Mocking Dependencies in Spring Tests using Mockito

Exercise 1: Mocking a Service Dependency in a Controller Test

Task: Write a unit test for a Spring controller that uses a service to fetch data. Mock the service dependency using Mockito.

Step-by-Step Solution:

1. **Create the User Entity:**

```
@Entity
public class User {
   @Id
   private Long id;
   private String name;
   // getters and setters
}
```

2. **Create the UserService:**

```
@Service
public class UserService {
   @Autowired
   private UserRepository userRepository;
   public User getUserById(Long id) {
      return userRepository.findById(id).orElse(null);
   }
}
```

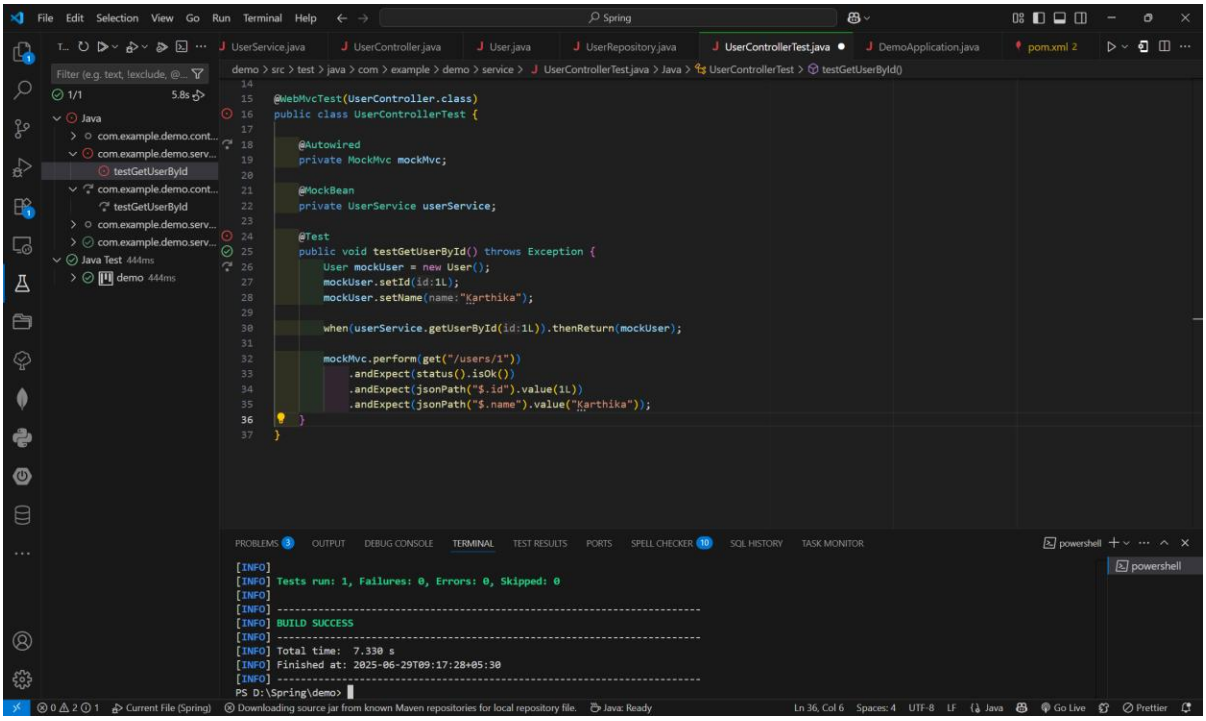
3. **Create the UserController:**

```
@RestController
```

```
@RequestMapping("/users")
public class UserController {
    @Autowired
    private UserService userService;
    @GetMapping("/{id}")
    public ResponseEntity<User> getUser(@PathVariable Long id) {
        return ResponseEntity.ok(userService.getUserById(id));
    }
}
```

4. **Create the UserControllerTest:**

```
@WebMvcTest(UserController.class)
public class UserControllerTest {

    @Autowired
    private MockMvc mockMvc;

    @MockBean
    private UserService userService;

    @Test
    public void testGetUserById() throws Exception {
        User mockUser = new User();
        mockUser.setId(1L);
        mockUser.setName("Karthika");

        when(userService.getUserById(1L)).thenReturn(mockUser);

        mockMvc.perform(get("/users/1"))
            .andExpect(status().isOk())
            .andExpect(jsonPath("$.id").value(1L))
            .andExpect(jsonPath("$.name").value("Karthika"));
    }
}
```



Exercise 2: Mocking a Repository in a Service Test

Task: Write a unit test for a Spring service that uses a repository to fetch data. Mock the repository dependency using Mockito.

