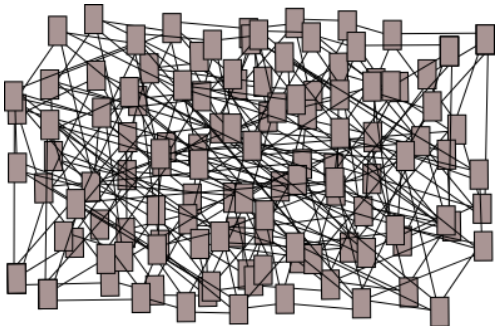
Advanced JUnit

CS585 Software Verification and Validation
http://cs585.yusun.io
January 14, 2015

Yu Sun, Ph.D.
http://yusun.io
yusun@cpp.edu

Testing Complex Structure is Challenging

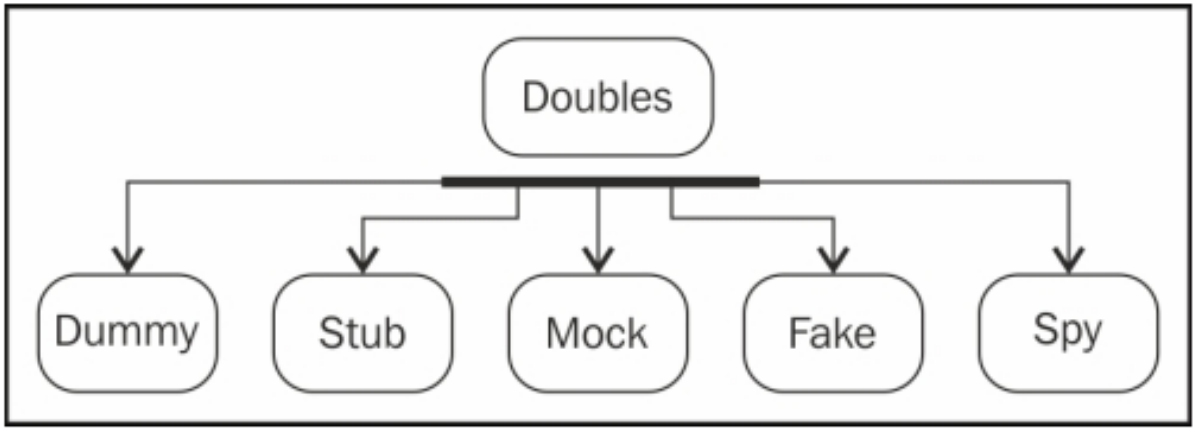
- Complex object interactions
- Complex 3rd party libraries
- Complex system environments



Use Test Doubles

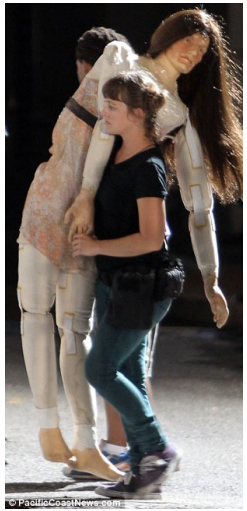
Use Test Doubles

- Test doubles act as stunt doubles
- They look and behave like their release-intended counterparts, but are actually simplified versions that reduce the complexity and facilitate testing



Dummy

- Dummy objects are often passed to avoid NullPointerException for mandatory parameter objects



