SWE 637 Software Testing

Activities, week 4

Testing with test doubles

Dr. Brittany Johnson-Matthews

(Dr. B for short)

https://go.gmu.edu/SWE637

Adapted from slides by Jeff Offutt and Bob Kurtz

Class Activity #4

Test doubles

Consider testing a class for airline reservation services. It has a ranking service as a dependency, with the idea that higher ranking customers get better arrangements (better seats, early boarding, etc.). *We are not testing the ranking service*, and we should assume it's stochastic (that is, it might give different answers every time).

A simple example

An implementation of ReservationService might look like this:

```
public class ReservationService {
   // instance variables, constructors, other methods omitted for now

   public void reserve (Customer customer) {
      RankingService rankingService = new RankingService();
      // more code that uses the ranking service by calling its method
      //   public Rank getRank(Customer customer)
      // on the rankingService object.
    }
}
```

An associated test might look like this:

```
@Test public void testReservationService () {
   ReservationService reservationService = new ReservationService();
   RankingService fakeRankingService = new FakeRankingService(); // inherits from RankingService
   // Umm... How do I get RankingService.reserve() to use this test double?
   // some assertion about reservationService.reserve("John Smith");
   // some assertion about reservationService.reserve("Jane Doe");
}
```

Utilizing the Test Double

What if we make rankingService a member variable of ReservationService?

```
public class ReservationService {

   private RankingService rankingService;

   // instance variables, constructors, other methods omitted for now

   public void reserve (Customer customer) {
      rankingService = new RankingService();
      // more code that uses the ranking service by calling
      //   public Rank getRank(Customer customer)
      // on the rankingService object.
    }
}
```

What else do we need to do? Hint: think about **seams**

Develop one or more simple tests with FakeRankingService.
Focus on what you want to test, not the JUnit syntax

Utilizing Seams

Given what we know about seams, develop an approach to using a fake ranking service

1. Using a compiler seam
2. Using an inheritance seam without dependency injection
3. Using an inheritance seam with dependency injection
4. Using Jmockit (try this one if your group finishes your assigned approach)

1. Using a compiler seam

```
public class ReservationService {
   private boolean testMode = false;

   public ReservationService (boolean testMode) {
      this.testMode = testMode;
   }

  // instance variables, other methods omitted for now

   public void reserve (Customer customer) {
      RankingService rankingService;
      int rank;
      if (testMode) {
         rankingService = new RankingServiceFake();
      } else {
         rankingService = new RankingService();
      }
      rank = rankingService.getRank();
      // more code that uses the rank provided by the getRank() call
    }
}
```

Here we set 'test mode' in the constructor and execute different code if we're in 'test mode'.

```
@Test public void testReservationService () {
   ReservationService reservationService = new ReservationService(true);
   // some assertion about reservationService.reserve("John Smith");
   // some assertion about reservationService.reserve("Jane Doe");
}
```

2a. Without dependency injection

```
public class ReservationService {

   private RankingService rankingService;

   public ReservationService () {
      this.rankingService = RankingServiceFactory.getRankingService();
   }

   public void setTestMode () {
      this.rankingService = RankingServiceFactory.getRankingServiceFake();
   }

   // instance variables, other methods omitted for now

   public void reserve (Customer customer) {
      // more code that uses the ranking service by calling
      //   public Rank getRank(Customer customer)
      // on the rankingService object.
    }
}
```

Here we use a method to put the class into 'test mode' and get a fake ranking service using a RankingServiceFactory class.

```
@Test public void testReservationService () {
   ReservationService reservationService = new ReservationService();
   reservationService.setTestMode();  // enable unit test mode
   // some assertion about reservationService.reserve("John Smith");
   // some assertion about reservationService.reserve("Jane Doe");
}
```

What if a call to **setTestMode()** sneaks into delivered software?

2b. Without dependency injection

```
public class ReservationService {

   private RankingService rankingService;

   public ReservationService () {
      this.rankingService = RankingServiceFactory.getRankingServi
   }

   public ReservationService (RankingService rankingServiceFake) {
      this.rankingService = rankingServiceFake;
   }

   // instance variables, other methods omitted for now

   public void reserve (Customer customer) {
      // more code that uses the ranking service by calling
      //   public Rank getRank(Customer customer)
      // on the rankingService object.
    }
}
```

Here we use a 'test-mode' constructor to pass in the fake ranking service (which was externally-constructed).

```
@Test public void testReservationService () {
   RankingService rankingServiceFake = new RankingServiceFake(); // inherits from RankingService
   ReservationService reservationService = new ReservationService(rankingServiceFake);
   // some assertion about reservationService.reserve("John Smith");
   // some assertion about reservationService.reserve("Jane Doe");
}
```

3. With dependency injection

```
public class ReservationService {

   private RankingService rankingService;

   // instance variables, constructors, other methods omitt

   public void setRankingService (RankingService rankingService) {
      this.rankingService = rankingService;
   }

   public void reserve (Customer customer) {
      // more code that uses the ranking service by calling
      // public Rank getRank(Customer customer)
      // on the rankingService object.
    }
}
```

Here we use a setter method to always pass in the ranking service (real OR fake); or we could use a single constructor.

```
@Test public void testReservationService () {
   ReservationService reservationService = new ReservationService();
   RankingService rankingServiceFake = new RankingServiceFake(); // inherits from RankingService
   reservationService.setRankingService(rankingServiceFake);
   // some assertion about reservationService.reserve("John Smith");
   // some assertion about reservationService.reserve("Jane Doe");
}
```

This affects **ReservationService** users and breaks encapsulation

4. Using JMockit

```
public class ReservationService {

   private RankingService rankingService;

   // instance variables, constructors, other methods omitted for now

   public void reserve (Customer customer) {
      rankingService = RankingServiceFactory.getRankingService();
      // more code that uses the ranking service by calling
      //   public Rank getRank(Customer customer)
      // on the rankingService object.
   }
}
```

```
@Mocked RankingService rankingServiceMock;

@Test public void testReservationService () {
   new Expectations() {
      RankingService.getRank(new Customer("John Smith");
      returns(5); // John Smith has rank=5

      RankingService.getRank(new Customer("Jane Doe");
      returns(2); // Jane Doe has rank=2
   }

   ReservationService reservationService = new ReservationService();
   // some assertion about reservationService.reserve("John Smith");
   // some assertion about reservationService.reserve("Jane Doe");
}
```

"Expect the **getRank()** method to be called with 'John Smith' as the argument, and when that happens, then return 5."