Complete Testing Cheat-Sheet

🚀 Quick Reference für Java Testing Mastery

Von Dr. Cassian Holt - Java Fleet Systems Consulting

🏗️ JUnit 5 - Core Annotations

```
// Basic Test Structure
@Test
@DisplayName("Should calculate tax for German customers correctly")
void shouldCalculateTaxForGermanCustomers() {
    // Given (Arrange)
    TaxCalculator calculator = new TaxCalculator();
    Customer germanCustomer = new Customer("DE", CustomerType.REGULAR);

    // When (Act)
    BigDecimal tax = calculator.calculateTax(germanCustomer, new
BigDecimal("100.00"));

    // Then (Assert)
    assertThat(tax).isEqualTo(new BigDecimal("19.00"));
}

// Lifecycle Hooks
@BeforeAll    // Runs once before all tests in class
@BeforeEach   // Runs before each test method
@AfterEach    // Runs after each test method
@AfterAll     // Runs once after all tests in class

// Conditional Testing
@EnabledOnOs(OS.LINUX)
@EnabledOnJre(JRE.JAVA_17)
@EnabledIf("customCondition()")

// Parameterized Tests
@ParameterizedTest
@ValueSource(strings = {"", "  ", "null"})
void shouldRejectInvalidNames(String name) { }

@ParameterizedTest
@CsvSource({
    "19.99, STANDARD, 19.99",
    "19.99, VIP, 15.99",
    "99.99, STUDENT, 89.99"
})
void shouldCalculateDiscountedPrice(String basePrice, String customerType,
String expected) { }

// Timeout Testing
@Test
@Timeout(value = 2, unit = SECONDS)
void shouldCompleteWithinTimeout() { }
```

🔍 AssertJ - Fluent Assertions

```
// Basic Assertions
assertThat(actual).isEqualTo(expected);
assertThat(actual).isNotNull().isInstanceOf(String.class);

// String Assertions
assertThat(email)
    .isNotBlank()
    .contains("@")
    .startsWith("user")
    .endsWith(".com");

// Number Assertions
assertThat(price)
    .isPositive()
    .isBetween(new BigDecimal("10.00"), new BigDecimal("100.00"))
    .isEqualTo(new BigDecimal("19.99"));

// Collection Assertions
assertThat(users)
    .hasSize(3)
    .extracting(User::getName)
    .containsExactly("Alice", "Bob", "Charlie")
    .doesNotContain("Dave");

// Exception Assertions
assertThatThrownBy(() -> calculator.divide(10, 0))
    .isInstanceOf(ArithmeticException.class)
    .hasMessage("Division by zero")
    .hasMessageContaining("zero");

// Soft Assertions (all assertions executed, report all failures)
SoftAssertions softly = new SoftAssertions();
softly.assertThat(user.getName()).isEqualTo("John");
softly.assertThat(user.getAge()).isEqualTo(30);
softly.assertThat(user.getEmail()).contains("@");
softly.assertAll(); // Fails if any assertion failed
```

🎭 Mockito - Mocking Framework

```
// Creating Mocks
@Mock
private PaymentService paymentService;

@MockBean // For Spring Boot Tests
private PaymentService paymentService;

// Stubbing
when(paymentService.processPayment(any(PaymentRequest.class)))
    .thenReturn(PaymentResult.success());

// Argument Matchers
when(service.findUser(eq("john@example.com"), anyInt()))
    .thenReturn(user);

// Verification
verify(paymentService).processPayment(paymentRequest);
verify(paymentService, times(2)).processPayment(any());
verify(paymentService, never()).cancelPayment(any());
```

```
// Argument Captors
@Captor
private ArgumentCaptor<PaymentRequest> paymentCaptor;

verify(paymentService).processPayment(paymentCaptor.capture());
PaymentRequest captured = paymentCaptor.getValue();
assertThat(captured.getAmount()).isEqualTo(new BigDecimal("100.00"));

// Spy (Partial Mocking)
@Spy
private PaymentCalculator calculator = new PaymentCalculator();

doReturn(new BigDecimal("10.00")).when(calculator).calculateFee(any());
```

🏔️ Test Pyramid - Distribution Guide

```
        /\      E2E Tests (10%)
       /  \     - Full system tests
      /____\    - UI automation
     /      \   - Slow, expensive
    /________\
   /          \ Integration Tests (20%)
  /____________\  - Service interactions
 /              \ - Database tests
/________________\ Unit Tests (70%)
                   - Fast, isolated
                   - Business logic
                   - Pure functions
```

