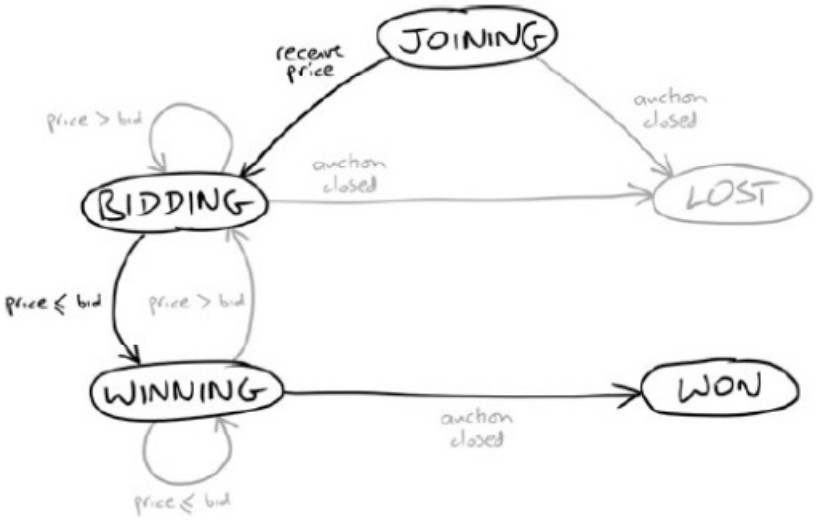
Chương 14: Sniper thắng cuộc

Ở đây chúng tôi thêm một tính năng mới cho Sniper, chúng tôi test nó bằng cách nghe các lệnh callback của nó. Chúng tôi thấy rằng, ngay ở giai đoạn đầu này, công sức dọn dẹp của chúng tôi đã mang lại thành quả.

Đầu tiên, một test fail

Chúng tôi đã có một Sniper có thể phản ứng với giá tăng bằng cách bid cao hơn, nhưng nó còn chưa biết khi nó thành công. Feature tiếp theo tại to-do-list là thắng một cuộc đấu giá. Tính năng này đòi hỏi một kiểu chuyển trạng thái mới, như minh họa trong Hình 14.1



Hình 14.1: Một sniper đặt bid rồi thắng cuộc.

Để diễn đạt nội dung này, chúng tôi thêm một test end-to-end dựa theo sniperMakesAHigherBidButLoses() nhưng với một kết luận khác—sniperWinsAnAuctionByBiddingHigher(). Test mới có nội dung như sau, với tính năng mới được đánh đậm:

```
public class AuctionSniperEndToEndTest { […]
  @Test
  public void sniperWinsAnAuctionByBiddingHigher() throws Exception {
    auction.startSellingItem();

    application.startBiddingIn(auction);
    auction.hasReceivedJoinRequestFrom(ApplicationRunner.SNIPER_XMPP_ID);

    auction.reportPrice(1000, 98, "other bidder");
    application.hasShownSniperIsBidding();

    auction.hasReceivedBid(1098, ApplicationRunner.SNIPER_XMPP_ID);

    auction.reportPrice(1098, 97, ApplicationRunner.SNIPER_XMPP_ID);
    application.hasShownSniperIsWinning();
```

```
    auction.announceClosed();
    application.showsSniperHasWonAuction();
  }
}
```

Tại cơ sở hạ tầng test, chúng tôi thêm vào ApplicationRunner hai phương thức kiểm tra xem ui có hiển thị trạng thái mới hay không. Thông báo lỗi mới như sau:

```
java.lang.AssertionError:
Tried to look for...
  exactly 1 JLabel (with name "sniper status")
  in exactly 1 JFrame (with name "Auction Sniper Main" and showing on screen)
  in all top level windows
and check that its label text is "Winning"
but...
  all top level windows
  contained 1 JFrame (with name "Auction Sniper Main" and showing on screen)
  contained 1 JLabel (with name "sniper status")
  label text was "Bidding"
```

Giờ chúng tôi biết mình đang đi đâu và có thể bắt đầu cài feature mới.

Ai biết về các Bidder?

