The first thing I want to improve in this method is how we construct arrival or departure data. So you can see in every test method, you're repeating these 2 pm or 10 am times. This is creating extra noise in the code, and it doesn't add any value. So I'm going to create 2 helper methods in this test class to construct arrival or departure date times. So, we create a private helper method, and put them all at the end of this class, because we don't them to pollute our test code.

So, this method returns a DateTime object. And we're going to call them ArriveOn so it takes a year, month, and date without hour, minute, and seconds. So here we simply return a new DateTime with year, month, and day, and set the hour to 14, minutes to 0 and seconds to 0. Similarly, we're going to create another method called DepartOn, and create a DateTime object for that date, let's set the time to 10 am. With these two methods, we can name our test method a little bit cleaner. So, look what happens. We can replace this with ArriveOn. And get rid of these three magic numbers, look, it's already much better. And similarly, DepartOn and then we get rid of these 3 numbers. one more time, ArriveOn let's clean this up, and finally, Depart on Okay, now, it's a little bit better, but we're not done yet. It would be nicer if we could construct the ArrivalDate and DepartureDate for this new booking relative to the existing booking. So instead of having four different dates here, it would be nice if we could have a helper method like

Before, and then we pass existingBooking. .ArrivalDate. This way we'll have less magic numbers in our test, and our code clearly reveals the intention, which is the scenario we're testing here, booking starts and finishes before an existing booking. So, in order to get here, we should extract this booking into a private field in this class. And this adds another value, because for other test methods we're going to create here, we don't have to repeat creating this booking object.

Because to make this test simple, I want to use the same booking, the same existing booking for 15th to 20th of January. And I only want to modify the arrival and departure date of the new booking. So, why this is selected, here we can refactor this code by pressing alt and enter.

It's going to refactor, and Introduce Field. So, look, Rider created a private field called underline booking, and initialized it here in our test method, and then simply referenced it in the list of existing bookings.

Now, we don't want to keep the initialization of this field here, because we don't want to repeat it in every test. So, we put it in our SetUp method. So, let's create a SetUp method, public void SetUp and here we simply initialize this booking object. Now, with this SetUp method in place, we can also move these few lines in the SetUp method, because we're going to repeat these lines in every test method and that's really bad.

So, let's cut these few lines, and put them inside the SetUp method. Now, we need to promote this repository object here, which is a local object in this method, we need to promote this to a private field, so we can reference it here. So one more time, I'm going to refactor this code and introduce a private field. So alt and enter, refactor, introduce field. So, now, look we have another private field here, called \_repository. Now finally, we can reference that object here. Look, our test method is already way shorter, it's only 5 lines of code. Now finally,

let's add this helper method, before and then after It's very easy to implement. So, private DateTime Before it gets a DateTime object. That's simply returns dateTime.Add Days, -1, so we can shift it one day before. Okay? With this, we can change this line to something like this. Now we need to change this to \_booking.ArrivalDate, like this.

Actually it's better to rename this field and call it ExistingBooking.

Because it shows the intention very clearly. So, let's just name these two, existingBooking. Now similarly, let's create another method called After. So, let's select this, duplicate and call it after, and here we simply get this DateTime object, and add 1 day to it. Now we're not going to use this method in this test case, but we're going to use it when writing future tests.

Now there is one tiny problem here. If we use our before method here, we'll end up with ArrivalDate and DepartureDate being equal. So, I want to allow the consumer of this method to pass the number of days before a given date.

So, let's add a parameter called days and here we replace one with days. Now, to make it easier to call this method we can also set a default value for this for this parameter, we can set it to 1, now here, when calling the before method, we can pass 2 as an argument. However, someone looking at this code, they may not know straight away what 2 represents here.

So, we can use name arguments in C#, days: 2. Now it's very clear, so 2 days, before the arrival date of existing booking. Now similarly I'm going to replace this with Beforeexisting Booking.arrival date, but this time I'm not going to pass days, because by default days is 1.

So this is the end result, our test method is very short, only a few lines of code, and the code clearly reveals an intention. Now, let's run this test and make sure we heave not broken anything during our refactorings. Okay, the test is passing beautiful. Next, we're going to look at the second test case.