**Mockito :Customer Onboarding TDD Example**

**Business Requirements**

**Domain Context:** A bank needs to implement a customer onboarding system that validates customer information, checks eligibility, and processes account creation requests.

**Primary Business Requirements:**

1. **Customer Information Validation**: The system must validate basic customer information (name, email, phone) before proceeding with onboarding
2. **Eligibility Verification**: The system must check customer eligibility based on credit score and age requirements
3. **Account Creation Processing**: The system must handle account creation requests and manage potential system failures gracefully

**Testable Requirements Derived from Business Requirements**

**Requirement 1: Customer Information Validation**

* **Given** a customer with valid basic information (non-empty name, valid email format, valid phone)
* **When** the validation service is called
* **Then** the system should return true indicating successful validation

**Requirement 2: Eligibility Check Processing**

* **Given** a customer with valid information
* **When** eligibility is checked and customer meets criteria (credit score ≥ 650, age ≥ 18)
* **Then** the system should return an eligibility status of "APPROVED"

**Requirement 3: Account Creation with Error Handling**

* **Given** an eligible customer attempting account creation
* **When** the external account service is unavailable
* **Then** the system should throw a ServiceUnavailableException with appropriate error message

**Prerequisites**

Add the following dependencies to your pom.xml:

<dependencies>  
 <dependency>  
 <groupId>org.junit.jupiter</groupId>  
 <artifactId>junit-jupiter</artifactId>  
 <version>5.9.2</version>  
 <scope>test</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.mockito</groupId>  
 <artifactId>mockito-core</artifactId>  
 <version>5.1.1</version>  
 <scope>test</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.mockito</groupId>  
 <artifactId>mockito-junit-jupiter</artifactId>  
 <version>5.1.1</version>  
 <scope>test</scope>  
 </dependency>  
</dependencies>

**PASS 1: RED-GREEN-REFACTOR (Requirement 1)**

**RED Phase - Write Failing Test**

**File:** src/test/java/edu/m001/CustomerOnboardingServiceTest.java

package edu.m001;  
  
// Import JUnit 5 testing framework annotations and assertions  
import org.junit.jupiter.api.Test;  
import org.junit.jupiter.api.BeforeEach;  
import static org.junit.jupiter.api.Assertions.\*;  
  
// Import Mockito framework for mocking dependencies  
import org.mockito.Mock;  
import org.mockito.MockitoAnnotations;  
import static org.mockito.Mockito.\*;  
  
// Test class for CustomerOnboardingService using TDD approach  
public class CustomerOnboardingServiceTest {  
   
 // Mock object for CustomerValidationService dependency  
 @Mock  
 private CustomerValidationService validationService;  
   
 // Mock object for EligibilityService dependency   
 @Mock  
 private EligibilityService eligibilityService;  
   
 // Mock object for AccountService dependency  
 @Mock  
 private AccountService accountService;  
   
 // System under test - the main service we're testing  
 private CustomerOnboardingService onboardingService;  
   
 // Setup method executed before each test method  
 @BeforeEach  
 void setUp() {  
 // Initialize all mock objects annotated with @Mock  
 MockitoAnnotations.openMocks(this);  
 // Create instance of system under test with mocked dependencies  
 onboardingService = new CustomerOnboardingService(validationService, eligibilityService, accountService);  
 }  
   
 // Test case for Requirement 1: Customer Information Validation  
 @Test  
 void shouldValidateCustomerInformation() {  
 // Arrange: Create a customer object with valid information  
 Customer customer = new Customer("John Doe", "john@email.com", "1234567890");  
   
 // Act & Assert: This will fail initially as classes don't exist  
 // Call the validation method and verify it returns true for valid customer  
 assertTrue(onboardingService.validateCustomer(customer));  
 }  
}

**Expected Result:** Compilation errors - classes don't exist yet.

**GREEN Phase - Make Test Pass**

**File:** src/main/java/edu/m001/Customer.java

package edu.m001;  
  