Ứng dụng biết rằng: Sniper đang ở thế thắng (winning) nếu nó chính là bidder vừa đặt bid mới nhất mà auction vừa nhận được. Chúng tôi phải quyết định xem nên đặt logic đó ở chỗ nào. Xem lại Hình 13.5, một lựa chọn là translator có thể truyền bidder cho Sniper và để Sniper quyết định. Điều đó có nghĩa Sniper sẽ phải biết gì đó về việc auction định danh các bidder như thế nào, với một nguy cơ phải dính đến các chi tiết về XMPP mà chúng tôi đã cẩn thận giữ chúng tách biệt. Để xác định xem mình có đang thắng hay không, điều duy nhất mà Sniper cần biết là khi nhận được một message Price, giá đó có phải là thứ mình vừa đặt hay không. Thông tin này chỉ là "từ Sniper" hoặc "không phải từ Sniper" , nó không phải là một ID, do đó chúng tôi sẽ biểu diễn nó bằng một kiểu enum PriceSource đặt trong AuctionEventListener.

Tình cờ, PriceSource cũng là một ví dụ về value type. Chúng tôi muốn code mô tả domain của Sniper, chứ không phải cái gì đó chẳng hạn như một giá trị boolean mà chúng tôi sẽ phải giải nghĩa mỗi lần đọc nó; (mục "Value Types trang 59" thảo luận kĩ hơn về chủ đề này).

```
public interface AuctionEventListener extends EventListener {
  enum PriceSource {
    FromSniper, FromOtherBidder;
  };
  […]
```

Chúng tôi cho rằng việc xác định xem đây là giá của tôi hay giá của người khác là một phần trong nhiệm vụ của translator. Do đó, chúng tôi mở rộng phương thức currentPrice(): thêm một tham số mới và sửa unit test cho translator; lưu ý rằng chúng tôi cũng sửa cả tên của test để bao hàm nội dung về feature mới. Chúng tôi cũng nhân cơ hội này truyền ID của Sniper (SNIPER_ID) cho translator. Nó hoàn tất setup đối với input message cho translator tại test thứ hai.

```
public class AuctionMessageTranslatorTest { […]
  private final AuctionMessageTranslator translator = new AuctionMessageTranslator(SNIPER_ID, listener);

  @Test
  public void notifiesBidDetailsWhenCurrentPriceMessageReceivedFromOtherBidder() {
    context.checking(new Expectations() {{
      exactly(1).of(listener).currentPrice(192, 7, PriceSource.FromOtherBidder);
    }});
    Message message = new Message();
    message.setBody(
        "SOLVersion: 1.1; Event: PRICE; CurrentPrice: 192; Increment: 7; Bidder: Someone else;");
    translator.processMessage(UNUSED_CHAT, message);
  }

  @Test
  public void notifiesBidDetailsWhenCurrentPriceMessageReceivedFromSniper() {
    context.checking(new Expectations() {{
      exactly(1).of(listener).currentPrice(234, 5, PriceSource.FromSniper);
    }});
    Message message = new Message();
    message.setBody(
         "SOLVersion: 1.1; Event: PRICE; CurrentPrice: 234; Increment: 5; Bidder: " + SNIPER_ID + ";");
    translator.processMessage(UNUSED_CHAT, message);
  }
}
```

Test mới fail như sau:

```
unexpected invocation:
  auctionEventListener.currentPrice(<192>, <7>, <FromOtherBidder>)
expectations:
! expected once, never invoked:
    auctionEventListener.currentPrice(<192>, <7>, <FromSniper>)
      parameter 0 matched: <192>
      parameter 1 matched: <7>
      parameter 2 did not match: <FromSniper>, because was <FromOtherBidder>
what happened before this: nothing!
```

Cách sửa lỗi là so sánh ID của Sniper với ID của bidder được cho trong event message.

```
public class AuctionMessageTranslator implements MessageListener { […]
  private final String sniperId;

  public void processMessage(Chat chat, Message message) {
    […]
    } else if (EVENT_TYPE_PRICE.equals(type)) {
      listener.currentPrice(event.currentPrice(),
                            event.increment(),
                            event.isFrom(sniperId));
    }
  }

  public static class AuctionEvent { […]
    public PriceSource isFrom(String sniperId) {
      return sniperId.equals(bidder()) ? FromSniper : FromOtherBidder;
    }
    private String bidder() { return get("Bidder"); }
  }
}
```

