# TDD Lab Exercises: Spy as a Test Double

This document contains two practice exercises for Test-Driven Development (TDD) using the Spy test double pattern in Java.

## Exercise 1: MessageBroadcaster Spy Example

Objective:

Implement a MessageBroadcaster class that sends a message to multiple recipients using a MessageSender. Students will use both Mockito’s spy() and a manual spy to verify interaction behavior.

TDD Steps:

1. Write a failing test that verifies the MessageSender.send() method is called for each recipient.  
2. Create minimal implementation to make the test compile (RED → GREEN).  
3. Refactor to clean up names and verify arguments.  
4. Extend with a manual spy implementation that records sent messages.

### Code Template

Production Code (src/main/java/com/example/MessageBroadcaster.java):

package com.example;  
  
public class MessageBroadcaster {  
 private final MessageSender sender;  
  
 public MessageBroadcaster(MessageSender sender) {  
 this.sender = sender;  
 }  
  
 public void broadcast(String message, String... recipients) {  
 for (String r : recipients) {  
 sender.send(message, r);  
 }  
 }  
}  
  
class MessageSender {  
 public void send(String message, String recipient) {  
 System.out.printf("Sending '%s' to %s%n", message, recipient);  
 }  
}  
  
class MessageSenderSpy extends MessageSender {  
 private int sendCount = 0;  
 private final java.util.List<String> sentMessages = new java.util.ArrayList<>();  
  
 @Override  
 public void send(String message, String recipient) {  
 sendCount++;  
 sentMessages.add(message + " -> " + recipient);  
 }  
  
 public int getSendCount() { return sendCount; }  
 public java.util.List<String> getSentMessages() { return sentMessages; }  
}

Test Code (src/test/java/com/example/MessageBroadcasterTest.java):

package com.example;  
  
import org.junit.jupiter.api.Test;  
import static org.junit.jupiter.api.Assertions.\*;  
import static org.mockito.Mockito.\*;  
  
class MessageBroadcasterTest {  
  
 @Test  
 void sendsMessagesToAllRecipients() {  
 MessageSender realSender = new MessageSender();  
 MessageSender spySender = spy(realSender);  
  
 MessageBroadcaster broadcaster = new MessageBroadcaster(spySender);  
 broadcaster.broadcast("Hello", "alice", "bob", "charlie");  
  
 verify(spySender, times(3)).send(anyString(), anyString());  
 verify(spySender).send("Hello", "alice");  
 verify(spySender).send("Hello", "bob");  
 verify(spySender).send("Hello", "charlie");  
 }  
  
 @Test  
 void logsSentMessagesWithManualSpy() {  
 MessageSenderSpy spy = new MessageSenderSpy();  
 MessageBroadcaster broadcaster = new MessageBroadcaster(spy);  
  
 broadcaster.broadcast("Hi", "john", "jane");  
  
 assertEquals(2, spy.getSendCount());  
 assertTrue(spy.getSentMessages().contains("Hi -> john"));  
 assertTrue(spy.getSentMessages().contains("Hi -> jane"));  
 }  
}

## Exercise 2: Bank Notification System (Real-World Spy Example)

Objective:

Implement a TransactionService that processes a transaction and triggers a NotificationService. Use a spy to verify that notifications are sent after each successful transaction.

TDD Steps:

1. Write a failing test verifying that notifyUser() is called after saving a transaction.  
2. Create the minimal implementation for TransactionService and its dependencies.  
3. Use Mockito’s spy() for NotificationService to verify the interaction.  
4. Optionally, implement a manual spy to confirm recorded notifications.

### Code Template

Production Code (src/main/java/com/example/TransactionService.java):

package com.example;  
  
public class TransactionService {  
 private final NotificationService notificationService;  
 private final TransactionRepository repository;  
  
 public TransactionService(NotificationService notificationService, TransactionRepository repository) {  
 this.notificationService = notificationService;  
 this.repository = repository;  
 }  
  
 public void processTransaction(Transaction txn) {  
 repository.save(txn);  
 notificationService.notifyUser(txn.getUserId(), "Transaction successful");  
 }  
}  
  
class NotificationService {  
 public void notifyUser(String userId, String message) {  
 System.out.println("Notifying user " + userId + ": " + message);  
 }  
}  
  
interface TransactionRepository {  
 void save(Transaction txn);  
}  
  
class Transaction {  
 private final String userId;  
 private final double amount;  
  
 public Transaction(String userId, double amount) {  
 this.userId = userId;  
 this.amount = amount;  
 }  
  
 public String getUserId() { return userId; }  
 public double getAmount() { return amount; }  
}

Test Code (src/test/java/com/example/TransactionServiceTest.java):

package com.example;  
  
import org.junit.jupiter.api.Test;  
import static org.mockito.Mockito.\*;  
  
class TransactionServiceTest {  
  
 @Test  
 void sendsNotificationAfterTransactionSaved() {  
 NotificationService realNotification = new NotificationService();  
 NotificationService spyNotification = spy(realNotification);  
  
 TransactionRepository mockRepo = mock(TransactionRepository.class);  
 TransactionService service = new TransactionService(spyNotification, mockRepo);  
  
 Transaction txn = new Transaction("u123", 500);  
  
 service.processTransaction(txn);  
  
 verify(mockRepo).save(txn);  
 verify(spyNotification).notifyUser("u123", "Transaction successful");  
 }  
}