CHAPTER 15

Tests Involving the Date and Time

Our group continues to evolve the rental tests from the previous chapter, finding that it's good to use a common `SetUp` while avoiding unnecessary interactions between tests. The group finds that it's easy, after all, to write tests that depend on the date and time.

When tackling business transactions, the group makes such transactions explicit in the tests. Finally, the group develops *sad-path* tests, which check when things go wrong.

> **Programmers:** The fixture code for the tests in this chapter is given in Chapters 33 and 34.

15.1 Introduction

On Thursday, Emily, Don, and Sarah continued with the tests. Emily suggested that they start with deposits: "That seems simple and would be a good warmup before tackling date and time." Don agrees: "I'm not sure how to test when the system is affected by the date and time."

In the past, they hadn't automated time-based tests. "Doing the tests manually is no fun either," said Don, "as a test can't be completed until the right time has elapsed. Lots of careful checks are needed, which is time consuming. So after changes to *RentEz*, the tests were often not done and then often not as well as they could be."

We began with charging deposits.

15.2 Charging a Deposit

The general business rule here is as follows.

> As well as the rental charge, a client pays a deposit before taking the rental items. When the client returns the items, the deposit is refunded if the items are in good condition and don't need replacement, fixing, or cleaning. Otherwise, a partial refund is made, or the client is asked to pay extra for the damage.

The group's first step was to add deposit information in the `SetUp` to the HireItems, as last shown in Figure 14.6 on p. 110. After some debate, Don made the deposit $0.00 for the two dispensers; that meant that the previous tests should still pass. As Don argued, "It would be great if we could add more data to the `SetUp` without breaking the old tests." The others agreed.

Don added cups as an item with a deposit and included the three rental rates: hourly, daily, and weekly, as shown in Figure 15.1.

rent.StartApplication

enter rental items					
name	*count*	*$/hour*	*$/day*	*$/week*	*deposit*
coffee dispenser	10	1.50	8.20	60.00	0.00
hot water dispenser	12	1.50	8.00	50.00	0.00
cup	500	0.05	0.45	2.00	0.10

enter clients	
name	*phone*
Joanna	373 7599

client	Joanna

Figure 15.1 Extend the `SetUp`

Sarah wrote the test shown in Figure 15.2. She then questioned whether it made sense to have only a ***`weeks`*** column in those tables: "Will we need ***`hours`***, ***`days`***, and ***`weeks`***?" "Let's wait until we add date/time processing and see what happens," Emily suggested.

rent	100		cup	*for*	2	*weeks*
check	cost	210.00				

rentals for client	Joanna	
rental item	*count*	*weeks*
cup	100	2

Figure 15.2 Rent with a Deposit

"This test is incomplete," Don then pointed out. "We should also check that there are 100 fewer cups in stock afterward." He extended Figure 15.2 to give Figure 15.3.

Immediately, Don was unhappy with that: "We need to check only the count." Emily added, "And we'd be better checking just the cups, not the other items. Otherwise, when we change the `SetUp` again, the last table (in Figure 15.3) will

rent	100		cup	*for*	2	*weeks*
check	cost	210.00				

rentals for client	Joanna	
rental item	*count*	*weeks*
cup	100	2

rental item list					
name	*count*	*$/hour*	*$/day*	*$/week*	*deposit*
coffee dispenser	10	1.50	8.20	60.00	0.00
hot water dispenser	12	1.50	8.00	50.00	0.00
cup	400	0.05	0.45	2.00	0.10

Figure 15.3 Rent with a Deposit with Extra Checking Table

fail." Don changed the table to give Figure 15.4, using a `SubsetFixture`.[1] He pointed out, "We'll need to have at least one test that checks that the other rental items are not altered at all."

rent	100		cup	*for*	2	*weeks*
check	cost	210.00				

rentals for client	Joanna	
rental item	*count*	*weeks*
cup	100	2

rental item subset list	
name	*count*
cup	400

Figure 15.4 Revised Last Table

"We need to write a test for returning the deposit when the rental items are returned," Emily noted. However, she realized, "We really haven't handled payments yet, let alone refunds. Let's leave that until after we do dates."

Questions & Answers

Do you tend to have lots of `SetUps` or only one for all the tests?

For RentAPartySoftware, a common `SetUp` was used for almost all the tests. This avoided the need to make up new data for each test, which gets tedious, and it's difficult to remember the particular data. With common `SetUp`, we could keep a printed copy on hand.

I suppose the downside is that we may have had a little less variation in our tests than we'd have liked. No, maybe not, when I think about it some more.