Step-by-Step Solution:

1. **Create the User Entity:**

```
@Entity
public class User {
  @Id
  private Long id;
  private String name;
  // getters and setters
}
```

2. **Create the UserRepository:**

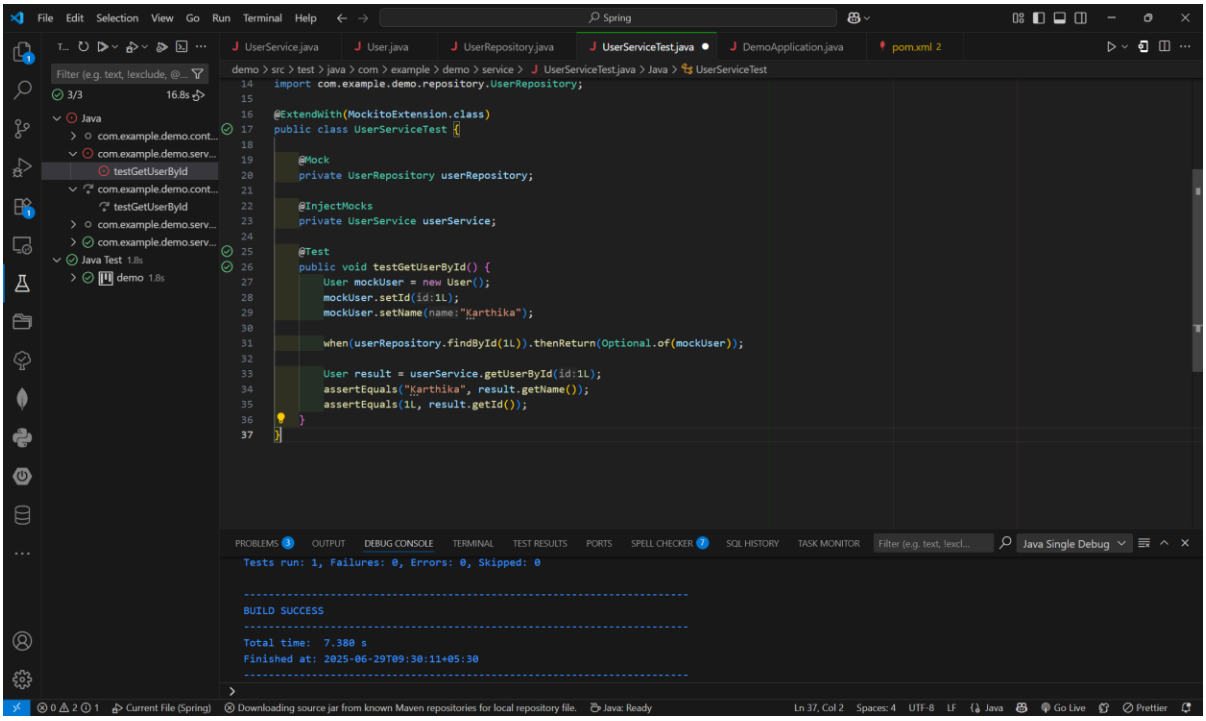
```
public interface UserRepository extends JpaRepository<User, Long> {
}
```

3. **Create the UserService:**

```
@Service
public class UserService {
    @Autowired
    private UserRepository userRepository;
    public User getUserById(Long id) {
        return userRepository.findById(id).orElse(null);
    }
}
```

4. **Create the UserServiceTest:**

```
@ExtendWith(MockitoExtension.class)
public class UserServiceTest {

    @Mock
    private UserRepository userRepository;

    @InjectMocks
    private UserService userService;

    @Test
    public void testGetUserById() {
        User mockUser = new User();
        mockUser.setId(1L);
        mockUser.setName("Karthika");

        when(userRepository.findById(1L)).thenReturn(Optional.of(mockUser));

        User result = userService.getUserById(1L);
        assertEquals("Karthika", result.getName());
        assertEquals(1L, result.getId());
    }
}
```



Exercise 3: Mocking a Service Dependency in an Integration Test

Task: Write an integration test for a Spring Boot application that mocks a service dependency using Mockito.

Step-by-Step Solution:

1. **Create the User Entity:**

```
@Entity
public class User {
   @Id
   private Long id;
   private String name;
   // getters and setters
}
```

2. **Create the UserService:**

```
@Service
public class UserService {
```

```
    @Autowired
    private UserRepository userRepository;
    public User getUserById(Long id) {
        return userRepository.findById(id).orElse(null);
    }
}
```

3. **Create the UserController:**

```
@RestController
@RequestMapping("/users")
public class UserController {
    @Autowired
    private UserService userService;
    @GetMapping("/{id}")
    public ResponseEntity<User> getUser(@PathVariable Long id) {
        return ResponseEntity.ok(userService.getUserById(id));
    }
}
```

4. **Create the UserIntegrationTest:**

```java
@SpringBootTest
@AutoConfigureMockMvc
public class UserIntegrationTest {

    @Autowired
    private MockMvc mockMvc;

    @MockBean
    private UserService userService;

    @BeforeEach
    void setUp() {
        User mockUser = new User();
        mockUser.setId(1L);
        mockUser.setName("John");
        Mockito.when(userService.getUserById(1L)).thenReturn(mockUser);
    }

    @Test
    void testGetUser() throws Exception {
        mockMvc.perform(get("/users/1"))
            .andExpect(status().isOk())
            .andExpect(jsonPath("$.id").value(1L))
            .andExpect(jsonPath("$.name").value("John"));
    }
}
```