// Simple data class representing a customer with basic information  
public class Customer {  
 // Private fields to store customer information  
 private String name;  
 private String email;   
 private String phone;  
   
 // Constructor to initialize customer with required information  
 public Customer(String name, String email, String phone) {  
 this.name = name;  
 this.email = email;  
 this.phone = phone;  
 }  
   
 // Getter method to retrieve customer name  
 public String getName() {  
 return name;  
 }  
   
 // Getter method to retrieve customer email  
 public String getEmail() {  
 return email;  
 }  
   
 // Getter method to retrieve customer phone  
 public String getPhone() {  
 return phone;  
 }  
}

**File:** src/main/java/edu/m001/CustomerValidationService.java

package edu.m001;  
  
// Interface defining contract for customer validation operations  
public interface CustomerValidationService {  
 // Method to validate customer basic information  
 // Returns true if customer information is valid, false otherwise  
 boolean validateBasicInfo(Customer customer);  
}

**File:** src/main/java/edu/m001/EligibilityService.java

package edu.m001;  
  
// Interface defining contract for customer eligibility operations   
public interface EligibilityService {  
 // Method to check customer eligibility for account creation  
 // Returns eligibility status based on business rules  
 String checkEligibility(Customer customer);  
}

**File:** src/main/java/edu/m001/AccountService.java

package edu.m001;  
  
// Interface defining contract for account creation operations  
public interface AccountService {  
 // Method to create account for eligible customer  
 // Returns account number if successful, throws exception if failed  
 String createAccount(Customer customer);  
}

**File:** src/main/java/edu/m001/CustomerOnboardingService.java

package edu.m001;  
  
// Main service class orchestrating customer onboarding process  
public class CustomerOnboardingService {  
 // Dependency for validating customer information  
 private CustomerValidationService validationService;  
 // Dependency for checking customer eligibility  
 private EligibilityService eligibilityService;  
 // Dependency for creating customer accounts  
 private AccountService accountService;  
   
 // Constructor injection of all required dependencies  
 public CustomerOnboardingService(CustomerValidationService validationService,   
 EligibilityService eligibilityService,  
 AccountService accountService) {  
 this.validationService = validationService;  
 this.eligibilityService = eligibilityService;  
 this.accountService = accountService;  
 }  
   
 // Method to validate customer information using validation service  
 public boolean validateCustomer(Customer customer) {  
 // Delegate validation to the validation service  
 return validationService.validateBasicInfo(customer);  
 }  
}

**Updated Test File:** src/test/java/edu/m001/CustomerOnboardingServiceTest.java

package edu.m001;  
  
// Import JUnit 5 testing framework annotations and assertions  
import org.junit.jupiter.api.Test;  
import org.junit.jupiter.api.BeforeEach;  
import static org.junit.jupiter.api.Assertions.\*;  
  
// Import Mockito framework for mocking dependencies  
import org.mockito.Mock;  
import org.mockito.MockitoAnnotations;  
import static org.mockito.Mockito.\*;  
  
// Test class for CustomerOnboardingService using TDD approach  
public class CustomerOnboardingServiceTest {  
   
 // Mock object for CustomerValidationService dependency  
 @Mock  
 private CustomerValidationService validationService;  
   
 // Mock object for EligibilityService dependency   
 @Mock  
 private EligibilityService eligibilityService;  
   
 // Mock object for AccountService dependency  
 @Mock  
 private AccountService accountService;  
   
 // System under test - the main service we're testing  
 private CustomerOnboardingService onboardingService;  
   
 // Setup method executed before each test method  
 @BeforeEach  
 void setUp() {  
 // Initialize all mock objects annotated with @Mock  
 MockitoAnnotations.openMocks(this);  
 // Create instance of system under test with mocked dependencies  
 onboardingService = new CustomerOnboardingService(validationService, eligibilityService, accountService);  
 }  
   
 // Test case for Requirement 1: Customer Information Validation  
 @Test  
 void shouldValidateCustomerInformation() {  
 // Arrange: Create a customer object with valid information  
 Customer customer = new Customer("John Doe", "john@email.com", "1234567890");  
   
