Mockito

Shristi Technology Labs

Contents

- Mockito
- What is mocking?
- Adding mockito to a project
- Creating mock object with @Mock
- Injecting mock objects with @InjectMocks
- Verify the behavior using verify()
- Verify the call order
- Wrapping the java objects using @spy or spy()

Overview of Mockito

- Mockito is an open source testing framework for java.
- Mockito allows the creation of mock objects for unit testing of classes with external dependencies.
- A Mock object is a proxy for actual implementations.
- Static methods, private methods and constructors cannot be mocked.

Mocking

What is mocking?

- Mocking is technique to test the functionality of a class in isolation
- Mock objects are used to mock the real service .
- Pass mock input to check the functionality of the method

Example:

- Need to test the methods of OrderDetails class
- OrderDetails class has dependency on BookService.
- Mock the BookService and get mock data to work with the test class

How to use Mockito

- Add the dependencies into the class under test using mock objects
- Test the class
- Validate the result with the expected output

Environment Setup

- Use Maven
- Add dependencies
  - mockito-core
  - mockito-junit-jupiter
  - junit-jupiter-api
  - junit-jupiter-engine
  - junit-platform-runner

```
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-api</artifactId>
    <version>5.7.0</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-engine</artifactId>
    <version>5.7.0</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.junit.platform</groupId>
    <artifactId>junit-platform-runner</artifactId>
    <version>1.7.0</version>
    <!-- <scope>test</scope> -->
</dependency>
<dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-core</artifactId>
    <version>3.11.2</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-junit-jupiter</artifactId>
    <version>3.11.2</version>
    <scope>test</scope>
</dependency>
```

Creating and injecting mock objects

Creating Mock objects

- Create Mock Objects using **@Mock** Annotation

Injecting mock objects

- **@InjectMocks** annotation is used to inject the mock object into the class under test using constructor or setter methods

Run the test class

- Use these annotations above the class to run the testcases

@ExtendWith(MockitoExtension.class)

@RunWith(JUnitPlatform.class)

Example

```
public interface BookService {

    List<Book> getAllBooks();
    Double getDiscountedPrice(Book book);
}
```

Class for Mocking

```
public class OrderDetails {
    BookService bookService;
    public void setBookService(BookService bookService) {
        this.bookService = bookService;
    }
    public double getTotalOrder(Book book, int quantity) {
        double price = bookService.getDiscountedPrice(book);
        return  price * quantity;
    }
}
```

Class under Test

Example

```
@ExtendWith(MockitoExtension.class)
@RunWith(JUnitPlatform.class)
class OrderTests {
    @Mock // creating a mock/proxy for bookservice
    IBookService bookService;

    @InjectMocks // equivalent to details = new OrderDetails();
    OrderDetails details;

    List<Book> bookList;
    Book book1, book2, book3, book4, book5, book6;

    @BeforeEach
    void setUp() throws Exception {

        // injecting the mock by calling the setter method
        details.setBookService(bookService);
        book1 = new Book("Java", "Kathy", 1900, 1);
        book2 = new Book("Spring", "Kathy", 800, 2);
        book3 = new Book("Java", "Joe", 900, 3);
        book4 = new Book("Leadership", "Kathy", 1800, 4);
        book5 = new Book("Java", "Kathy", 500, 5);
        book6 = new Book("Miracle", "Kathy", 800, 6);
        bookList = Arrays.asList(book1, book2, book3, book4, book5, book6);
    }
```

Static methods of Mockito

- Mocks can return different values based on arguments passed into a method
  - `when thenReturn`
  - `when thenThrow`
  - `doReturn when`
  - `doThrow when` **- to mock void methods**
- The return values from the method call for the mock object can be configured
- Unspecified method calls returns empty values as
  - null for objects
  - 0 for numbers
  - false for boolean
  - empty collections for collections

