Assignment 4

Rasika Mohod
rmohod@gmu.edu
G01044774

Story explaining Test Doubles:

Test Double is a generic term used in a scenario where tester replaces a software component in production by a testing component (test double) for testing purposes.

Let me narrate you one of my interesting programming experiences where I employed test doubles for testing convenience.

In one of my interviews, I was asked to test a software which was used to make reservations for customers using a Reservation Service based on the customers's ranks. The higher the customer's rank, better reservation arrangements would be made. The rank of customer was determined by a special Ranking Service which would score customer and then determine his rank.

The main functionality of the software to be tested was to verify if software successfully make correct reservations for customers. Now, as the Reservation Service was using a special third party Ranking Service to determine rank of the customer and using this rank to reserve that customer, it was difficult and expensive in time and cost to test this third party application. As a solution to this expensive method, I devised and implemented an inexpensive way to test reservation service by creating a new Fake Ranking Service which would determine customer' rank in some simplest way. This rank determined by Fake Ranking Service could then be used to test if reservations are correctly made for customers or not.

The Fake Ranking Service used here was actually the "Test Double" implemented in the software during the testing phase for testing convenience without modifying the production software i.e. real Ranking Service.

However, to implement this test double I needed some entry point into the Reservation Service where I could set the ranking service to be used as Fake Ranking Service for my testing purposes. I enabled this entry point by implementing setter method in Reservation Service with ranking service as argument.

This implementation of Fake Ranking Service and setter method in Reservation Service allowed my JUnit Test to set the ranking service to be used as Fake Ranking Service and assert the correct behavior of the Reservation Service.

Thus, this way you can simply use “Test Doubles” for your testing conveniences.

In-Class Exercise 3.

Suppose we pull the `rankingService` variable out as an instance variable in the `ReservationService` class. This a step in the right direction. Why? (Your answer should say something about "seams"). What does the reservation service code look like now? What does the test code look like now?

In order to create a test double for RankingServices as FakeRankingServices, we need to break the dependency of the system from the RankingServices which is our test double component. Thus, we need a variable which can change the program behavior without modifying the actual program component. This variable, termed as seam can be obtained in our code by pulling out the `rankingService` variable and using it as seam.

New ReservationServices.java code:

```
package com.gmu.rmohod;

public class ReservationService {

	private RankingServices rankingService; // This is seam

	public void reserve( Customer customer) {
	   rankingService = RankingServices.getRankingservice();

	   Rank rank = rankingService.getRank(customer);

	   System.out.println("Rank is:" + rank.rank);
	}
}
```

New TestReservationServices.java code:

```
package com.gmu.rmohod;

import static org.junit.Assert.*;

import org.junit.Test;

public class TestReservationService {
```

```
	@Test public void testReservationService() {
		ReservationService reservationService = new ReservationService();

		Rank rank1 = new Rank(1);
		Customer customer = new Customer(rank1);
		reservationService.reserve(customer);

		//RankingServices fakeRankingService = new FakeRankingServices();
	}

}
```

What else do we need to do in the `ReservationService` class? Hint: think about "enabling points".

We need provide an enabling point for seam which can be done by including setter method in ReservationServices class. This “enabling point” will enable testers to set test suitable rankingService value.

```
package com.gmu.rmohod;

public class ReservationService {

	private RankingServices rankingService; // This is seam

	public void reserve( Customer customer) {
	   rankingService = RankingServices.getRankingservice();

	   Rank rank = rankingService.getRank(customer);

	   System.out.println("Rank is:" + rank.rank);
	}

	public RankingServices getRankingService() {
		return rankingService;
	}

	void setRankingService(RankingServices rankingService) {
		this.rankingService = rankingService;
	} // This is enabling point
}
```

How does this help in the test code? Hint: think about "exploiting enabling points".

We can exploit the enabling point from test code by simply calling setter with our test double as an argument. This will help the testers to run test code conveniently using test suitable values for ranking service.

```
package com.gmu.rmohod;

import static org.junit.Assert.*;

import org.junit.Test;
```

```
public class TestReservationService {

	@Test public void testReservationService() {
		ReservationService reservationService = new ReservationService();
		RankingServices fakeRankingService = new FakeRankingServices();

		Rank rank1 = new Rank(1);
		Customer customer = new Customer(rank1);

		fakeRankingService.setRankingservice(fakeRankingService);//
		Exploited and Injected with test double dependency

		Rank rank = fakeRankingService.getRank(customer);
		int actual = rank.rank;
		assertEquals(1,actual);

	}

}
```

JUnit Test run result:

Finally, I need some assertions. What can I say about the fake ranking service that I can't say about the real ranking service? Vice versa? Hint: think about "state testing" vs. "interaction testing". Hint: Recall that you have total control over the fake ranking service.

Interaction based testing can weaken the assertion as Fake ranking service can only verify if the ranking service was used by reservation service to reserve the table. But it cannot verify if the ranking service behaved correctly to return correct rank for reserving table for customer as the ranking service used was not real but fake.

Also, to perform such interaction testing the setter method should be given required access level such that tester can access it and exploit its use for testing.

On the other hand real ranking service can verify the real software behavior which might be missed by testing while using fake ranking service in certain scenarios.