 // Mockito stubbing using when() - specifies method call to be stubbed  
 when(validationService.validateBasicInfo(customer)).thenReturn(true);  
   
 // Act: Call the validation method on our service  
 boolean result = onboardingService.validateCustomer(customer);  
   
 // Assert: Verify the result matches expected outcome  
 assertTrue(result);  
   
 // Verify that the mock was called exactly once with correct parameter  
 verify(validationService, times(1)).validateBasicInfo(customer);  
 }  
}

**Expected Result:** Test passes ✅

**REFACTOR Phase**

No refactoring needed at this stage - code is simple and clean.

**PASS 2: RED-GREEN-REFACTOR (Requirement 2)**

**RED Phase - Add Failing Test**

**Updated Test File:** src/test/java/edu/m001/CustomerOnboardingServiceTest.java

package edu.m001;  
  
// Import JUnit 5 testing framework annotations and assertions  
import org.junit.jupiter.api.Test;  
import org.junit.jupiter.api.BeforeEach;  
import static org.junit.jupiter.api.Assertions.\*;  
  
// Import Mockito framework for mocking dependencies  
import org.mockito.Mock;  
import org.mockito.MockitoAnnotations;  
import static org.mockito.Mockito.\*;  
  
// Test class for CustomerOnboardingService using TDD approach  
public class CustomerOnboardingServiceTest {  
   
 // Mock object for CustomerValidationService dependency  
 @Mock  
 private CustomerValidationService validationService;  
   
 // Mock object for EligibilityService dependency   
 @Mock  
 private EligibilityService eligibilityService;  
   
 // Mock object for AccountService dependency  
 @Mock  
 private AccountService accountService;  
   
 // System under test - the main service we're testing  
 private CustomerOnboardingService onboardingService;  
   
 // Setup method executed before each test method  
 @BeforeEach  
 void setUp() {  
 // Initialize all mock objects annotated with @Mock  
 MockitoAnnotations.openMocks(this);  
 // Create instance of system under test with mocked dependencies  
 onboardingService = new CustomerOnboardingService(validationService, eligibilityService, accountService);  
 }  
   
 // Test case for Requirement 1: Customer Information Validation  
 @Test  
 void shouldValidateCustomerInformation() {  
 // Arrange: Create a customer object with valid information  
 Customer customer = new Customer("John Doe", "john@email.com", "1234567890");  
   
 // Mockito stubbing using when() - specifies method call to be stubbed  
 when(validationService.validateBasicInfo(customer)).thenReturn(true);  
   
 // Act: Call the validation method on our service  
 boolean result = onboardingService.validateCustomer(customer);  
   
 // Assert: Verify the result matches expected outcome  
 assertTrue(result);  
   
 // Verify that the mock was called exactly once with correct parameter  
 verify(validationService, times(1)).validateBasicInfo(customer);  
 }  
   
 // Test case for Requirement 2: Eligibility Verification (NEW TEST)  
 @Test  
 void shouldCheckCustomerEligibility() {  
 // Arrange: Create a customer object for eligibility check  
 Customer customer = new Customer("Jane Smith", "jane@email.com", "0987654321");  
   
 // Act & Assert: This will fail initially as method doesn't exist  
 // Call the eligibility check method and verify it returns approved status  
 String eligibilityStatus = onboardingService.checkEligibility(customer);  
 assertEquals("APPROVED", eligibilityStatus);  
 }  
}

**Expected Result:** Compilation error - checkEligibility method doesn't exist.

**GREEN Phase - Make Test Pass**

**Updated Implementation File:** src/main/java/edu/m001/CustomerOnboardingService.java

package edu.m001;  
  
// Main service class orchestrating customer onboarding process  
public class CustomerOnboardingService {  
 // Dependency for validating customer information  
 private CustomerValidationService validationService;  
 // Dependency for checking customer eligibility  
 private EligibilityService eligibilityService;  
 // Dependency for creating customer accounts  
 private AccountService accountService;  
   
