Case Study: Product-Order Management System (With Mockito Testing)

Objective Develop a simple Product-Order system using Spring Boot with MySQL. Test the business logic of services using Mockito. No integration testing or H2 database involved.

// OrderController.java

```
package com.example.springtest.controller;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
import com.example.springtest.entity.Order;
import com.example.springtest.service.OrderService;
@RestController
@RequestMapping("/api/orders")
public class OrderController {
    @Autowired
    private OrderService orderService;
    @PostMapping
    public Order placeOrder(@RequestParam Long productId, @RequestParam int quantity) {
        return orderService.placeOrder(productId, quantity);
    }
    @GetMapping
    public List<Order> getAllOrders() {
        return orderService.getAllOrders();
    }
}
```

// ProductController.java

```
package com.example.springtest.controller;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
```

```
import org.springframework.web.bind.annotation.*;
import com.example.springtest.entity.Product;
import com.example.springtest.service.ProductService;
@RestController
@RequestMapping("/api/products")
public class ProductController {
    @Autowired
    private ProductService productService;
    @PostMapping
    public Product addProduct(@RequestBody Product product) {
        return productService.addProduct(product);
    }
    @GetMapping
    public List<Product> getAllProducts() {
        return productService.getAllProducts();
    }
    @PutMapping("/{id}/stock")
    public Product updateStock(@PathVariable Long id, @RequestParam int quantity) {
        return productService.updateStock(id, quantity);
    }
}
```

// Order.java

```
package com.example.springtest.entity;
import java.time.LocalDateTime;
import jakarta.persistence.*;
@Entity
@Table(name = "orders")
public class Order {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
    private Long orderId;
    @ManyToOne
    @JoinColumn(name = "product_id")
    private Product product;
    private LocalDateTime orderDate;
    private int quantityOrdered;
    public Order() {}
    public Order(Long orderId, Product product, LocalDateTime orderDate, int quantityOrdered) {
        this.orderId = orderId;
        this.product = product;
        this.orderDate = orderDate;
        this.quantityOrdered = quantityOrdered;
    }
public Long getOrderId() {
                return orderId;
        }
        public void setOrderId(Long orderId) {
                this.orderId = orderId;
        }
        public Product getProduct() {
                return product;
        }
        public void setProduct(Product product) {
                this.product = product;
        }
        public LocalDate getOrderDate() {
                return orderDate;
        }
        public void setOrderDate(LocalDate orderDate) {
                this.orderDate = orderDate;
        }
```

```
	public int getQuantityOrdered() {
		return quantityOrdered;
	}
	public void setQuantityOrdered(int quantityOrdered) {
		this.quantityOrdered = quantityOrdered;
	}

}
```

// Product.java

```
package com.example.springtest.entity;
import jakarta.persistence.*;
@Entity
@Table(name = "products")
public class Product {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long productId;
   private String name;
   private double price;
   private int availableQuantity;
   public Product() {}
   public Product(Long productId, String name, double price, int availableQuantity) {
      this.productId = productId;
      this.name = name;
      this.price = price;
      this.availableQuantity = availableQuantity;
   }
public Long getProductId() {
		return productId;
	}
```

```
	public void setProductId(Long productId) {
		this.productId = productId;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public double getPrice() {
		return price;
	}
	public void setPrice(double price) {
		this.price = price;
	}
	public int getAvailableQuantity() {
		return availableQuantity;
	}
	public void setAvailableQuantity(int availableQuantity) {
		this.availableQuantity = availableQuantity;
	}

}
```

// OrderRepository.java

```
package com.example.springtest.repository;
import org.springframework.data.jpa.repository.JpaRepository;
import com.example.springtest.entity.Order;
public interface OrderRepository extends JpaRepository<Order, Long> {}
```

// ProductRepository.java

```
package com.example.springtest.repository;
import org.springframework.data.jpa.repository.JpaRepository;
import com.example.springtest.entity.Product;
public interface ProductRepository extends JpaRepository<Product, Long> {}

// OrderService.java
package com.example.springtest.service;
import java.time.LocalDateTime;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.example.springtest.entity.Order;
import com.example.springtest.entity.Product;
import com.example.springtest.repository.OrderRepository;
import com.example.springtest.repository.ProductRepository;
@Service
public class OrderService {
    @Autowired
    private OrderRepository orderRepository;
    @Autowired
    private ProductRepository productRepository;
    public Order placeOrder(Long productId, int quantity) {
        Product product = productRepository.findById(productId)
            .orElseThrow(() -> new RuntimeException("Product not found"));
        if (product.getAvailableQuantity() < quantity) {
            throw new RuntimeException("Insufficient stock");
        }
        product.setAvailableQuantity(product.getAvailableQuantity() - quantity);
        productRepository.save(product);
        Order order = new Order();
        order.setProduct(product);
```

```
        order.setOrderDate(LocalDateTime.now());
        order.setQuantityOrdered(quantity);
        return orderRepository.save(order);
    }
    public List<Order> getAllOrders() {
        return orderRepository.findAll();
    }
}
```