[1] See Section 10.7 on p. 79 for an introduction to `SubsetFixture` tables.

In general, it depends on the system under test and how many distinct areas of the application there are. It may be that most tests share some common data, and then categories of tests share still more, so it's convenient to have automated support in gathering up the right `SetUp` to use for a particular test.

15.3 Dates

"We need an action that sets the date. Then we can control the time whenever we want," Sarah offered. After trying several action words, she suggested setting the date as shown in Figure 15.5.

time is now	2004/05/06 09:01

Figure 15.5 An Action to Set the Date

After Sarah added the time-change action to several tests, Emily suggested that she "put it at the end of the `SetUp`, as all tests will need date/time set." Sarah added that to Figure 15.1. She then changed the rental test in Figure 15.4 to become the test shown in Figure 15.6, having added in ***`hours`*** and ***`days`*** to the second table.

rent	100		cup	*for*	2	*weeks*
check	cost	210.00				

rentals for client	Joanna				
rental item	*count*	*start date*	*hours*	*days*	*weeks*
cup	100	2004/05/06 09:01	0	0	2

rental item subset list	
name	*count*
cup	400

Figure 15.6 Test Rental with Dates

"Why not use an end date as well?" Don suggested. He changed the second table of the test, as shown in Figure 15.7.

rentals for client	Joanna		
rental item	*count*	*start date*	*end date*
cup	100	2004/05/06 09:01	2004/05/20 09:01

Figure 15.7 Revised Second Table of Test Rental with Dates

Questions & Answers

It's easy to write the test, but making it work is not so easy.
Yes, that's true, as we'll see in Chapter 34. But the payback from doing so is immense, as we can then write automated tests that pick up errors soon after they're made.

In the medium term, having test support and higher quality in the software can make a huge difference in the speed at which changes can be made. And being able to communicate about what's needed, using Fit tests to express that, is also so valuable; it helps to focus the effort where it's needed.

15.4 Business Transactions

It was time for Sarah, Emily, and Don to deal with the money, but they weren't sure of all the details. So Mitra, a business analyst, was asked to join the group to help with that and to gain experience with Fit tests.

As Mitra explained: "Customers may start saying what they want to rent, which is entered by the staff member serving them. But when they find that some item is not available, they may cancel everything. We have to ensure that they get the rental items if they want them, so we handle this with a business transaction." "And they can cancel part of it at any time," Don added. "So we need some way of handling that in our tests."

"For a transaction to be completed," Mitra continued, "customers have to have paid or had the amount added to their charge accounts. To cancel after they've paid, they have to first have that payment refunded, or it will be automatically removed as a charge on their accounts."

"So we need to be explicit about the transaction as a business object in our test," said Sarah, "so how about this for the action part of the test?" She sketched out the tables in Figure 15.8: "I've put all the actions to do with the transaction into a single table, as they operate within the context of that transaction."

begin transaction for client	Joanna					
rent	100		cup	*for*	2	*weeks*
check	*total is $*	210.00				
pay with cash $	210.00					
check	*total is $*	0.00				
complete transaction						

Figure 15.8 Test Rental Within a Transaction

Questions & Answers

What do you mean by "business object"?

In the business domain, that's something that is relevant to our system and that we want to be clear about. Obvious business objects are things like rental items, apartments, vehicles, and accounts. Less obvious are things like time periods and transactions. In RentEz, it doesn't seem as though transactions are to do with the large-scale business, but they certainly underlie much of the business workflow.

We introduced testing of such business objects in Section 10.3 on p. 75.

"There's no need to check the total before and after paying," Mitra recommended. "If the payment isn't for the full amount of the rental cost, the transaction cannot be completed. And the transaction mentions the customer, so there's no need to do that separately in the `SetUp`."

"We'll need to write some tests for the transaction failing to complete," added Don. "Let's leave that until later." (We cover that in the next section.)