Công việc dọn dẹp mà chúng tôi đã làm tại mục “Dọn dẹp translator” (tr 135) để tách rời các trách nhiệm khác nhau bên trong translator đã mang lại thành quả ở đây. Tất cả những gì chúng tôi phải làm là thêm một hai phương thức vào AuctionEvent để có được một lời giải rất dễ đọc.
Cuối cùng, để tất cả code đều dịch được, chúng tôi sửa joinAuction() tại Main để truyền tham số ID vào constructor của translator. Chúng tôi có thể lấy được từ kết nối một ID với cấu trúc đúng.

```
private void joinAuction(XMPPConnection connection, String itemId) {
  […]
  Auction auction = new XMPPAuction(chat);
  chat.addMessageListener(
         new AuctionMessageTranslator(
               connection.getUser(),
               new AuctionSniper(auction, new SniperStateDisplayer())));
  auction.join();
}
```

Sniper Has More to Say

Lỗi của test end-to-end hiện tại cho chúng tôi biết rằng chúng tôi cần làm cho UI hiển thị trạng thái Sniper đang thắng. Bước tiếp theo của chúng tôi là đi tiếp theo hướng đó: sửa AuctionSniper để nó giải nghĩa tham số isFromSniper mà chúng tôi vừa tạo thêm. Một lần nữa, chúng tôi bắt đầu bằng một unit test.

```
public class AuctionSniperTest { […]
  @Test
  public void reportsIsWinningWhenCurrentPriceComesFromSniper() {
    context.checking(new Expectations() {{
      atLeast(1).of(sniperListener).sniperWinning();
    }});
    sniper.currentPrice(123, 45, PriceSource.FromSniper);
  }
}
```

Để compile được, chúng tôi thêm vào SniperListener một phương thức mới sniperWinning(). Kéo theo việc bổ sung một cài đặt rỗng vào lớp SniperStateDisplayer.
Test fail:

```
unexpected invocation: auction.bid(<168>)
expectations:
! expected at least 1 time, never invoked: sniperListener.sniperWinning()
what happened before this: nothing!
```

Lỗi trên là một ví dụ hay về việc bắt lỗi một phương thức mà chúng tôi không trông đợi. Chúng tôi không đặt expectation nào đối với auction, do đó lời gọi đến bất cứ phương thức nào của nó cũng làm cho test fail(). Nếu so sánh test này với bidsHigherAndReportsBiddingWhenNewPriceArrives() tại mục “AuctionSniper đặt Bid” (tr. 126), bạn đọc sẽ thấy rằng, tại `reportsIsWinningWhenCurrentPriceComesFromSniper`, chúng tôi không dùng các biến price và increment mà đẩy thẳng số vào lời gọi currentPrice(). Đó là vì tại test này không có việc tính toán gì, nên chúng tôi không cần phải nhắc đến các con số lần nữa trong expectation. Chúng chỉ là các tiểu tiết để chúng tôi setup được hành vi đáng quan tâm.

Việc sửa lỗi khá dễ dàng:

```
public class AuctionSniper implements AuctionEventListener { […]
  public void currentPrice(int price, int increment, PriceSource priceSource) {
    switch (priceSource) {
    case FromSniper:
      sniperListener.sniperWinning();
      break;
    case FromOtherBidder:
      auction.bid(price + increment);
      sniperListener.sniperBidding();
      break;
    }
  }
}
```

