Dependency Injection and Test Mocks

Michael Ballantyne and Zachary Morin

Adapted from "Big Modular Java with Guice" by Jesse Wilson and Dhanji Prasanna

Code for a twitter client

```
public void postButtonClicked() {
    String text = textField.getText();
    if (text.length() > 140) {
        Shortener shortener = new TinyUrlShortener();
        text = shortener.shorten(text);
    }
    if (text.length() <= 140) {
        Tweeter tweeter = new SmsTweeter();
        tweeter.send(text);
        textField.clear();
    }
}
```

How would we test this code?

Constructors called directly

```
public void postButtonClicked() {
    String text = textField.getText();
    if (text.length() > 140) {
Shortener shortener = new TinyUrlShortener();
        text = shortener.shorten(text);
    }
    if (text.length() <= 140) {
        Tweeter tweeter = new SmsTweeter();
        tweeter.send(text);
        textField.clear();
    }
}
```

We can't test with alternate Shorteners or Tweeters - they are constructed directly.

Dependencies from Factories

```
public void postButtonClicked() {
    String text = textField.getText();
    if (text.length() > 140) {
        Shortener shortener = ShortenerFactory.get();
        text = shortener.shorten(text);
    }
    if (text.length() <= 140) {
        Tweeter tweeter = TweeterFactory.get();
        tweeter.send(text);
        textField.clear();
    }
}
```

Factory Implementation

```
public class TweeterFactory {
   private static Tweeter testValue;
   public static Tweeter get() {
      if (testValue != null) {
         return testValue;
      }
      return new SmsTweeter();
   }
   public static void setForTesting(Tweeter
   tweeter) {
      testValue = tweeter;
   }
}
```

Testing with the factory

```
@Test
public void testSendTweet() {
    MockTweeter tweeter = new MockTweeter();
    TweeterFactory.setForTesting(tweeter);
    try {
        TweetClient tweetClient = new TweetClient();
        tweetClient.getEditor().setText("Hello!");
        tweetClient.postButtonClicked();
        assertEquals("Hello!", tweeter.getSent());
    } finally {
    TweeterFactory.setForTesting(null);
}
```

Must set up and tear down mock dependencies

Dependency Injection

```
public class TweetClient {
    private final Shortener shortener;
    private final Tweeter tweeter;
    public TweetClient(Shortener shortener, Tweeter
    tweeter)     {
        this.shortener = shortener;
        this.tweeter = tweeter;
    }
    public void postButtonClicked() {
    ...
    if (text.length() <= 140) {
        tweeter.send(text);
        textField.clear();
}
}
```

Rather than looking up dependencies, they are passed in

Testing with Dependency Injection

```
@Test
public void testSendTweet() {
    MockShortener shortener = new MockShortener();
    MockTweeter tweeter = new MockTweeter();
    TweetClient tweetClient = new TweetClient(shortener, tweeter);
    tweetClient.getEditor().setText("Hello!");
    tweetClient.postButtonClicked();
    assertEquals("Hello!", tweeter.getSent());
}
```

Easy to substitute mocks, no global state

But now clients need to build those dependencies...

```
public static void main(String[] args) {
    Shortener shortener = new TinyUrlShortener();
    Tweeter tweeter = new SmsTweeter();

    TweetClient client = new TweetClient(shortener,
tweeter);

    client.show();
}
```

Automatic Dependency Injection

```
public class TweetModule extends AbstractModule {
    protected void configure(){
        bind(Tweeter.class).to(SmsTweeter.class);
        bind(Shortener.class).to(TinyUrlShortener.class);
    }
}
public class TweetClient {
    private final Shortener shortener;
    private final Tweeter tweeter;
    @Inject
    public TweetClient(Shortener shortener, Tweeter
    tweeter)     {
        this.shortener = shortener;
        this.tweeter = tweeter;
    }
    ...
}
```

Writing a Test

```
public void postButtonClicked() {
    String text = textField.getText();
    if (text.length() > 140) {
        text = shortener.shorten(text);
    }
    if (text.length() <= 140) {
        tweeter.send(text);
        textField.clear();
    }
}
```

Manual Mocking

```
class ShortenerStub implements Shortener {
        private int timesCalled = 0;

        @Override
        public String shorten(String tweet) {
            timesCalled++;
            return "http://bitly.com/shortened";
        }

        public int getTimesCalled() {
            return timesCalled;
        }
    }
```

Why Mockito

Manual Mocks a lot of boilerplate (similar code).

Large API's take time and space to write.

Mockito is simple.

Mockito

Import

- download jar
- "import static org.mockito.Mockito.*"

Can automatically mock any interface or non-final class

Allows for easy stubbing and behaviour verification

Mockito cont'd

```
@Test
public void testClient(){
//setup mocks and stubs
    String testVal = "http://short.com"
    Shortener s = mock(Shortener.class);
    Tweeter t = mock(Tweeter.class);
    TweetClient client = new TweetClient(t, s);

    when(s.shorten(anyString())).thenReturn(testVal)
```

Mockito cont'd

```
    //run
    client.getEditor().setText("http://longURL.com/...");
    client.postButtonClicked();

    //verify
    assertEquals("http://short.com", client.getSent());

    //after the client.methodCalls()
    verify(s, times(1)).shorten(anyString());
    verify(t, times(1)).send(anyString());

}
```

Questions?

Exercise is posted.