Зачем нужен Mockito?

Как проверять результат при выполнении теста?

Единая Фронтальная Система

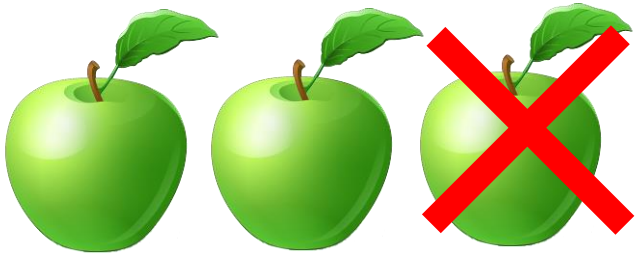
Существует 2 способа проверки результатов теста:

- проверка состояния;
- проверка поведения.

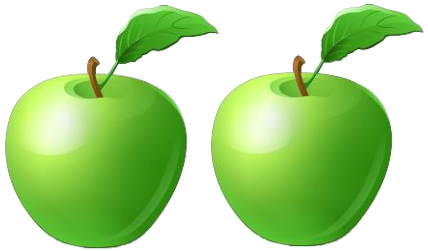
Что такое проверка состояния?

Было:

Сделали:



Ожидаем:



Интерфейс BankAccount

Единая Фронтальная Система

```
public interface BankAccount {
  BigDecimal getBalance();

  void deposit(BigDecimal amount);

  void withdraw(BigDecimal amount)
          throws InsufficientFundsException;
}
```

Класс PaymentBankAccount

Единая Фронтальная Система

```
public class PaymentBankAccount implements BankAccount {
  private BigDecimal balance = ZERO;

  @Override
  public BigDecimal getBalance() {  return balance;  }

  @Override
  public void deposit(BigDecimal amount) {
    balance = balance.add(amount);
  }

  @Override
  public void withdraw(BigDecimal amount)
          throws InsufficientFundsException {
    if (balance.compareTo(amount) >= 0) {
      balance = balance.subtract(amount);
    } else {
      throw new InsufficientFundsException();
    }
  }
}
```

Класс Order

Единая Фронтальная Система

```
public class Order {
  private final List<Item> items = new ArrayList<>();
  private final List<Item> itemsView = unmodifiableList(items);

  public List<Item> getItems() {
    return itemsView;
  }

  public void buyItem(Item item, BankAccount account)
          throws InsufficientFundsException {
    account.withdraw(item.getPrice());
    items.add(item);
  }
}
```

Пример: тест с проверкой состояния

Единая Фронтальная Система

```
class OrderTest {
  private BankAccount account = new PaymentBankAccount();
  private Order order = new Order();

  @BeforeEach
  void setUp() {
    account.deposit(valueOf(100));
  }

  @Test
  void testSucceededIfEnoughFunds() throws InsufficientFundsE… {
    Item cake = new Item("Cake", valueOf(70));
    order.buyItem(cake, account);
    assertAll(
      () -> assertEquals(valueOf(30), account.getBalance()
              , "Баланс уменьшился на сумму покупки"),
      () -> assertTrue(order.getItems().contains(cake)
              , "Товар добавлен в список приобретённых")
    );
  }
  …
}
```

Пример: тест с проверкой состояния

Единая Фронтальная Система

```
class OrderTest {
  private BankAccount account = new PaymentBankAccount();
  private Order order = new Order();

  @BeforeEach
  void setUp() {
    account.deposit(valueOf(100));
  }
  …
  @Test
  void testFailedIfInsufficientFunds() {
    Item car = new Item("Car", valueOf(1_000_000));
    assertAll(
      ()-> assertThrows(InsufficientFundsException.class,
            ()-> order.buyItem(car, account), "…"),
      ()-> assertAll("Состояние объектов не изменилось",
            ()-> assertEquals(valueOf(100), account.getBalance()),
            ()-> assertTrue(order.getItems().isEmpty())
          )
    );
  }
}
```

Пример: тест с проверкой поведения

Единая Фронтальная Система

```
class OrderTest {
  private BankAccount account = mock(BankAccount.class);
  private Order order = new Order();

  @Test
  void testSucceedIfEnoughFunds() throws InsufficientFundsE… {
    Item cake = new Item("Cake", valueOf(70));

    order.buyItem(cake, account);

    assertAll(
      () -> verify(account).withdraw(cake.getPrice()),
      () -> assertTrue(order.getItems().contains(cake),
            "Товар добавлен в список приобретённых")
    );
  }
  …
}
```

