Chương 11: Pass test đầu tiên

Ở đây chúng tôi viết cơ sở hạ tầng để điều khiển ứng dụng Sniper còn chưa tồn tại, để có thể làm cho test đầu tiên fail. Chúng tôi liên tục chạy test fail và sửa chữa các triệu chứng, cho đến khi chúng tôi có một ứng dụng tối thiểu chạy được và pass được cái test đầu tiên đó. Chúng tôi sẽ đi thật chậm từng bước qua quá trình đó để trình bày xem quy trình đó chạy như thế nào.

Dựng test rig

Khởi đầu mỗi lần chạy test, test script của chúng tôi bật server Openfire, tạo các account cho Snipper và auction (cuộc đấu giá), rồi chạy test. Mỗi test sẽ tự bật ứng dụng và action rởm, rồi test sự liên lạc giữa chúng thông qua server. Đầu tiên, chúng tôi sẽ chạy tất cả mọi thứ tại cùng một máy. Về sau, khi cơ sở hạ tầng đã ổn định, chúng tôi có thể xem xét việc chạy các component khác nhau tại các máy khác nhau, khi đó sẽ gần hơn so với deployment thật.

Như vậy là chúng tôi còn hai component cần phải viết cho cơ sở hạ tầng test: ApplicationRunner và FakeAuctionServer.

Set up server Openfire

Tại thời điểm viết sách, chúng tôi dùng Openfire phiên bản 3.6. Đối với các test end-to-end, chúng tôi setup server nội bộ với ba user account và mật khẩu sau:

sniper : sniper

auction-item-54321 : auction

auction-item-65432 : auction

Để phục vụ phát triển tại máy desktop, chúng tôi thường bật server bằng tay và để nó chạy. Chúng tôi setup để nó không lưu các message offline, có nghĩa là không có trạng thái persistent. Tại System Manager, chúng tôi đặt tính chất "System Name" là localhost, để các test sẽ chạy một cách nhất quán. Cuối cùng, chúng tôi đặt resource policy là "Never kick," nó sẽ không cho phép một resource mới được log in nếu đang có xung đột..

Application Runner

Một đối tượng ApplicationRunner bao gói toàn bộ công việc quản lý và liên lạc với ứng dụng Swing mà chúng ta đang xây dựng. Nó chạy ứng dụng như thể chạy từ command line, nó nhận và giữ một tham chiếu tới cửa sổ chính của ứng dụng để truy vấn trạng thái của GUI và để tắt ứng dụng khi test chạy xong.

Chúng tôi không có nhiều việc phải làm ở đây vì có thể dùng WindowLicker cho các công việc khó khăn: tìm và điều khiển các GUI component, đồng bộ hóa với các thread và event queue của Swing, và gói toàn bộ những việc đó vào bên trong một API đơn giản. WindowLicker sử dụng khái niệm về một ComponentDriver: một đối tượng có thể điều khiển một feature trong một giao diện người dùng Swing. Nếu một ComponentDriver không thể tìm thấy component mà nó được nối với, nó sẽ time-out với trạng thái lỗi. Đối với test này, chúng tôi tìm một

component kiểu label chứa một chuỗi kí tự cho trước; nếu ứng dụng đang được test không tạo ra label đó, chúng tôi sẽ nhận được một exception. Dưới đây là cài đặt (các hằng được bỏ đi để dễ đọc) và một số giải thích:

```
public class ApplicationRunner {
  public static final String SNIPER_ID = "sniper";
  public static final String SNIPER_PASSWORD = "sniper";
  private AuctionSniperDriver driver;

  public void startBiddingIn(final FakeAuctionServer auction) {
    Thread thread = new Thread("Test Application") {
      @Override public void run() { (1)
        try {
          Main.main(XMPP_HOSTNAME, SNIPER_ID, SNIPER_PASSWORD, auction.getItemId()); (2)
        } catch (Exception e) {
          e.printStackTrace(); (3)
        }
      }
    };
    thread.setDaemon(true);
    thread.start();
    driver = new AuctionSniperDriver(1000); (4)
    driver.showsSniperStatus(STATUS_JOINING); (5)
  }

  public void showsSniperHasLostAuction() {
    driver.showsSniperStatus(STATUS_LOST);  (6)
  }

  public void stop() {
    if (driver != null) {
      driver.dispose(); (7)
    }
  }
}
```