✅ DO: 70% Unit, 20% Integration, 10% E2E
❌ DON'T: Ice Cream Cone (more E2E than Unit)

🔄 TDD - Red Green Refactor

```
// 1. RED - Write failing test first
@Test
void shouldCalculateDiscountForVipCustomers() {
    VipCustomer customer = new VipCustomer("john@example.com");
    DiscountCalculator calculator = new DiscountCalculator();

    BigDecimal discount = calculator.calculateDiscount(customer, new
BigDecimal("100.00"));

    assertThat(discount).isEqualTo(new BigDecimal("20.00")); // 20% VIP discount
}

// 2. GREEN - Write minimal code to pass
public class DiscountCalculator {
    public BigDecimal calculateDiscount(Customer customer, BigDecimal amount) {
        if (customer instanceof VipCustomer) {
            return amount.multiply(new BigDecimal("0.20"));
        }
        return BigDecimal.ZERO;
    }
}
```

```
// 3. REFACTOR - Improve code while keeping tests green
public class DiscountCalculator {
    private static final BigDecimal VIP_DISCOUNT_RATE = new BigDecimal("0.20");

    public BigDecimal calculateDiscount(Customer customer, BigDecimal amount) {
        return customer.getDiscountRate().multiply(amount);
    }
}
```

🐳 TestContainers - Integration Testing

```
@Testcontainers
class DatabaseIntegrationTest {

    @Container
    static PostgreSQLContainer<?> postgres = new
PostgreSQLContainer<>("postgres:15")
            .withDatabaseName("testdb")
            .withUsername("test")
            .withPassword("test");

    @DynamicPropertySource
    static void configureProperties(DynamicPropertyRegistry registry) {
        registry.add("spring.datasource.url", postgres::getJdbcUrl);
        registry.add("spring.datasource.username", postgres::getUsername);
        registry.add("spring.datasource.password", postgres::getPassword);
    }

    @Test
    void shouldPersistUser() {
        // Test with real database
        User user = userRepository.save(new User("john@example.com"));
        assertThat(user.getId()).isNotNull();
    }
}
```

🧬 Mutation Testing - PIT Configuration

```
<!-- pom.xml -->
<plugin>
    <groupId>org.pitest</groupId>
    <artifactId>pitest-maven</artifactId>
    <version>1.15.0</version>
    <configuration>
        <targetClasses>
            <param>com.yourcompany.business.*</param>
        </targetClasses>
        <targetTests>
            <param>com.yourcompany.business.*Test</param>
        </targetTests>
        <mutationThreshold>80</mutationThreshold>
        <coverageThreshold>90</coverageThreshold>
    </configuration>
</plugin>

<!-- Run: mvn org.pitest:pitest-maven:mutationCoverage -->
```

🔐 Security Testing Patterns

```
// Input Validation Testing
@ParameterizedTest
@ValueSource(strings = {"<script>alert('xss')</script>", "'; DROP TABLE users;
--", "../../../etc/passwd"})
void shouldRejectMaliciousInput(String maliciousInput) {
    assertThatThrownBy(() -> userService.createUser(maliciousInput))
        .isInstanceOf(ValidationException.class);
}

// Authentication Testing
@Test
@WithMockUser(roles = "USER")
void shouldAllowUserAccess() { }

@Test
@WithMockUser(roles = "ADMIN")
void shouldAllowAdminAccess() { }

@Test
void shouldDenyUnauthenticatedAccess() {
    assertThatThrownBy(() -> protectedService.getData())
        .isInstanceOf(AccessDeniedException.class);
}
```

🎯 Property-Based Testing (jqwik)

```
@Property
void listReverseTwiceShouldBeIdentical(@ForAll List<Integer> list) {
    List<Integer> reversed = reverse(reverse(list));
    assertThat(reversed).isEqualTo(list);
}

@Property
void additionIsCommutative(@ForAll int a, @ForAll int b) {
    assertThat(a + b).isEqualTo(b + a);
}

@Provide
Arbitrary<User> validUsers() {
    return Combinators.combine(
        Arbitraries.strings().alpha().ofMinLength(1),
        Arbitraries.integers().between(18, 100),
        Arbitraries.of("user@example.com", "test@company.de")
    ).as(User::new);
}
```