Sarah changed the `SetUp` to include information about staff, as shown in Figure 15.9.

rent.StartApplication

setup					
rental item name	*count*	*$/hour*	*$/day*	*$/week*	*deposit*
coffee dispenser	10	1.50	8.20	60.00	0.00
hot water dispenser	12	1.50	8.00	50.00	0.00
cup	500	0.05	0.45	2.00	0.10

setup	
client name	*phone*
Joanna	373 7599

setup	
staff name	*phone*
Bill	555 9876

time is now	2004/05/06 09:01

Figure 15.9 Revised `SetUp`

She then removed the amount checks from Figure 15.8, resulting in the test shown in Figure 15.10, which assumes the `SetUp` in Figure 15.9, as usual. The last table in Figure 15.10 was also changed by Don to include staff member Bill in the transaction.

begin transaction for client	Joanna	*staff*	Bill			
rent	100		cup	*for*	2	*weeks*
pay with cash $	210.00					
complete transaction						

rentals for client	Joanna		
rental item	*count*	*start date*	*end date*
cup	100	2004/05/06 09:01	2004/05/20 09:01

rental item subset	
name	*count*
cup	400

Figure 15.10 Revised Test Rental Within a Transaction

15.5 Sad Paths

Don was eager to focus on "when the transaction can't complete." In testing, this is often called a *sad path* because it concerns what happens when things don't work out well. After discussion and a few trials, the group came up with the first sad-path test, as shown in Figure 15.11.

begin transaction for client	Joanna	*staff*	Bill			
rent	100		cup	*for*	2	*weeks*
reject	*complete transaction*					
pay with cash $	210.00					
complete transaction						

Figure 15.11 Testing a Transaction That Can't Complete

No payment has been made in the third row of Figure 15.11. So the *`complete transaction`* in the third table must be rejected by the system under test. Once the payment has been made in the fourth row, the transaction can be completed.

The group wrote 20 more sad-path tests for *complete* and *cancel* transactions, including the one shown in Figure 15.12. A transaction can't be canceled until any paid money is refunded.

begin transaction for client	Joanna	*staff*	Bill			
rent	100		cup	*for*	2	*weeks*
pay with cash $	410.00					
reject	*cancel transaction*					
refund cash $	410.00					
cancel transaction						

Figure 15.12 Testing a Transaction That Can't Be Canceled

Questions & Answers

But doesn't the test in Figure 15.11 consist of two tests?
Yes, you're right, with the first test simply checking that the *`complete transaction`* was rejected. We also need to check that a previously rejected completion of a transaction can be completed later.

However, the test is so short that splitting it into two doesn't seem worthwhile, especially as there are no other cases yet. There's usually a judgment call and often a tradeoff in such decisions, and people differ in their approach to this issue.

15.6 Reports

By the following Wednesday, Mitra and Emily had finished developing the fixtures for these tests, as discussed in Chapter 34. Everyone was eager to see the results. The report for Figure 15.10 is shown in Figures 15.13 and 15.14.