1. Chúng tôi chạy ứng dụng từ hàm main() của nó để đảm bảo là chúng tôi đã lắp ghép các mảnh với nhau đúng cách. Chúng tôi đang dùng quy ước rằng cổng vào ứng dụng là lớp Main nằm tại package ở top-level. WindowLicker có thể điều khiển các component Swing đang ở trong cùng một máy ảo Java, vậy nên chúng tôi bật Sniper tại một thread mới. Một cách lí tưởng, test sẽ bật Sniper ở một tiến trình mới, nhưng làm vậy sẽ khó test hơn nhiều; nên chúng tôi nghĩ đây là một thỏa hiệp hợp lí.
2. Để giữ cho mọi thứ đơn giản tại giai đoạn này, chúng tôi giả thiết rằng chỉ bid một item và truyền id cho main().
3. Nếu main() ném exception, chúng tôi chỉ in nó ra. Test nào đang chạy cũng sẽ fail, và chúng tôi có thể xem stack trace tại output. Sau này, chúng tôi sẽ xử lý exception đúng kiểu.
4. Chúng tôi giảm thời gian time-out cho việc tìm frame và component. Các giá trị mặc định dài hơn cần thiết cho một ứng dụng đơn giản như cái chúng tôi đang test và sẽ làm các test chậm lại khi chúng fail. Chúng tôi dùng time-out 1 giây, đủ để cho độ trễ không đáng kể khi chạy test.
5. Chúng tôi đợi đổi trạng thái sang running, để chúng tôi biết rằng ứng dụng đã thực hiện kết nối. Lệnh assertion này nói rằng ở đâu đó tại UI có một label mô tả trạng thái của Sniper
6. Khi Sniper thua cuộc, chúng tôi trông đợi là nó sẽ hiển thị trạng thái thua. Nếu việc này không xảy ra, driver sẽ ném exception
7. Sau test, chúng tôi sẽ yêu cầu driver hủy bỏ cửa sổ để đảm bảo rằng nó sẽ bị thu hồi bộ nhớ thay vì được gọi cho test tiếp theo.

AuctionSniperDriver chỉ là một mở rộng của JFrameDriver của WindowLicker, được chuyên hóa cho test của chúng tôi:

```
public class AuctionSniperDriver extends JFrameDriver {
  public AuctionSniperDriver(int timeoutMillis) {
    super(new GesturePerformer(), JFrameDriver.topLevelFrame(named(Main.MAIN_WINDOW_NAME),
          showingOnScreen()), new AWTEventQueueProber(timeoutMillis, 100));
  }

  public void showsSniperStatus(String statusText) {
    new JLabelDriver(this, named(Main.SNIPER_STATUS_NAME)).hasText(equalTo(statusText));
  }
}
```

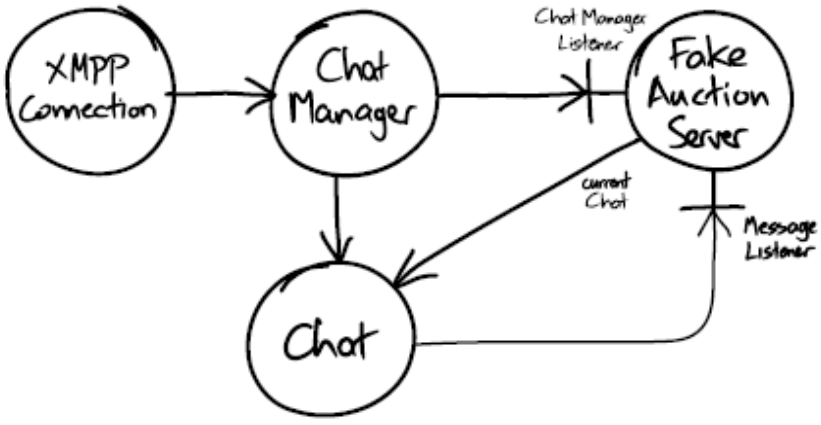
Constructor cố tìm một cửa sổ hiện ở top-level cho AuctionSniper trong khoảng thời gian đã cho. Phương thức showsSniperStatus() tìm label có liên quan tại UI và kiểm tra xem label đó có hiển thị trạng thái đã cho hay không. Nếu driver không thể tìm thấy đặc điểm mà nó trông đợi, nó sẽ ném một exception và làm cho test fail.