 // Constructor injection of all required dependencies  
 public CustomerOnboardingService(CustomerValidationService validationService,   
 EligibilityService eligibilityService,  
 AccountService accountService) {  
 this.validationService = validationService;  
 this.eligibilityService = eligibilityService;  
 this.accountService = accountService;  
 }  
   
 // Method to validate customer information using validation service  
 public boolean validateCustomer(Customer customer) {  
 // Delegate validation to the validation service  
 return validationService.validateBasicInfo(customer);  
 }  
   
 // NEW METHOD: Method to check customer eligibility using eligibility service  
 public String checkEligibility(Customer customer) {  
 // Delegate eligibility check to the eligibility service  
 return eligibilityService.checkEligibility(customer);  
 }  
}

**Updated Test File with Proper Stubbing:** src/test/java/edu/m001/CustomerOnboardingServiceTest.java

package edu.m001;  
  
// Import JUnit 5 testing framework annotations and assertions  
import org.junit.jupiter.api.Test;  
import org.junit.jupiter.api.BeforeEach;  
import static org.junit.jupiter.api.Assertions.\*;  
  
// Import Mockito framework for mocking dependencies  
import org.mockito.Mock;  
import org.mockito.MockitoAnnotations;  
import static org.mockito.Mockito.\*;  
  
// Test class for CustomerOnboardingService using TDD approach  
public class CustomerOnboardingServiceTest {  
   
 // Mock object for CustomerValidationService dependency  
 @Mock  
 private CustomerValidationService validationService;  
   
 // Mock object for EligibilityService dependency   
 @Mock  
 private EligibilityService eligibilityService;  
   
 // Mock object for AccountService dependency  
 @Mock  
 private AccountService accountService;  
   
 // System under test - the main service we're testing  
 private CustomerOnboardingService onboardingService;  
   
 // Setup method executed before each test method  
 @BeforeEach  
 void setUp() {  
 // Initialize all mock objects annotated with @Mock  
 MockitoAnnotations.openMocks(this);  
 // Create instance of system under test with mocked dependencies  
 onboardingService = new CustomerOnboardingService(validationService, eligibilityService, accountService);  
 }  
   
 // Test case for Requirement 1: Customer Information Validation  
 @Test  
 void shouldValidateCustomerInformation() {  
 // Arrange: Create a customer object with valid information  
 Customer customer = new Customer("John Doe", "john@email.com", "1234567890");  
   
 // Mockito stubbing using when() - specifies method call to be stubbed  
 when(validationService.validateBasicInfo(customer)).thenReturn(true);  
   
 // Act: Call the validation method on our service  
 boolean result = onboardingService.validateCustomer(customer);  
   
 // Assert: Verify the result matches expected outcome  
 assertTrue(result);  
   
 // Verify that the mock was called exactly once with correct parameter  
 verify(validationService, times(1)).validateBasicInfo(customer);  
 }  
   
 // Test case for Requirement 2: Eligibility Verification  
 @Test  
 void shouldCheckCustomerEligibility() {  
 // Arrange: Create a customer object for eligibility check  
 Customer customer = new Customer("Jane Smith", "jane@email.com", "0987654321");  
   
 // Mockito stubbing using when() and thenReturn() - sets return value for stubbed method call  
 when(eligibilityService.checkEligibility(customer)).thenReturn("APPROVED");  
   
 // Act: Call the eligibility check method on our service  
 String eligibilityStatus = onboardingService.checkEligibility(customer);  
   
 // Assert: Verify the result matches expected outcome  
 assertEquals("APPROVED", eligibilityStatus);  
   
 // Verify that the mock was called exactly once with correct parameter  
 verify(eligibilityService, times(1)).checkEligibility(customer);  
 }  
}

**Expected Result:** Test passes ✅

**REFACTOR Phase**

No refactoring needed - code remains clean and focused.