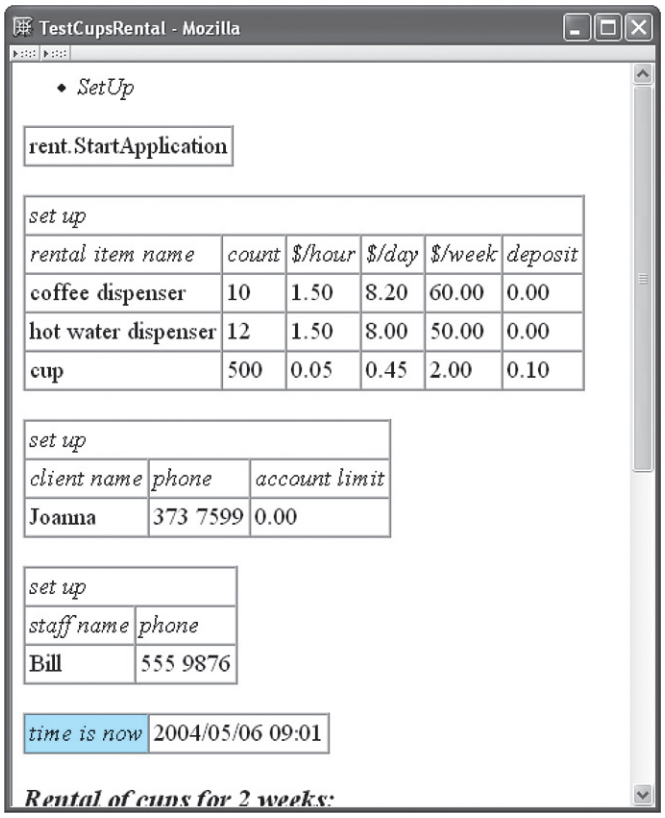

Figure 15.13 First Half of the Report for Figure 15.10

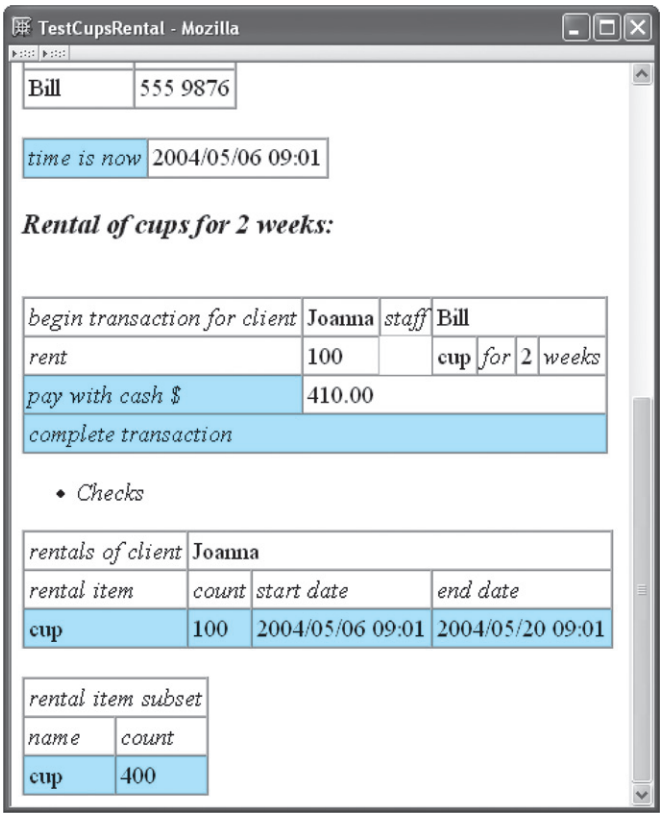

Figure 15.14 Second Half of the Report for Figure 15.10

Questions & Answers

Why are so many of the actions in Figures 15.13 and 15.14 green?

Because each of those actions may fail, under various circumstances; for example, a transaction can't be started with an unknown client or staff member.

(The fixture needs to be defined appropriately by a programmer to have this effect, as we discuss in Chapter 28.)

So why don't you explicitly check that they succeed?

We could do that. It's just a matter of personal style; the group preferred to keep the actions brief.

The report for Figure 15.12 is shown in Figure 15.15.

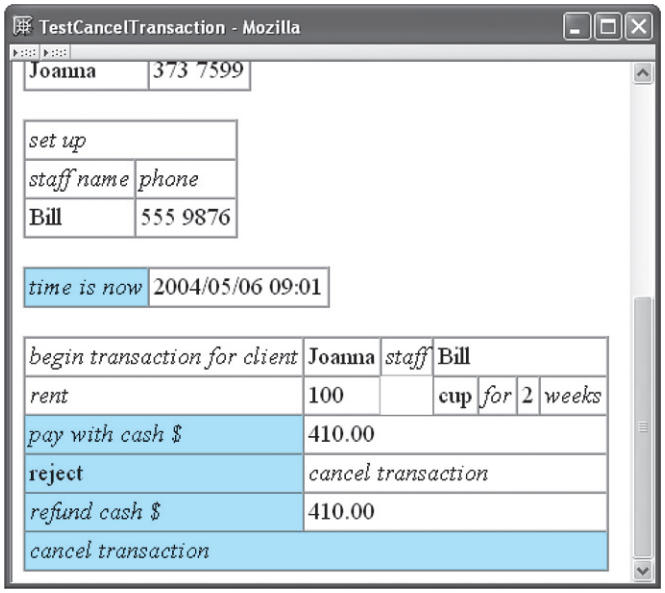

Figure 15.15 Part of the Report for Figure 15.12

15.7 Summary

- It's convenient to use a common `SetUp` for a set of tests. But as data is added to the `SetUp`, it's good to avoid breaking existing tests. We saw several ways of doing that.
- It's straightforward to write tests that set the date and time.
- It's useful to make business objects, such as business transactions, explicit in the tables in the tests. For example, user operations as a part of the transaction, such as making payments, are treated as actions in the *`transaction`* table in the test.
- We showed some examples of *sad-path* tests, which check when things go wrong.

15.8 Exercises

Write tests for the following situations.

1. A transaction may be canceled even when nothing has happened.
2. A transaction may be completed even when nothing has happened.
3. A partial payment may be made in a transaction, but the transaction cannot be completed until the rest of the payment is made.

4. A partial refund may be made in a transaction in which a payment has been made. The transaction cannot be canceled until the rest of the refund is made.
5. An overpayment is not permitted in a transaction.
6. An overrefund is not permitted in a transaction.
7. A negative payment, or refund, is not permitted.
8. A negative count, or time period, is not permitted in a *`rent`* action in a *`transaction`*.