The Fake Auction

FakeAuctionServer là server thay thế, nó cho phép test kiểm tra được tương tác giữa AuctionSniper và một auction thông qua các message XMPP. Nó có ba trách nhiệm: (1) kết nối với XMPP broker và chấp nhận một yêu cầu tham gia chat được gửi từ Sniper; (2) nhận các message chat từ Sniper hoặc fail nếu không có message nào tới trong một khoảng time-out nào đó; và, (3) cho phép test gửi message trở lại cho Sniper như đã được đặc tả bởi hệ thống Southabee's On-Line.

Smack (thư viện client XMPP) là event-driven, do đó, auction rởm phải đăng kí các đối tượng listener để cho Smack call back. Các event thuộc về một trong hai mức: event *về* một chat, chẳng hạn người gia nhập chat, và event *bên trong* một chat, chẳng hạn message nhận được. Chúng tôi cần nghe cả hai loại event này.

Chúng tôi bắt đầu bằng việc cài phương thức startSellingItem(). Đầu tiên, nó nối với XMPP broker, dùng id của item để tạo login name; rồi nó đăng kí một ChatManagerListener. Smack sẽ gọi listener này kèm một đối tượng Chat đại diện cho session mà một Sniper nối đến. Auction giả bám giữ lấy đối tượng chat đó để có thể qua đó mà trao đổi message với Sniper đang nối đến.



Hình 11.1: Các đối tượng Smack và các lệnh gọi callback.

Ta có cài đặt như sau:

```
public class FakeAuctionServer {
  public static final String ITEM_ID_AS_LOGIN = "auction-%s";
  public static final String AUCTION_RESOURCE = "Auction";
  public static final String XMPP_HOSTNAME = "localhost";
  private static final String AUCTION_PASSWORD = "auction";
  private final String itemId;
  private final XMPPConnection connection;
  private Chat currentChat;

  public FakeAuctionServer(String itemId) {
    this.itemId = itemId;
    this.connection = new XMPPConnection(XMPP_HOSTNAME);
  }
  public void startSellingItem() throws XMPPException {
    connection.connect();
    connection.login(format(ITEM_ID_AS_LOGIN, itemId), AUCTION_PASSWORD, AUCTION_RESOURCE);
    connection.getChatManager().addChatListener( new ChatManagerListener() {
      public void chatCreated(Chat chat, boolean createdLocally) {
        currentChat = chat;
      }
    });
  }
  public String getItemId() {
    return itemId;
  }
}
```

Cài đặt rởm tối thiểu

Chúng tôi muốn nhấn mạnh lần nữa rằng đây là cài đặt tối thiểu chỉ để hỗ trợ việc chạy test. Ví dụ, chúng tôi dùng một biến thực thể (instance variable) duy nhất để móc tới đối tượng chat. Một server đấu giá thực sự sẽ cần quản lý nhiều chat để phục vụ nhiều bidder – nhưng đây là đồ rởm, nó chỉ dùng để chạy test, nên chỉ cần một đối tượng chat.

