```
//..
namespace BillingSystem.Specs {
  [Trait("Monthly Billing", "Payment Is
Received")]
  public class PaymentReceived{
    //... the constructor prepares the test data
    [Fact]
    public void An_Invoice_Is_Created(){
        //...
```

In this code you can see the features directly by examining the Trait attribute on the test class. The first element is the feature ("Monthly Billing"), the second is the scenario ("Payment Is Received"). Your testing library might have a different way for specifying these things.

SO WHAT?

You might be wondering, at this point, why all of this even matters? BDD does have a number of advantages:

» **Readability**. It sounds idealistic, but being able to print out a test run and read, in common language, what's going on is quite powerful.

» **Ubiquity**. I hate that word, but it's applicable here: you know what these tests describe and your client/boss will know as well. In this, you're speaking the same language.

» **Focus**. If you're focused on how your application behaves, you're aligning yourself with the business goals. This is important for programmers! You can watch your application evolve into something exciting and understand why it's doing what it's doing. You might even have some questions about this, which means you can contribute your genius to the application's design.