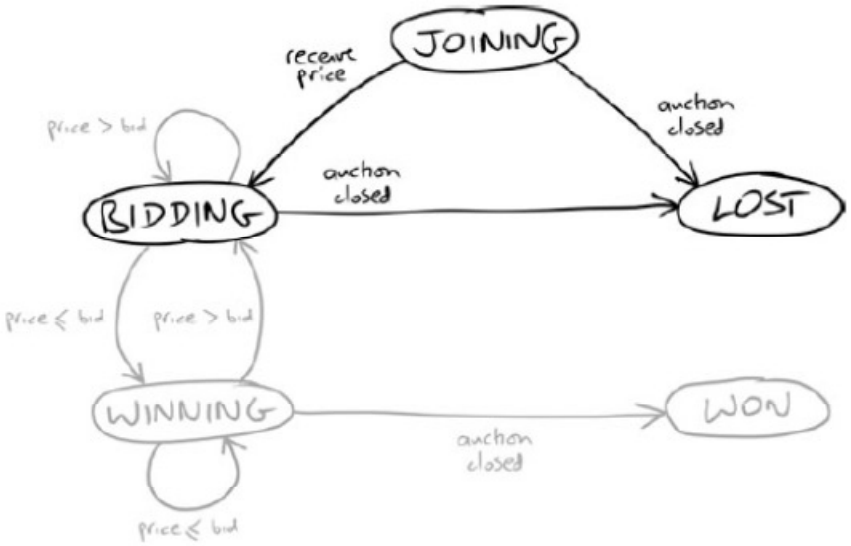
Chạy test end-to-end lần nữa cho thấy chúng tôi đã sửa xong lỗi test ở đầu chương này (lỗi hiển thị Bidding thay vì Winning). Giờ chúng tôi cần cho Sniper thắng:

```
java.lang.AssertionError:
Tried to look for...
  exactly 1 JLabel (with name "sniper status")
  in exactly 1 JFrame (with name "Auction Sniper Main" and showing on screen)
  in all top level windows
and check that its label text is "Won"
but...
  all top level windows
  contained 1 JFrame (with name "Auction Sniper Main" and showing on screen)
  contained 1 JLabel (with name "sniper status")
  label text was "Lost"
```

Sniper cần có trạng thái

Chúng tôi chuẩn bị thực hiện một thay đổi mạnh đối Sniper. Khi cuộc đấu giá kết thúc, chúng tôi muốn Sniper tuyên bố nó đã thắng hay thua, có nghĩa là nó phải biết tại thời điểm đó nó đang bidding hay winning. Điều đó hàm ý rằng Sniper sẽ phải giữ một trạng thái nào đó, cái mà từ đầu đến giờ nó chưa có.

Để có được chức năng đó, chúng tôi sẽ bắt đầu với các trường hợp đơn giản hơn: khi Sniper thua. Như trong hình 14.2, chúng tôi đang bắt đầu với việc chuyển trạng thái 1 bước, rồi 2 bước, trước khi bổ sung bước chuyển đưa Sniper tới trạng thái WON:



Hình 14.2. Sniper đặt bid rồi thua.

Đầu tiên, chúng tôi xem lại một unit test cũ và thêm một test mới. Các test này sẽ pass với cài đặt hiện tại; nhiệm vụ của chúng là đảm bảo rằng chúng tôi sẽ không phá vỡ hành vi vốn có khi chúng tôi bổ sung việc chuyển trạng thái.

Việc này dẫn đến việc sử dụng một cú pháp mới của JMock là *states*. Ý tưởng là chúng tôi có thể dùng nó để viết các assertion về trạng thái bên trong của đối tượng đang được test. Chúng tôi sẽ quay lại bàn tiếp về điểm này.

```
public class AuctionSniperTest { […]
  private final States sniperState = context.states("sniper"); (1)

  @Test
  public void reportsLostIfAuctionClosesImmediately() { (2)
    context.checking(new Expectations() {{
      atLeast(1).of(sniperListener).sniperLost();
    }});
    sniper.auctionClosed();
  }

  @Test
  public void reportsLostIfAuctionClosesWhenBidding() {
    context.checking(new Expectations() {{
      ignoring(auction); (3)
      allowing(sniperListener).sniperBidding();
                    then(sniperState.is("bidding")); (4)

      atLeast(1).of(sniperListener).sniperLost();
                    when(sniperState.is("bidding")); (5)
    }});

    sniper.currentPrice(123, 45, PriceSource.FromOtherBidder); (6)
    sniper.auctionClosed();
  }
}
```

(1) Chúng tôi muốn theo dõi trạng thái hiện tại của Sniper dựa theo các event mà nó gửi ra ngoài. Do đó, chúng tôi yêu cầu context cung cấp một đối tượng States để dành cho nhiệm vụ đó. Trạng thái mặc định mà nó lưu là null.