Tiếp theo, chúng tôi phải gắn thêm một MessageListener vào chat để nhận message từ Sniper. Điều đó có nghĩa cần phối hợp giữa thread chạy test và thread của Smack – nơi đẩy message vào listener: Test cần đợi message đến và time-out nếu nó không đến. Nên chúng tôi sẽ dùng một cái BlockingQueue gồm chỉ một phần tử (package java.util.concurrent). Vì chúng tôi chỉ có một chat trong test, chúng tôi chỉ phải xử lý mỗi lúc 1 message. Để làm rõ chủ ý của mình hơn, chúng tôi bọc queue vào trong một lớp helper SingleMessageListener. Đây là phần còn lại của FakeAuctionServer:

```
public class FakeAuctionServer {
  private final SingleMessageListener messageListener = new SingleMessageListener();
  public void startSellingItem() throws XMPPException {
    connection.connect();
    connection.login(format(ITEM_ID_AS_LOGIN, itemId), AUCTION_PASSWORD, AUCTION_RESOURCE);
    connection.getChatManager().addChatListener(new ChatManagerListener() {
      public void chatCreated(Chat chat, boolean createdLocally) {
        currentChat = chat;
        chat.addMessageListener(messageListener);
      }
    });
  }
  public void hasReceivedJoinRequestFromSniper() throws InterruptedException {
    messageListener.receivesAMessage(); (1)
  }
```

```
  public void announceClosed() throws XMPPException {
    currentChat.sendMessage(new Message()); (2)
  }
  public void stop() {
    connection.disconnect(); (3)
  }
}

public class SingleMessageListener implements MessageListener {
  private final ArrayBlockingQueue<Message> messages = new ArrayBlockingQueue<Message>(1);
  public void processMessage(Chat chat, Message message) {
    messages.add(message);
  }
  public void receivesAMessage() throws InterruptedException {
    assertThat("Message", messages.poll(5, SECONDS), is(notNullValue())); (4)
  }
}
```

1. Test cần biết khi một message Join đã đến. Ta chỉ kiểm tra xem đã có message nào đến hay chưa. Lí do là vì hiện tại Sniper chỉ gửi các message Join chứ không có loại nào khác; và ta sẽ bổ sung những thứ khác khi ta cho phần mềm phát triển tiếp. Cài đặt này sẽ fail nếu không có message nào đến trong vòng 5 giây.
2. Test cần có khả năng giả lập việc auction tuyên bố kết thúc, đó là lí do chúng tôi cần giữ đối tượng Chat hiện tại sau khi đã mở nó. Cũng như đối với yêu cầu Join, auction rởm cũng chỉ gửi một message rỗng, vì cho đến giờ thì chúng tôi mới chỉ hỗ trợ mỗi event này nên không cần thêm thông tin gì.
3. stop() đóng kết nối.
4. Biểu thức is(notNullValue()) sử dụng cú pháp Hamcrest matcher. Matcher được mô tả tại trang 339; ở thời điểm này, chỉ cần biết rằng nó kiểm tra xem Listener đã nhận được một message trong khoảng thời gian cho trước hay chưa

The Message Broker

Còn một component nữa không liên quan đến coding- việc cài đặt một XMPP message broker. Chúng tôi set up Openfire tại máy địa phương. Trong các test end-to-end của chúng tôi, Sniper và action rởm sẽ liên lạc với nhau qua server này, kể cả nếu chúng chạy trong cùng một tiến trình. Chúng tôi cũng setup các tài khoản login khớp với vài định danh item mà chúng tôi sẽ dùng trong các test.

Một sự thỏa hiệp chạy được – A Working Compromise

Như chúng tôi đã viết, chúng tôi đang gian lận một chút ở giai đoạn này để giữ cho quá trình phát triển tiếp tục tiến. Chúng tôi muốn các developer có được môi trường của mình để họ không can thiệp lẫn nhau trong khi chạy test. Có những đội phát triển đã tự gây nhiều khó khăn cho mình vì họ không muốn tạo một bản DB cho mỗi developer. Trong một tổ chức chuyên nghiệp, chúng tôi cũng trông đợi rằng ít nhất một test rig đại diện được cho môi trường thực, trong đó có sự phân tán xử lý tại nhiều phần trên mạng, và xây dựng một chy trình build dùng đến test rig đó để đảm bảo hệ thống chạy được

---

Nhìn test fail rồi cho test pass. - Failing and Passing the Test

Ta đã lắp ráp đủ cơ sở hạ tầng để chạy test và xem nó fail. Đối với phần còn lại của chương này, ta sẽ bổ sung chức năng, mỗi lần một lát mỏng, cho đến khi ta làm được cho test pass.