// ProductService.java

```
package com.example.springtest.service;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.example.springtest.entity.Product;
import com.example.springtest.repository.ProductRepository;
@Service
public class ProductService {
    @Autowired
    private ProductRepository productRepository;
    public Product addProduct(Product product) {
        return productRepository.save(product);
    }
    public List<Product> getAllProducts() {
        return productRepository.findAll();
    }
    public Product updateStock(Long productId, int quantity) {
        Product product = productRepository.findById(productId)
            .orElseThrow(() -> new RuntimeException("Product not found"));
        product.setAvailableQuantity(quantity);
        return productRepository.save(product);
```

```
    }
}

// OrderServiceTest.java
package com.example.springtest;
import static org.assertj.core.api.Assertions.*;
import static org.mockito.Mockito.*;
import java.time.LocalDateTime;
import java.util.*;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.*;
import org.mockito.junit.jupiter.MockitoExtension;
import com.example.springtest.entity.*;
import com.example.springtest.repository.*;
import com.example.springtest.service.OrderService;
@ExtendWith(MockitoExtension.class)
public class OrderServiceTest {

    @Mock
    private OrderRepository orderRepository;
    @Mock
    private ProductRepository productRepository;
    @InjectMocks
    private OrderService orderService;
    @Test
    public void testPlaceOrder_Success() {
        Product greenTea = new Product(1L, "Green Tea", 120, 50);
        Order expectedOrder = new Order(1L, greenTea, LocalDateTime.now(), 5);
        when(productRepository.findById(1L)).thenReturn(Optional.of(greenTea));
        when(orderRepository.save(any(Order.class))).thenReturn(expectedOrder);
```

```
        Order result = orderService.placeOrder(1L, 5);
        assertThat(result).isNotNull();
        assertThat(result.getQuantityOrdered()).isEqualTo(5);
        assertThat(result.getProduct().getName()).isEqualTo("Green Tea");
    }
    @Test
    public void testPlaceOrder_InsufficientStock() {
        Product proteinBar = new Product(2L, "Protein Bar", 45, 3);
        when(productRepository.findById(2L)).thenReturn(Optional.of(proteinBar));
        assertThatThrownBy(() -> orderService.placeOrder(2L, 10))
            .isInstanceOf(RuntimeException.class)
            .hasMessageContaining("Insufficient stock");
        verify(orderRepository, never()).save(any());
    }
    @Test
    public void testGetAllOrders() {
        Product honey = new Product(1L, "Honey", 300, 20);
        Product oats = new Product(2L, "Oats", 180, 15);
        Order o1 = new Order(1L, honey, LocalDateTime.now(), 2);
        Order o2 = new Order(2L, oats, LocalDateTime.now(), 4);
        when(orderRepository.findAll()).thenReturn(Arrays.asList(o1, o2));
        List<Order> result = orderService.getAllOrders();
        assertThat(result)
            .hasSize(2)
            .extracting(Order::getProduct)
            .extracting(Product::getName)
            .containsExactly("Honey", "Oats");
    }
}
```

// ProductServiceTest.java

```
package com.example.springtest;
import static org.assertj.core.api.Assertions.*;
import static org.mockito.Mockito.*;
import java.util.*;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.*;
import org.mockito.junit.jupiter.MockitoExtension;
import com.example.springtest.entity.Product;
import com.example.springtest.repository.ProductRepository;
import com.example.springtest.service.ProductService;
@ExtendWith(MockitoExtension.class)
public class ProductServiceTest {
   @Mock
   private ProductRepository productRepository;
   @InjectMocks
   private ProductService productService;
   @Test
   public void testAddProduct() {
      Product almonds = new Product(1L, "Almonds", 500, 60);
      when(productRepository.save(any(Product.class))).thenReturn(almonds);
      Product result = productService.addProduct(almonds);
      assertThat(result)
         .isNotNull()
         .extracting(Product::getName, Product::getPrice)
         .containsExactly("Almonds", 500.0);
   }
   @Test
   public void testGetAllProducts() {
      Product pasta = new Product(1L, "Pasta", 75, 80);
      Product cheese = new Product(2L, "Cheese", 180, 50);
```

```
        when(productRepository.findAll()).thenReturn(Arrays.asList(pasta, cheese));
        List<Product> result = productService.getAllProducts();
        assertThat(result)
            .hasSize(2)
            .extracting(Product::getName)
            .containsExactly("Pasta", "Cheese");
    }
    @Test
    public void testUpdateStock() {
        Product cereal = new Product(1L, "Cereal", 130, 40);
        Product updated = new Product(1L, "Cereal", 130, 100);
        when(productRepository.findById(1L)).thenReturn(Optional.of(cereal));
        when(productRepository.save(any(Product.class))).thenReturn(updated);
        Product result = productService.updateStock(1L, 100);
        assertThat(result.getAvailableQuantity()).isEqualTo(100);
    }
    @Test
    public void testUpdateStock_ProductNotFound() {
        when(productRepository.findById(99L)).thenReturn(Optional.empty());
        assertThatThrownBy(() -> productService.updateStock(99L, 50))
            .isInstanceOf(RuntimeException.class)
            .hasMessageContaining("Product not found");
        verify(productRepository, never()).save(any());
    }
}
```