(2) Chúng tôi giữ test cũ, nhưng giờ nó chỉ áp dụng cho trường hợp không nhận được message Price nào.

(3) Sniper sẽ gọi auction, nhưng chúng tôi không quan tâm đến việc đó tại test này, nên chúng tôi bảo test lờ hẳn lớp cộng tác này đi

(4) Khi Sniper gửi một event đặt bid, nó bảo với chúng tôi rằng nó đang ở trạng thái bidding, và chúng tôi ghi lại trạng thái đó bằng mệnh đề *then*(...). Chúng tôi dùng mệnh đề *allowing*() để nói rằng đây là một hoạt động hỗ trợ của test, nó được phép xảy ra nhưng không phải là phần chúng tôi thực sự quan tâm và muốn kiểm tra.

(5) Đây là câu quan trọng, là expectation mà chúng tôi muốn khẳng định. Nếu Sniper không phải đang bidding nếu nó dùng lời gọi này, test sẽ fail.

(6) Đây là test đầu tiên mà chúng tôi cần một chuỗi event để đưa Sniper vào trạng thái cần test. Chúng tôi gọi các phương thức của nó theo thứ tự để thực hiện việc này.

Biểu diễn trạng thái đối tượng - Representing Object State

Trong những trường hợp như thế này, chúng tôi muốn assert về hành vi của một đối tượng theo trạng thái của nó, nhưng chúng tôi không muốn phá vỡ tính đóng gói khi chọc vào cách cài đặt trạng thái của đối tượng (để đối tượng cho phép truy nhập trạng thái từ ngoài). Thay vào đó, test có thể lắng nghe các các event mà Sniper gửi để thông báo về trạng thái của nó cho các đối tượng cộng tác. jMock cung cấp các đối tượng States để test có thể ghi lại và kiểm tra trạng thái của một đối tượng khi có gì đó quan trọng xảy ra, chẳng hạn như khi nó gọi phương thức của một đối tượng khác (xem cú pháp sử dụng tại Appendix A)

Đây là một biểu diễn "logic" của những gì đang xảy ra bên trong đối tượng, cụ thể ở đây là Sniper. Nó cho phép test mô tả những gì nó cho là quan trọng về Sniper, không cần biết Sniper thực ra được cài như thế nào. Như bạn đọc sẽ sớm nhận ra, sự phân tách này sẽ cho phép chúng tôi thực hiện các thay đổi mạnh mẽ đối với cài đặt của Sniper mà không phải sửa nội dung test.

Tên unit test reportsLostIfAuctionClosesWhenBidding rất gần với expectation mà nó áp đặt:

```
atLeast(1).of(sniperListener).sniperLost(); when(sniperState.is("bidding"));
```

Đó không phải sự tình cờ. Chúng tôi dành rất nhiều công sức để tìm các mệnh đề mà jMock cung cấp và phát triển một phong cách thể hiện được chủ ý cốt lõi của một unit test.

Sniper thắng cuộc

Cuối cùng, chúng tôi đã có thể nối vòng và cho Sniper thắng bid. Test tiếp theo bổ sung event Won.

```
@Test
public void reportsWonIfAuctionClosesWhenWinning() {
  context.checking(new Expectations() {{
    ignoring(auction);
    allowing(sniperListener).sniperWinning(); then(sniperState.is("winning"));
    atLeast(1).of(sniperListener).sniperWon(); when(sniperState.is("winning"));
  }});
  sniper.currentPrice(123, 45, true);
  sniper.auctionClosed();
}
```

Test này có cấu trúc giống test trước nhưng đại diện cho tình huống Sniper thắng cuộc. Test fail vì Sniper gọi sniperLost().

```
unexpected invocation: sniperListener.sniperLost()
expectations:
  allowed, never invoked: auction.<any method>(<any parameters>) was[];
  allowed, already invoked 1 time: sniperListener.sniperWinning();
                                   then sniper is winning
expected at least 1 time, never invoked: sniperListener.sniperWon();
                                         when sniper is winning
states:
  sniper is winning
what happened before this:
  sniperListener.sniperWinning()
```

Chúng tôi gắn một cờ để đại diện cho trạng thái của Sniper, và cài thêm phương thức sniperWon() cho SniperStateDisplayer.