Chúng tôi bắt đầu bằng việc viết một script build cho ant. Mục tiêu là chỉ dùng 1 lệnh là đủ để compile, build, deploy, và test ứng dụng. Chúng tôi chỉ bắt đầu coding sau khi đã có một script tự động build và test.

Tại thời điểm này, chúng tôi sẽ mô tả từng bước, bàn về từng lần test fail. Về sau, chúng tôi sẽ tăng tốc

UI đầu tiên

Test fail.

Test không thể tìm thấy một UI component với tên "Auction Sniper Main".

```
java.lang.AssertionError:
Tried to look for...
  exactly 1 JFrame (with name "Auction Sniper Main" and showing on screen)
   in all top level windows
but...
  all top level windows
  contained 0 JFrame(with name "Auction Sniper Main" and showing on screen)
[…]
at auctionsniper.ApplicationRunner.stop()
at auctionsniper.AuctionSniperEndToEndTest.stopApplication()
[…]
```

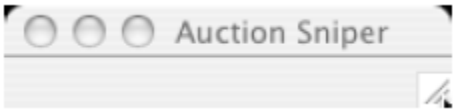
WindowLicker rất rõ ràng trong thông báo lỗi. Ở đây, ta không thể tìm thấy frame ở top-level, do đó JUnit fail ngay khi test còn chưa bắt đầu chạy.

Cài đặt - Implementation

Chúng tôi cần một cửa sổ top-level cho ứng dụng của mình. Chúng tôi viết một lớp MainWindow trong package auctionsniper.ui, lớp này là lớp con của JFrame, và gọi nó từ main(). Nó chỉ làm mỗi việc là tạo một cửa sổ với một cái tên đúng.

```
public class Main {
  private MainWindow ui;
  public Main() throws Exception {
    startUserInterface()
  }
  public static void main(String... args) throws Exception {
    Main main = new Main();
  }
  private void startUserInterface() throws Exception {
    SwingUtilities.invokeAndWait(new Runnable() {
      public void run() {
        ui = new MainWindow();
      }
    });
  }
}
public class MainWindow extends JFrame {
  public MainWindow() {
    super("Auction Sniper");
    setName(MAIN_WINDOW_NAME);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setVisible(true);
  }
}
```

Kết quả là UI trong hình 11.2. Đó là UI thực sự tối thiểu. Nhưng nó đủ để khẳng định rằng chúng tôi có thể bật một cửa sổ ứng dụng và kết nối với nó. Test của chúng tôi vẫn fail, nhưng chúng tôi đã tiến lên được một bước. Giờ chúng tôi biết rằng bộ yên cương (harness) đã dùng được, bớt được một việc cần lo khi chúng tôi tiến tới các chức năng thú vị hơn.



Hình 11.2 Chỉ là một cửa sổ top-level

Hiện trạng thái của Sniper

Test fail.

Test tìm thấy một cửa sổ top-level, nhưng nó không hiển thị trạng thái hiện tại của Sniper. Sniper cần cho thấy trạng thái Joining trong khi đợi action trả lời.

```
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 "Joining"
but...
  all top level windows
  contained 1 JFrame (with name "Auction Sniper Main" and showing on screen)
  contained 0 JLabel (with name "sniper status")
at com.objogate.wl.AWTEventQueueProber.check()
[…]
at AuctionSniperDriver.showsSniperStatus()
at ApplicationRunner.startBiddingIn()
at AuctionSniperEndToEndTest.sniperJoinsAuctionUntilAuctionCloses()
[…]
```

