Testing Spring Boot Applications Demystified

Lab 1: Spring Boot Testing Basics

About Me

- [Your introduction here]
- Contact: [Your contact info]
- GitHub: [Your GitHub info]

Workshop Agenda

- Slot 1: Spring Boot Testing Basics (105 min)
- Slot 2: Sliced Testing (115 min)
- Slot 3: Integration Testing (90 min)
- Slot 4: Best Practices & Pitfalls (70 min)

Workshop Repository

- GitHub: [URL]
- Materials: 4 lab projects
- Each lab focuses on different testing techniques
- Domain: Library Management System

Development Environment Setup

- Java 21
- IDE (IntelliJ, VS Code, Eclipse)
- Maven
- Git

Maven Basics

```xml
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

Maven Lifecycle

1. **validate**: validate project structure
2. **compile**: compile source code
3. **test**: run tests
4. **package**: package compiled code
5. **verify**: run integration tests
6. **install**: install in local repository
7. **deploy**: deploy to remote repository

Spring Boot Testing Basics

- Built on JUnit 5 (Jupiter)
- Mockito integration
- Spring Test Context Framework
- Auto-configuration for tests
- Sliced testing support

JUnit 5 & Mockito - The Foundation

```java
@Test
void testBookService() {
    // Given
    Book book = new Book("123", "Test Book", "Test Author");
    when(bookRepository.findById("123")).thenReturn(Optional.of(book));

    // When
    Optional<Book> result = bookService.getBookById("123");

    // Then
    assertTrue(result.isPresent());
    assertEquals("Test Book", result.get().getTitle());
    verify(bookRepository).findById("123");
}
```

Design for Testability

Prefer Constructor Injection

```java
// Hard to test
public class BookService {
    private final BookRepository bookRepository = new BookRepositoryImpl();
}

// Testable
public class BookService {
    private final BookRepository bookRepository;

    public BookService(BookRepository bookRepository) {
        this.bookRepository = bookRepository;
    }
}
```

Avoiding Static and Direct Instantiation

```java
// Hard to test
public LocalDate getDueDate() {
    return LocalDate.now().plusDays(14);
}

// Testable
public LocalDate getDueDate(Clock clock) {
    return LocalDate.now(clock).plusDays(14);
}
```

Don't Overuse Mockito

- Mocks can make tests brittle
- Consider using real objects when:
    - They have no external dependencies
    - They're simple value objects
    - They're collections or other standard library classes

JUnit 5 Extensions

```
@ExtendWith(MockitoExtension.class)
class BookServiceTest {
    @Mock
    private BookRepository bookRepository;

    @InjectMocks
    private BookService bookService;
}
```

Exercise: Write a JUnit Jupiter Extension

Create a custom extension for timing tests:

```java
public class TimingExtension implements BeforeTestExecutionCallback,
                                        AfterTestExecutionCallback {
    private static final Logger logger = LoggerFactory.getLogger(TimingExtension.class);

    @Override
    public void beforeTestExecution(ExtensionContext context) {
        getStore(context).put("start", System.currentTimeMillis());
    }

    @Override
    public void afterTestExecution(ExtensionContext context) {
        long start = getStore(context).remove("start", Long.class);
        long duration = System.currentTimeMillis() - start;
        logger.info("Test {} took {} ms", context.getDisplayName(), duration);
    }

    private Store getStore(ExtensionContext context) {
        return context.getStore(Namespace.create(getClass(), context.getRequiredTestMethod()));
    }
}
```

Refactoring Time-Based Code

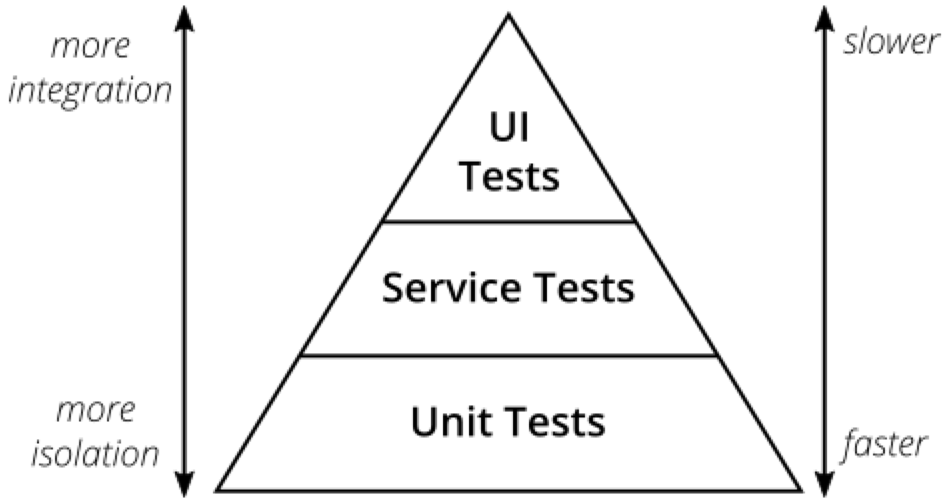
Before:

```java
public boolean isOverdue(BookLoan loan) {
    return LocalDate.now().isAfter(loan.getDueDate());
}
```

After:

```java
public boolean isOverdue(BookLoan loan, Clock clock) {
    return LocalDate.now(clock).isAfter(loan.getDueDate());
}
```

Spring's Testing Pyramid



- Unit Tests: Fast, focused, isolated
- Integration Tests: Verify components work together
- End-to-End Tests: Full application testing

Lab 1: Exercises

1. Write unit tests for the Book entity
2. Create a custom JUnit 5 extension
3. Refactor time-based code to use Clock
4. Write tests for the BookService class

Questions?