Dummy

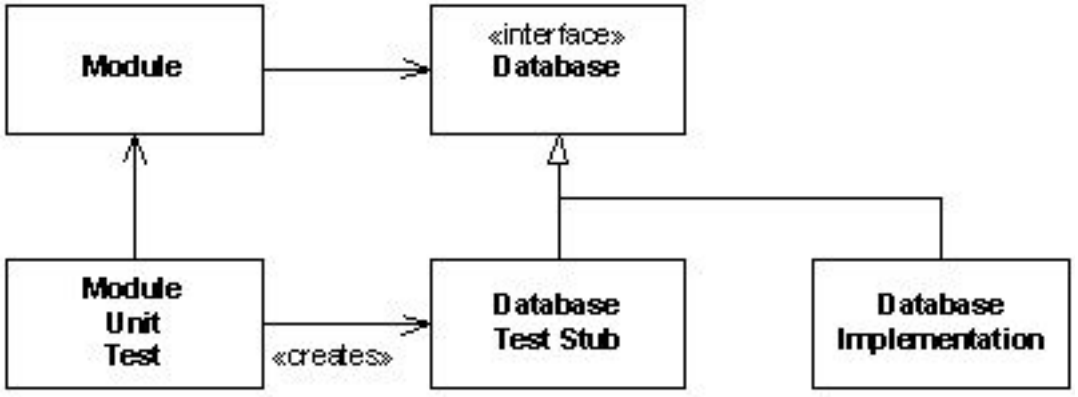
```
Book javaBook = new Book("Java 101", "123456");
Member dummyMember = new DummyMember();
javaBook.issueTo(dummyMember);

assertEquals(javaBook.numberOfTimesIssued(),1);
```

- It is often used in a hierarchy

Stub

- A stub delivers indirect inputs (pre-programmed results) to the caller when the stub's methods are called
- Stubs are programmed only for the test scope



Stub

```
public interface Dispenser {
  void dispense(BigDecimal amount) throws DispenserFailed;
}

Public class DefaultDispenser {
  public void dispense(BigDecimal amount) throws DispenserFailed {
    // real database connection
    // ...
  }
}

public class AlwaysFailingDispenserStub implements Dispenser{
  public void dispense(BigDecimal amount) throws DispenserFailed {
    throw new DispenserFailed (ErrorType.HARDWARE,"not  responding");
  }
}
```

Stub

```
class ATMTest...
  @Test
  public void transaction_is_rolledback_when_hardware_fails() {
    Account myAccount = new Account("John", 2000.00);
    TransactionManager txMgr =
        TransactionManager.forAccount(myAccount);
    txMgr.registerMoneyDispenser(new AlwaysFailingDispenserStub());
    WithdrawalResponse response = txMgr.withdraw(500.00);
    assertEquals(false, response.wasSuccess());
    assertEquals(2000.00, myAccount.remainingAmount());
  }
```

Fake

- Fake objects are working implementations
- The fake class extends the original class, but it usually hacks the performance, which makes it unsuitable for production

Fake

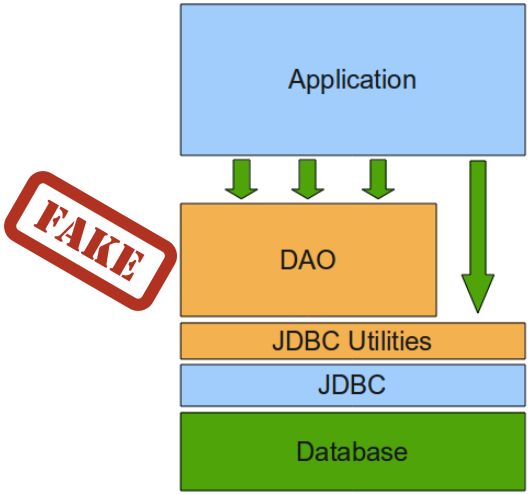
```
public class AddressDao extends SimpleJdbcDaoSupport{

  public SimpleJdbcTemplate getSimpleJdbcTemplate() {
    // returns the JDBC template to connect to the database
    return jdbcTemplate;
  }

  public void batchInsertOrUpdate(List<AddressDTO> addressList, User user){
    List<AddressDTO> insertList = buildListWhereLastChangeTimeMissing(addressList);
    List<AddressDTO> updateList = buildListWhereLastChangeTimeValued(addressList);
    int rowCount =  0;
    rowCount = getSimpleJdbcTemplate().batchUpdate(INSERT_SQL,…);
    rowCount += getSimpleJdbcTemplate().batchUpdate(UPDATE_SQL,…);
    if (addressList.size() != rowCount){
      raiseErrorForDataInconsistency(…);
    }
}
```

Fake

```
public class FakeAddressDao extends AddressDao{
  @Override
  public SimpleJdbcTemplate getSimpleJdbcTemplate() {
    return myMockedObject;
  }
}
```



Spy

- Spy is a variation of a mock/stub, but instead of only setting expectations, spy records the calls made to the collaborator