Cài đặt

Chúng tôi thêm một label thể hiện trạng thái của Sniper tại MainWindow.

```
public class MainWindow extends JFrame {
  public static final String SNIPER_STATUS_NAME = "sniper status";
  private final JLabel sniperStatus= createLabel(STATUS_JOINING);

  public MainWindow() {
    super("Auction Sniper");
    setame(MAIN_WINDOW_NAME);
    add(sniperStatus);
    pack();
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setVisible(true);
  }
  private static JLabel createLabel(String initialText) {
    JLabel result = new JLabel(initialText);
    result.setName(SNIPER_STATUS_NAME);
    result.setBorder(new LineBorder(Color.BLACK));
    return result;
  }
}
```

Ta sửa một chút, và giờ có thể hiển thị một chút nội dung tại ứng dụng của chúng tôi như trong Hình 11.3.

Hình 11.3: Hiển thị trang thái Joining.

Kết nối với Auction

Test fail.

UI của chúng tôi đã chạy, nhưng action chưa nhận được yêu cầu join từ Sniper.

```
java.lang.AssertionError:
  Expected: is not null
    got: null
at org.junit.Assert.assertThat()
at SingleMessageListener.receivesAMessage()
at FakeAuctionServer.hasReceivedJoinRequestFromSniper()
at AuctionSniperEndToEndTest.sniperJoinsAuctionUntilAuctionCloses()
[…]
```

Thông báo lỗi này hơi khó hiểu, nhưng những cái tên trong stack trace cho ta biết vấn đề là ở đâu.

Cài đặt

Chúng tôi viết một cài đặt đơn giản để pass được lỗi trên. Nó nối với chat trong Main và gửi một message rỗng. Chúng tôi tạo một MessageListener rỗng để cho phép tạo một đối tượng Chat dùng cho việc gửi message rỗng khởi đầu, vì chúng tôi chưa quan tâm đến việc nhận message.

```
public class Main {
  private static final int ARG_HOSTNAME = 0;
  private static final int ARG_USERNAME = 1;
  private static final int ARG_PASSWORD = 2;
  private static final int ARG_ITEM_ID = 3;
  public static final String AUCTION_RESOURCE = "Auction";
  public static final String ITEM_ID_AS_LOGIN = "auction-%s";
  public static final String AUCTION_ID_FORMAT = ITEM_ID_AS_LOGIN + "@%s/" + AUCTION_RESOURCE;
  […]
  public static void main(String... args) throws Exception {
    Main main = new Main();
    XMPPConnection connection = connectTo(args[ARG_HOSTNAME], args[ARG_USERNAME], args[ARG_PASSWORD]);
    Chat chat = connection.getChatManager().createChat(
          auctionId(args[ARG_ITEM_ID], connection),
          new MessageListener() {
            public void processMessage(Chat aChat, Message message) {
               // nothing yet
            }
          });
    chat.sendMessage(new Message());
  }
  private static XMPPConnection connectTo(String hostname, String username, String password)
              throws XMPPException
  {
    XMPPConnection connection = new XMPPConnection(hostname);
    connection.connect();
    connection.login(username, password, AUCTION_RESOURCE);
    return connection;
  }
  private static String auctionId(String itemId, XMPPConnection connection) {
    return String.format(AUCTION_ID_FORMAT, itemId,
    connection.getServiceName());
  }
  […]
}
```

Lưu ý:
Phần trên cho thấy chúng tôi có thể thiết lập một kết nối từ Sniper tới auction, có nghĩa chúng tôi phải quan tâm đến các chi tiết như giải nghĩa thông tin item và user từ các tham số dòng lệnh và sử dụng thư viện Smack. Chúng tôi đang tạm bỏ qua nội dung các message vì chúng tôi chỉ có một loại message, do đó gửi một giá trị rỗng là đủ để chứng tỏ là có kết nối.
Cài đặt này có vẻ ngây thơ vô cớ - dù sao thì chúng tôi cũng có thể thiết kế tử tế hơn, nhưng chúng tôi thường thấy rằng cứ viết một đoạn code bẩn và ngắn để xem nó có kết cục ra sao cũng đáng công. Nó giúp chúng tôi kiểm nghiệm các ý tưởng trước khi đi quá xa, và đôi khi kết quả có thể rất đáng ngạc nhiên. Điều quan trọng là phải đảm bảo rằng chúng tôi sẽ không để nó mãi ở tình trạng bẩn như vậy.
Chúng tôi lưu ý việc giữ cho code lập kết nối tách khỏi hàm Swing invokeAndWait() tạo MainWindow, vì chúng tôi muốn UI phải được ổn định trước khi chúng tôi thử điều gì đó phức tạp hơn.

