfer TransferManager;

    Establish context = () =>
    {
        FromAccount = new Account(1000);
        ToAccount = new Account(2000);
        TransferManager = new Transfer();
    };
}
```

Name of **Establish** delegate instance can be anything, but usually "context" by convention.

Anatomy of a Specification

```
public class When_transferring_money_between_two_accounts
{
    static Account FromAccount;
    static Account ToAccount;
    static Transfer TransferManager;

    Establish context = () =>
    {
        FromAccount = new Account(1000);
        ToAccount = new Account(2000);
        TransferManager = new Transfer();
    };
}
```

Fields need to be **static** to be accessible from the lambda expressions.

Anatomy of a Specification

```
public class When_transferring_money_between_two_accounts
{
   //Fields and Establish delegate ...

   Because of = () => TransferManager.TransferFunds(fromAccount,toAccount,250);

   It Should_debit_the_from_account_by_the_transfer_amount = () =>
      FromAccount.Balance.ShouldEqual(750);

   It Should_credit_the_to_account_by_the_transfer_amount = () =>
      ToAccount.Balance.ShouldEqual(2250);
}
```

Aim for one line **Because** and **It** delegates (with some exceptions).

Refactoring Specifications

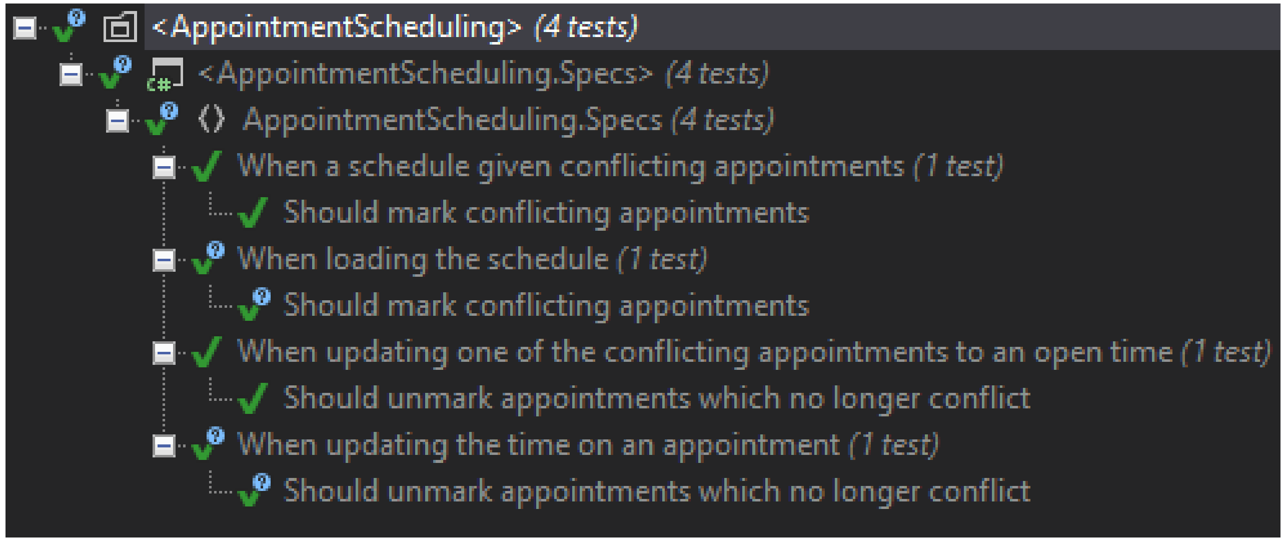
- Two very similar contexts, mostly duplicated code
- How to stay DRY?
  - Don't Repeat Yourself
  - Does DRY matter?
- <opinion>
  DRY is a fundamental principle and applies to test code as well as production code
  </opinion>
- Two techniques for MSpec
  - Inheritance
  - Nested contexts

Summary

- Name elements according to intent and behavior of system, not specific code elements
- MSpec provides 3 delegate types for your specs
  - Establish
  - Because

  Establishing the context

  - It Actual specification/assertion
- Multiple It delegates for separate assertions about one context
- Keep your code DRY
  - Inheritance
  - Nested contexts



```
<AppointmentScheduling> (4 tests)
  <AppointmentScheduling.Specs> (4 tests)
    AppointmentScheduling.Specs (4 tests)
      When a schedule given conflicting appointments (1 test)
        Should mark conflicting appointments
      When loading the schedule (1 test)
        Should mark conflicting appointments
      When updating one of the conflicting appointments to an open time (1 test)
        Should unmark appointments which no longer conflict
      When updating the time on an appointment (1 test)
        Should unmark appointments which no longer conflict
```