Example – BookDetails(class under test)

```
public class BookDetails {
    BookService bookService;

    public void setBookService(BookService bookService) {
        this.bookService = bookService;
    }

    public List<Book> getBooksOnSale() {
        return bookService.getAllBooks().stream().filter((book) -> book.getPrice() < 1000).collect(Collectors.toList());
    }

    public List<Book> getBooksByAuthor(String author) {
        List<Book> bookList = new ArrayList<>();
        try {
            bookList = bookService.getBooksByAuthor(author);
        } catch (BookNotFoundException e) {
            System.out.println(e);
        }
        return bookList.stream().sorted((book1, book2) -> book1.getTitle().compareTo(book2.getTitle()))
                .collect(Collectors.toList());
    }

    public String updateBook(String title, double price) {
        try {
            bookService.updateBook(title, price);
            return "book updated";
        } catch (BookNotFoundException e) {
        }
        ;
        return "book not updated";
    }
}
```

Example – BookService(class to be mocked)

```
public interface BookService {

    List<Book> getAllBooks();
    Double getDiscountedPrice(Book book);
    List<Book> getBooksByAuthor(String author) throws BookNotFoundException;
    void updateBook(String title, double price)throws BookNotFoundException;
}
```

Example – BookTest (setup test data)

```
@Mock
BookService bookService; // creating mock object
// inject mock object
@InjectMocks
BookDetails bookDetails;
List<Book> bookList;
@Before
public void setup(){
    bookDetails = new BookDetails();
    bookDetails.setBookService(bookService);
    bookList = Arrays.asList(
            new Book("Java","Kathy",400),
            new Book("Spring","Johnson",1900),
            new Book("Servlets","Kathy",600),
            new Book("Node","Arav",500),
            new Book("Angular","John",1000),
            new Book("Javascript","John",900)
            );
}
```

Example – when thenReturn

```
@Test
public void testgetAllBooks() {
    when(bookService.getAllBooks()).thenReturn(bookList);
    List<Book> allbooks = bookDetails.getBooksOnSale();
    assertEquals(allbooks.size(), 4);
}
```

```
@Test
public void testPosBooksByAuthor() {
    String author = "Kathy";
    try {
        when(bookService.getBooksByAuthor(author))
           .thenReturn(Arrays.asList(book1, book2, book4));
        List<Book> booksByAuthor = bookDetails.getBooksByAuthor(author);
        assertEquals(booksByAuthor.size(), 3);
    } catch (BookNotFoundException e) {
        System.out.println(e);
    }
}
```

Example – when thenThrow

```
@Test
public void testgetBooksByAuthor() {
    String author = "Wilson";
    try {
        when(bookService.getBooksByAuthor(author))
            .thenThrow(new BookNotFoundException());
        List<Book> allbooks = bookDetails.getBooksByAuthor(author);
        //assertEquals(allbooks.size(), 0);
        assertTrue(allbooks.isEmpty());
    } catch (BookNotFoundException e) {
        System.out.println(e);
    }
}
```

Handling Exceptions for method with return type

when().thenThrow()

Example – doReturn when

```
@Test
public void testBooklistByAuthor() throws BookNotFoundException {
    String author = "Kathy";
        doReturn(Arrays.asList(book1, book2, book4))
          .when(bookService.getBooksByAuthor(author));
        List<Book> booksByAuthor = bookDetails.getBooksByAuthor(author);
        assertEquals(booksByAuthor.size(), 3);
}
```

```
@Test
public void testallBooks(){
        doReturn(null)
          .when(bookService).getAllBooks();
        List<Book> booksList = bookDetails.getBooksOnSale();
        assertNull(booksList);
}
```

Example – doThrow when

```
@Test
public void testUpdateBook() throws BookNotFoundException {
    String title = "JSP";
    double price =890;
        doThrow(new BookNotFoundException("Title not found"))
          .when(bookService).updateBook(title, price);
        String result = bookDetails.updateBook(title, price);
        assertEquals("book not updated", result);
}
```