⚡ Performance Testing Patterns

```
@Test
@Timeout(value = 500, unit = MILLISECONDS)
void shouldCompleteWithinSLA() {
    // Test must complete within 500ms
```

```
    List<User> users = userService.findAllUsers();
    assertThat(users).hasSizeGreaterThan(0);
}

// Microbenchmark with JMH
@BenchmarkMode(Mode.AverageTime)
@OutputTimeUnit(TimeUnit.NANOSECONDS)
@State(Scope.Benchmark)
public class PerformanceBenchmark {

    @Benchmark
    public String concatenateStrings() {
        return "Hello" + " " + "World";
    }

    @Benchmark
    public String useStringBuilder() {
        return new StringBuilder("Hello").append("
").append("World").toString();
    }
}
```

🚨 Common Testing Anti-Patterns (AVOID!)

```
// ❌ Testing Implementation Details
@Test
void shouldCallRepositoryFindMethod() {
    userService.getUser("123");
    verify(userRepository).find("123"); // Tests HOW, not WHAT
}

// ✅ Testing Behavior
@Test
void shouldReturnUserWhenValidIdProvided() {
    User user = userService.getUser("123");
    assertThat(user.getId()).isEqualTo("123"); // Tests WHAT
}

// ❌ Brittle Tests (Too Many Mocks)
@Test
void brittleTest() {
    when(mock1.method1()).thenReturn(value1);
    when(mock2.method2()).thenReturn(value2);
    when(mock3.method3()).thenReturn(value3); // Hard to maintain
}

// ❌ Mystery Guest (Hidden Test Data)
@Test
void mysteryGuest() {
    User user = TestData.getUser(); // What user? What properties?
    // Test logic...
}

// ✅ Clear Test Data
@Test
void clearTestData() {
    User vipCustomer = new User.Builder()
        .withEmail("vip@example.com")
        .withStatus(CustomerStatus.VIP)
        .withRegistrationDate(LocalDate.now().minusYears(2))
```

```
        .build();
}
```

📊 Test Coverage vs Quality

```
Coverage Metrics:
- Line Coverage: % of code lines executed
- Branch Coverage: % of decision branches taken
- Method Coverage: % of methods called

Quality Metrics (More Important!):
- Mutation Score: % of mutants killed by tests
- Assertion Coverage: Tests with meaningful assertions
- Business Rule Coverage: Critical logic tested

✅ Target: 80%+ coverage with 85%+ mutation score
❌ Avoid: 100% coverage with poor assertions
```

🛠️ Spring Boot Testing Stack

```
// Full Integration Test
@SpringBootTest
@AutoConfigureTestDatabase(replace = NONE)
@Testcontainers
class ApplicationIntegrationTest { }

// Web Layer Only
@WebMvcTest(UserController.class)
class UserControllerTest {
    @MockBean UserService userService;
    @Autowired MockMvc mockMvc;
}

// Data Layer Only
@DataJpaTest
class UserRepositoryTest {
    @Autowired TestEntityManager entityManager;
    @Autowired UserRepository userRepository;
}

// JSON Serialization
@JsonTest
class UserJsonTest {
    @Autowired JacksonTester<User> json;
}
```

📋 Test Checklist

Before Writing Tests:

- [ ] Understand the business requirement
- [ ] Identify happy path and edge cases
- [ ] Choose appropriate test level (Unit/Integration/E2E)

Writing Good Tests:

- [ ] Clear, descriptive test names
- [ ] Given-When-Then structure
- [ ] Single responsibility per test
- [ ] Meaningful assertions with error messages

Test Maintenance:

- [ ] Remove flaky tests immediately
- [ ] Keep tests independent (can run in any order)
- [ ] Regular test suite cleanup
- [ ] Monitor test execution times

🎯 Quick Commands Reference

```
# Maven
mvn test                          # Run all tests
mvn test -Dtest=UserServiceTest   # Run specific test class
mvn test -Dtest="*Integration*"   # Run integration tests
mvn org.pitest:pitest-maven:mutationCoverage  # Mutation testing

# Gradle
./gradlew test                    # Run all tests
./gradlew test --tests UserServiceTest  # Run specific test
./gradlew test --continuous       # Continuous testing

# JUnit Platform Console
java -jar junit-platform-console-standalone.jar --scan-classpath
```

🚀*Keep this cheat-sheet handy for daily testing work!*
💡 *Remember: Good tests are your safety net for confident refactoring*