Пример: тест с проверкой поведения

Единая Фронтальная Система

Argument(s) are different! Wanted:
bankAccount.withdraw(**70**);
-> at r.s.t.a.m.OrderTest.lambda$testSucceedIfEnoughFunds$0(OrderTest.java:90)
Actual invocation has different arguments:
bankAccount.withdraw(**100**);
-> at r.s.t.a.m.Order.buyItem(Order.java:47)

Пример: тест с проверкой поведения

Единая Фронтальная Система

Wanted but not invoked:
bankAccount.withdraw(70);
-> at r.s.t.a.m.OrderTest.lambda$testSucceedIfEnoughFunds$0(OrderTest.java:90)
Actually, there were **zero** interactions with this mock.

Пример: тест с проверкой поведения

Единая Фронтальная Система

```
class OrderTest {
  private BankAccount account = mock(BankAccount.class);
  private Order order = new Order();
  …
  @Test
  void testFailIfInsufficientFunds() throws InsufficientFundsE… {
    Item car = new Item("Car", valueOf(1_000_000));
    doThrow(InsufficientFundsException.class)
      .when(account).withdraw(car.getPrice());

    assertAll(
      () -> assertThrows(InsufficientFundsException.class,
              () -> order.buyItem(car, account), "…"),
      () -> assertAll("Состояние объектов не изменилось",
              () -> verify(account).withdraw(car.getPrice()),
              () -> assertTrue(order.getItems().isEmpty())
            )
    );
  }
}
```

Mock

Единая Фронтальная Система

Mock – объекты с заранее запрограммированными ожиданиями вызовов, что формирует спецификацию взаимодействия.
Используются для проверки поведения объектов.

Dummy

Единая Фронтальная Система

Dummy – объект, который передаётся, но в действительности никогда не используется.
Обычно используется, чтобы заполнить список аргументов метода.

Dummy

Единая Фронтальная Система

```
public interface NotificationService {
  void notifyOrderShipped(Order order);
}

public class Order {
  private final NotificationService notificationService;

  public Order(NotificationService notificationService) {
    this.notificationService = notificationService;
  }
  …
}
```

Dummy

Единая Фронтальная Система

```
class OrderTest {
  private Order order;
  …
  @BeforeEach
  void setUp() {
    …
    order = new Order(new NotificationServiceDummy());
  }
  …
}

class NotificationServiceDummy implements NotificationService {
  @Override
  public void notifyOrderShipped(ClientOrder order) {
    throw new RuntimeException("Should not be called!");
  }
}
```

Stub

Единая Фронтальная Система

Stub – заглушки, предоставляющие заранее запрограммированные для теста ответы/действия заданных методов.

Stub

Единая Фронтальная Система

```
public interface PromotionService {
  List<Item> getGiftsByItem(Item item);
}

public class Order {
  private final PromotionService promotionService;

  public Order(PromotionService promotionService) {
    this.promotionService = promotionService;
  }
  …
  public void buyItem(Item item, BankAccount account)
          throws InsufficientFundsException {
    account.withdraw(item.getPrice());
    items.add(item);
    List<Item> gifts = promotionService.getGiftsByItem(item);
    items.addAll(gifts);
  }
}
```

Stub

Единая Фронтальная Система

```
class OrderTest {
  private Order order = new Order(new PromotionServiceStub());
  …
}

class PromotionServiceStub implements PromotionService {
  @Override
  public List<Item> getGiftsByItem(Item item) {
    return emptyList();
  }
}
```

Spy

Единая Фронтальная Система

Spy – объект, который записывает некоторую информацию о том, как вызывались его методы.

Spy


```
class OrderTest {
  private PromotionServiceSpy spy = new PromotionServiceSpy();
  private Order order = new Order(spy);

  @Test
  void testSucceedIfEnoughFunds()
          throws InsufficientFundsException {
    order.buyItem(cake, account);
    …
    assertTrue(spy.getGiftsByItemCalled
        , "Метод getGiftsByItem вызван");
  }
}

class PromotionServiceSpy implements PromotionService {
  boolean getGiftsByItemCalled = false;

  @Override
  public List<Item> getGiftsByItem(Item item) {
    getGiftsByItemCalled = true;
    …
  }
}
```

Почему именно mockito?

Единая Фронтальная Система

- Имеет простое API, позволяет писать чистые и понятные тесты.
- Не нужен самописный код.
- Огромное сообщество.