Nhận response từ Auction

Test Failure

Với một kết nối đã được thiết lập, Sniper nên nhận và hiển thị thông báo thua từ auction. Nó chưa làm được việc này:

```
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 "Lost"
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 "Joining"
[…]
at AuctionSniperDriver.showsSniperStatus()
at ApplicationRunner.showsSniperHasLostAuction()
at AuctionSniperEndToEndTest.sniperJoinsAuctionUntilAuctionCloses()
[…]
```

Cài đặt

Chúng tôi cần gắn UI với chat, để nó có thể nhận được response từ auction. Do đó chúng tôi tạo một kết nối và truyền nó cho Main để tạo đối tượng Chat. joinAuction() tạo một MessageListerner có nhiệm vụ gán trị cho status label, dùng một lời gọi invokeLater() để tránh block thư viện Smack. Cũng như với message Join, chúng tôi không quan tâm đến nội dung của message đến, vì chỉ có mỗi một loại response mà auction có thể gửi tại thời điểm này. Trong khi làm việc đó, chúng tôi sửa tên connect() thành connection() để code dễ đọc hơn.

```
public class Main {
  @SuppressWarnings("unused") private Chat notToBeGCd;
  […]
  public static void main(String... args) throws Exception {
    Main main = new Main();
    main.joinAuction(connection(args[ARG_HOSTNAME], args[ARG_USERNAME], args[ARG_PASSWORD]), args[ARG_ITEM_ID]);
  }
  private void joinAuction(XMPPConnection connection, String itemId) throws XMPPException {
    final Chat chat = connection.getChatManager()
        .createChat(auctionId(itemId, connection), new MessageListener() {
            public void processMessage(Chat aChat, Message message) {
              SwingUtilities.invokeLater(new Runnable() {
                public void run() {
                  ui.showStatus(MainWindow.STATUS_LOST);
```

```
                    }
                  });
                }
        });
    this.notToBeGCd = chat;
    chat.sendMessage(new Message());
  }
```

Chat field để làm gì?

Bạn đọc sẽ thấy là chúng tôi đã gắn đối tượng Chat đã tạo vào field **notToBeGCd** (not to be garbage collected) trong **Main**. Làm vậy là để đảm bảo rằng đối tượng đó sẽ không bị dọn rác. Có một lời lưu ý ở đầu tài liệu ChatManager rằng: Chat manager theo dõi các tham chiếu tới tất cả các chat hiện hành. Nó sẽ không giữ tham chiếu nào trong bộ nhớ của riêng nó, do đó cần phải dữ một tham chiếu tới chính đối tượng chat.

Nếu chat bị thu dọn rác, Smack run-time sẽ giao message cho một đối tượng Chat mới, nó sẽ được tạo ra cho nhiệm vụ này. Trong một ứng dụng tương tác, ta sẽ nghe và hiển thị các chat mới này. Nhưng nhu cầu của chúng ta không phải như vậy, nên chúng tôi dùng tiểu xảo này để tránh xảy ra tình huống đó.

Chúng tôi cố ý làm cho tham chiếu này có vẻ vụng về - để làm nó nổi bật trong code lí do vì sao chúng tôi lại làm như vậy. Chúng tôi cũng biết rằng sau một thời gian chúng tôi có thể sẽ nghĩ ra một giải pháp tốt hơn.

Chúng tôi cài phương thức hiển thị tại UI, và cuối cùng thì test pass toàn bộ.

```
public class MainWindow extends JFrame {
  […]
  public void showStatus(String status) {
    sniperStatus.setText(status);
  }
}
```

Lưu ý

Hình 11.4 là khẳng định rằng đoạn code trên đã hoạt động.