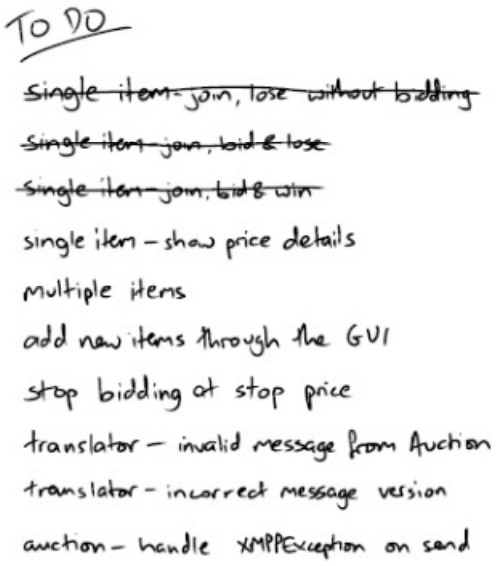
```
public class AuctionSniper implements AuctionEventListener { […]
  private boolean isWinning = false;

  public void auctionClosed() {
    if (isWinning) {
      sniperListener.sniperWon();
    } else {
      sniperListener.sniperLost();
    }
  }
  public void currentPrice(int price, int increment, PriceSource priceSource) {
    isWinning = priceSource == PriceSource.FromSniper;
    if (isWinning) {
      sniperListener.sniperWinning();
    } else {
      auction.bid(price + increment);
      sniperListener.sniperBidding();
    }
  }
}

public class SniperStateDisplayer implements SniperListener { […]
  public void sniperWon() {
    showStatus(MainWindow.STATUS_WON);
  }
}
```

Vừa mới cầu kì về PriceSource, có phải chúng tôi đang không nhất quán khi lại dùng boolean cho isWinning? Lý do của chúng tôi ở đây là chúng tôi đã thử dùng một cấu trúc enum cho trạng thái của Sniper, nhưng trông nó quá rối. isWinning là field private của AuctionSniper, nên nó đủ nhỏ để sau này có thể dễ dàng thay đổi, và code hiện giờ đã dễ đọc rồi.

Bây giờ tất cả các test unit cũng như end-to-end đều pass. Chúng tôi có thể gạch đi một mục nữa tại danh sách trong Hình 14.3.



Hình 14.3. Sniper thắng cuộc.

Còn nhiều test mà chúng tôi có thể viết, chẳng hạn test về kịch bản chuyển trạng thái từ bidding sang winning, rồi quay trở lại, nhưng chúng tôi sẽ dành việc này làm bài luyện tập cho bạn đọc. Thay vào đó, chúng tôi sẽ đi tiếp tới thay đổi quan trọng tiếp theo về chức năng.

Giữ cho tiến độ ổn định

Như thường lệ, chúng tôi đã giữ tiến độ ổn định với việc bổ sung từng lát mỏng các chức năng. Đầu tiên, chúng tôi làm cho Sniper thể hiện rằng nó đang winning, rồi đến thể hiện rằng nó đã thắng. Chúng tôi dùng các cài đặt rỗng để cho code dịch được, trong khi chúng tôi chưa sẵn sàng cài nội dung vì đang cần tập trung vào nhiệm vụ trước mắt.

Một trong những điều ngạc nhiên thú vị là giờ đây, khi code đã phát triển được thêm một chút, chúng tôi bắt đầu nhìn thấy thành quả của công sức ban đầu của mình, khi mà các tính năng mới khớp ngay vào cấu trúc sẵn có. Các nhiệm vụ tiếp theo cần phải cài sẽ dẫn đến các thay đổi đáng kể.