Spy

```
class ResourceAdapter {
  void print(String userId, String document, Object settings) {
    if (securityService.canAccess("lanPrinter1", userId)) {
      printer.print(document, settings);
    }
  }
}
```

```
class SpyPrinter implements Printer{
  private int noOfTimescalled = 0;
   @Override
   public void print(Object document, Object settings) {
     noOfTimescalled++;
   }
   public int getInvocationCount() {
     return noOfTimescalled;
   }
}
```

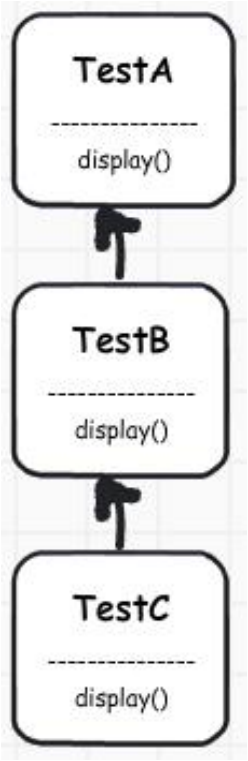
Spy

```
class FakeSecurityService implements SecurityService{
  public boolean canAccess(String printerName, String userId){
    return true;
  }
}
```

```
@Test public void verify() throws Exception {
  SpyPrinter spyPrinter = new SpyPrinter();
  adapter = new ResourceAdapter(new FakeSecurityService(), spyPrinter);
  adapter.print("john", "helloWorld.txt", "all pages");
  assertEquals(1, spyPrinter.getInvocationCount());
}
```

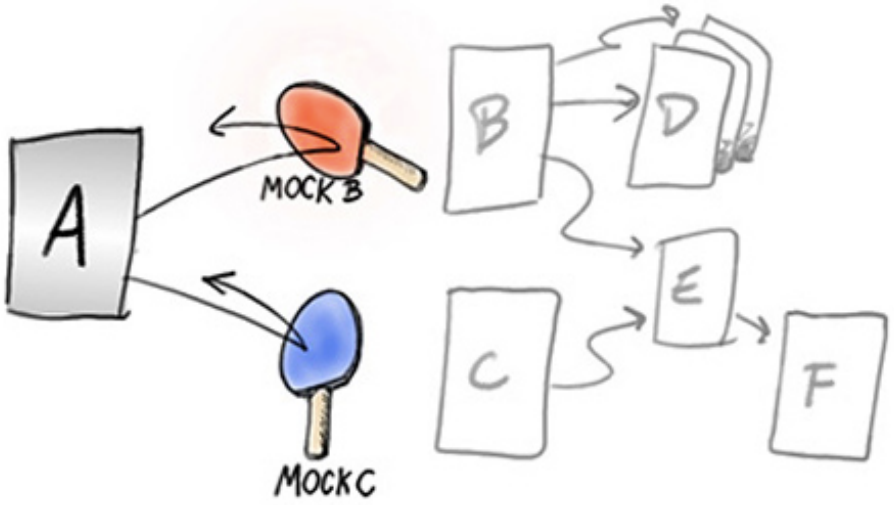
Keys to Test Doubles

- Use inheritance/override
- Expose the instance reference
- Ignore/Simplify/Simulate the test



Mock

- Mock objects have expectations
- Test expects a value from a mock object, and during execution, a mock object returns the expected result



Mock

```
public class ATMTest {
  @Mock Dispenser failingDispenser;

  @Before  public void setUp() throws Exception {
    MockitoAnnotations.initMocks(this);
  }

  @Test  public void transaction_is_rolledback_when_hardware_fails() throws
    DispenserFailed {
    Account myAccount = new Account(2000.00, "John");
    TransactionManager txMgr = TransactionManager.forAccount(myAccount);
    txMgr.registerMoneyDispenser(failingDispenser);

    doThrow(new DispenserFailed())
          .when(failingDispenser).dispense(isA(BigDecimal.class));
    txMgr.withdraw(500);
    assertEquals(2000.00, myAccount.getRemainingBalance());
    verify(failingDispenser, new Times(1)).dispense(isA(BigDecimal.class));
  }
}
```