Hình 11.4 Hiển thị trạng thái thua

Nó đã khẳng định rằng một Sniper có thể thiết lập một kết nối với một auction, nhận một response, và hiển thị kết quả

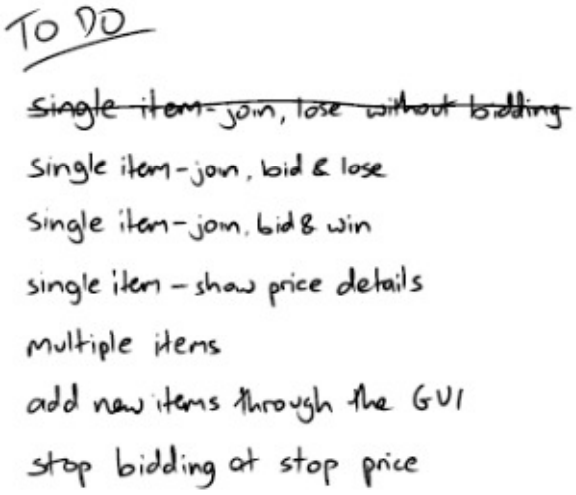
Yêu cầu tối thiểu - The Necessary Minimum

Tại một trong những bài thu hoạch phải nộp khi còn đi học, Steve đã được phê là "đã xác định tốt yêu cầu tối thiểu." Có vẻ như ông đã tìm thấy nơi thi triển khả năng của mình là nghề viết phần mềm, vì đó là kĩ năng tối quan trọng cho iteration zero.

Cái mà chúng tôi hy vọng bạn đọc thấy được ở chương này là mức độ tập trung cần phải có cho việc ghép nối lại bộ xương di động đầu tiên. Điểm quan trọng là thiết kế và thẩm định kiến trúc đầu tiên của hệ thống end-to-end – trong đó *end-to-end* bao gồm cả việc deploy tại một môi trường chạy được – để chứng tỏ rằng các lựa chọn của chúng tôi về thư viện, phần mềm, và tool thực sự chạy được. Sự sốt ruột sẽ giúp đội phát triển lọc bỏ chức năng xuống còn tối thiểu chỉ vừa đủ để kiểm chứng các giả thiết của mình. Đó là lí do vì sao chúng tôi không đưa nội dung gì vào trong các message của Sniper; nó sẽ là sự lạc hướng khỏi nhiệm vụ đảm bảo rằng liên lạc và xử lý event sẽ thông suốt. Chúng tôi đã không tốn quá nhiều công sức cho việc thiết kế chi tiết, phần vì không có nhiều việc, nhưng chủ yếu là vì chúng tôi mới chỉ đặt các mảnh ghép vào chỗ của nó; chúng tôi sẽ sớm cần nhiều công sức cho công việc này.

Tất nhiên, tất cả những gì bạn đọc thấy trong chương này chỉ là những điểm nổi bật đã được biên soạn. Chúng tôi đã bỏ qua nhiều chi tiết lệch trọng tâm và những cuộc thảo luận về việc nên dùng cái gì và làm thế nào để chạy chúng, những lúc rà soát các tài liệu kĩ thuật và danh mục thảo luận. Chúng tôi cũng đã bỏ qua một số thảo luận về mục tiêu của dự án này. Iteration zero thường nảy sinh những cuộc thảo luận khi đội phát triển xác định các tiêu chí định hướng cho các quyết định, do đó những người tài trợ cho dự án nên chuẩn bị trả lời những câu hỏi về mục tiêu xây dựng dự án.

Chúng tôi đã có được một cái gì đó nhìn thấy được để đánh dấu cho tiến độ phát triển, do dó chúng tôi có thể gạch đi mục đầu tiên trong danh sách của chúng tôi, như trong Hình 11.5



Hình 11.5 Mục đầu tiên đã hoàn thành

Bước tiếp theo là bắt đầu xây dựng chức năng thật.