Handling Exceptions for method without return type

doThrow().when().method

Example – doNothing when

```
@Test
@DisplayName("add to cart")
void testAddToCart() throws BookNotFoundException {
    // to check void methods will not return anything
    doNothing().when(cartService).addtoCart(book1);
    // calling method of testing class
    assertEquals("added successfully", details.addToCart(book1));
}
```

For methods without return type

doNothing().when().method

Verify the behavior – Mockito.verify()

Mockito.verify()

- is used on the mock object to verify that a method has been called with right parameters.

- This does not check the result of a method call, but it checks if a method is called with the right parameters.

Methods used with verify

- To check
  - Whether the method was called on the mock object
  - how many times the method is called on the mock object
  - The minimum number of times the method is called
  - The maximum number of times the method is called

```
verify(T mock, Verification mode))
verify(mockobject, never())
verify(mockobject, atleastOnce())
verify(mockobject, atleast(2))
verify(mockobject, atmost(3))
verify(mockobject, times())
```

Example

```
@Test
public void testVerify(){
    bookService.getDiscountedPrice(book1);
    // check if method was called with matching parameter
    verify(bookService).getDiscountedPrice(ArgumentMatchers.eq(book1));
    when(bookService.getDiscountedPrice(book1)).thenReturn(500.0);
    // check how many times the method gets invoked
    verify(bookService, times(2)).getDiscountedPrice(book1);
    verify(bookService,atLeastOnce()).getDiscountedPrice(book1);
    verify(bookService, never()).getAllBooks();
    //will give verification error as the method is called twice
    verify(bookService,never()).getDiscountedPrice(book1);
}
```

Verify the call order – Mockito.inOrder

Mockito.inOrder()

- This specifies the order in which the methods are called.
- Call verify() methods on the InOrder object to enforce order.

```
@Test
public void testOrder()throws Exception{
    bookService.getBooksByAuthor("Kathy");
    bookService.getDiscountedPrice(book3);
    bookService.getAllBooks();

    InOrder order = inOrder(bookService);
    order.verify(bookService,atLeastOnce()).getBooksByAuthor("Kathy");
    order.verify(bookService,atLeastOnce()).getDiscountedPrice(book3);
    order.verify(bookService,times(1)).getAllBooks();
}
```

Wrapping Java objects with Spy

- @Spy or the spy() method is used to wrap a real object.
- Mockito creates a new instance for the field annotated with @Spy
- It can call the original methods on the real instance

```
@Spy
@InjectMocks
BookDetails details ;// Mockito creates the instance
@Test
public void testSpy(){
    when(bookService.getAllBooks()).thenReturn(bookList);
    List<Book> allbooks = details.getBooksOnSale();
    assertEquals(allbooks.size(), 5);

    }
```

Mock Vs Spy

Mock

- is to mock the original object
- call a method on that object and return a result

Spy

- Is to spy the original object
- To do partial mock
- All the methods of the original object can be called

Example

Using Spy

```
@Spy
BookDetails sdetails;// Mockito creates the instance
@Test
public void testSpy() {
    // method called on real object
    String message = sdetails.greetMessage("Priya");
    assertEquals("welcome Priya",message);
}
```

```
@Mock
BookDetails mdetails;// creating a mock
@Test
public void testMock() {
    //make stubbing call on mock
    when(mdetails.greetMessage("Priya"))
        .thenReturn("welcome Priya");
    // this will return null as object not created
    //use when thenReturn to get result on stubbing call
    String message = mdetails.greetMessage("Priya");
    assertEquals("welcome Priya",message);
}
```

Using Mock

Summary

- Overview of Mockito
- Creating mock object with @Mock
- Injecting mock objects with @InjectMocks
- Verify the behavior
- Handling Exceptions
- Wrapping the java objects using @spy or spy()

Thank You