**PASS 3: RED-GREEN-REFACTOR (Requirement 3)**

**RED Phase - Add Failing Test**

**Updated Test File:** src/test/java/edu/m001/CustomerOnboardingServiceTest.java

package edu.m001;  
  
// Import JUnit 5 testing framework annotations and assertions  
import org.junit.jupiter.api.Test;  
import org.junit.jupiter.api.BeforeEach;  
import static org.junit.jupiter.api.Assertions.\*;  
  
// Import Mockito framework for mocking dependencies  
import org.mockito.Mock;  
import org.mockito.MockitoAnnotations;  
import static org.mockito.Mockito.\*;  
  
// Test class for CustomerOnboardingService using TDD approach  
public class CustomerOnboardingServiceTest {  
   
 // Mock object for CustomerValidationService dependency  
 @Mock  
 private CustomerValidationService validationService;  
   
 // Mock object for EligibilityService dependency   
 @Mock  
 private EligibilityService eligibilityService;  
   
 // Mock object for AccountService dependency  
 @Mock  
 private AccountService accountService;  
   
 // System under test - the main service we're testing  
 private CustomerOnboardingService onboardingService;  
   
 // Setup method executed before each test method  
 @BeforeEach  
 void setUp() {  
 // Initialize all mock objects annotated with @Mock  
 MockitoAnnotations.openMocks(this);  
 // Create instance of system under test with mocked dependencies  
 onboardingService = new CustomerOnboardingService(validationService, eligibilityService, accountService);  
 }  
   
 // Test case for Requirement 1: Customer Information Validation  
 @Test  
 void shouldValidateCustomerInformation() {  
 // Arrange: Create a customer object with valid information  
 Customer customer = new Customer("John Doe", "john@email.com", "1234567890");  
   
 // Mockito stubbing using when() - specifies method call to be stubbed  
 when(validationService.validateBasicInfo(customer)).thenReturn(true);  
   
 // Act: Call the validation method on our service  
 boolean result = onboardingService.validateCustomer(customer);  
   
 // Assert: Verify the result matches expected outcome  
 assertTrue(result);  
   
 // Verify that the mock was called exactly once with correct parameter  
 verify(validationService, times(1)).validateBasicInfo(customer);  
 }  
   
 // Test case for Requirement 2: Eligibility Verification  
 @Test  
 void shouldCheckCustomerEligibility() {  
 // Arrange: Create a customer object for eligibility check  
 Customer customer = new Customer("Jane Smith", "jane@email.com", "0987654321");  
   
 // Mockito stubbing using when() and thenReturn() - sets return value for stubbed method call  
 when(eligibilityService.checkEligibility(customer)).thenReturn("APPROVED");  
   
 // Act: Call the eligibility check method on our service  
 String eligibilityStatus = onboardingService.checkEligibility(customer);  
   
 // Assert: Verify the result matches expected outcome  
 assertEquals("APPROVED", eligibilityStatus);  
   
 // Verify that the mock was called exactly once with correct parameter  
 verify(eligibilityService, times(1)).checkEligibility(customer);  
 }  
   
 // Test case for Requirement 3: Account Creation with Error Handling (NEW TEST)  
 @Test  
 void shouldThrowExceptionWhenAccountServiceUnavailable() {  
 // Arrange: Create a customer object for account creation  
 Customer customer = new Customer("Bob Wilson", "bob@email.com", "5555555555");  
   
 // Act & Assert: This will fail initially as method and exception don't exist  
 // Verify that ServiceUnavailableException is thrown when account service fails  
 assertThrows(ServiceUnavailableException.class, () -> {  
 onboardingService.createAccount(customer);  
 });  
 }  
}

**Expected Result:** Compilation errors - createAccount method and ServiceUnavailableException don't exist.

**GREEN Phase - Make Test Pass**

**New Exception File:** src/main/java/edu/m001/ServiceUnavailableException.java

package edu.m001;  
  
// Custom exception class for handling service unavailability scenarios  
public class ServiceUnavailableException extends RuntimeException {  
   
 // Constructor accepting error message  
 public ServiceUnavailableException(String message) {  
 // Call parent constructor with error message  
 super(message);  
 }  
   
 // Constructor accepting error message and root cause  
 public ServiceUnavailableException(String message, Throwable cause) {  
 // Call parent constructor with error message and cause  
 super(message, cause);  
 }  
}

**Updated Implementation File:** src/main/java/edu/m001/CustomerOnboardingService.java

package edu.m001;  
  
// Main service class orchestrating customer onboarding process  
public class CustomerOnboardingService {  
 // Dependency for validating customer information  
 private CustomerValidationService validationService;  
 // Dependency for checking customer eligibility  
 private EligibilityService eligibilityService;  
 // Dependency for creating customer accounts  
 private AccountService accountService;  
   
 // Constructor injection of all required dependencies  
 public CustomerOnboardingService(CustomerValidationService validationService,   
 EligibilityService eligibilityService,  
 AccountService accountService) {  
 this.validationService = validationService;  
 this.eligibilityService = eligibilityService;  
 this.accountService = accountService;  
 }  
   
 // Method to validate customer information using validation service  
 public boolean validateCustomer(Customer customer) {  
 // Delegate validation to the validation service  
 return validationService.validateBasicInfo(customer);  
 }  
   
 // Method to check customer eligibility using eligibility service  
 public String checkEligibility(Customer customer) {  
 // Delegate eligibility check to the eligibility service  
 return eligibilityService.checkEligibility(customer);  
 }  
   
 // NEW METHOD: Method to create account using account service with error handling  
 public String createAccount(Customer customer) {  
 // Delegate account creation to the account service  
 // Any exceptions from account service will propagate up  
 return accountService.createAccount(customer);  
 }  
}

**Updated Test File with Exception Stubbing:** src/test/java/edu/m001/CustomerOnboardingServiceTest.java

package edu.m001;  
  
// Import JUnit 5 testing framework annotations and assertions  
import org.junit.jupiter.api.Test;  
import org.junit.jupiter.api.BeforeEach;  
import static org.junit.jupiter.api.Assertions.\*;  
  
// Import Mockito framework for mocking dependencies  
import org.mockito.Mock;  
import org.mockito.MockitoAnnotations;  
import static org.mockito.Mockito.\*;  
  
// Test class for CustomerOnboardingService using TDD approach  
public class CustomerOnboardingServiceTest {  
   
 // Mock object for CustomerValidationService dependency  
 @Mock  
 private CustomerValidationService validationService;  
   
 // Mock object for EligibilityService dependency   
 @Mock  
 private EligibilityService eligibilityService;  
   
 // Mock object for AccountService dependency  
 @Mock  
 private AccountService accountService;  
   
 // System under test - the main service we're testing  
 private CustomerOnboardingService onboardingService;  
   
 // Setup method executed before each test method  
 @BeforeEach  
 void setUp() {  
 // Initialize all mock objects annotated with @Mock  
 MockitoAnnotations.openMocks(this);  
 // Create instance of system under test with mocked dependencies  
 onboardingService = new CustomerOnboardingService(validationService, eligibilityService, accountService);  
 }  
   
 // Test case for Requirement 1: Customer Information Validation  
 @Test  
 void shouldValidateCustomerInformation() {  
 // Arrange: Create a customer object with valid information  
 Customer customer = new Customer("John Doe", "john@email.com", "1234567890");  
   
 // Mockito stubbing using when() - specifies method call to be stubbed  
 when(validationService.validateBasicInfo(customer)).thenReturn(true);  
   
 // Act: Call the validation method on our service  
 boolean result = onboardingService.validateCustomer(customer);  
   
 // Assert: Verify the result matches expected outcome  
 assertTrue(result);  
   
 // Verify that the mock was called exactly once with correct parameter  
 verify(validationService, times(1)).validateBasicInfo(customer);  
 }  
   
 // Test case for Requirement 2: Eligibility Verification  
 @Test  
 void shouldCheckCustomerEligibility() {  
 // Arrange: Create a customer object for eligibility check  
 Customer customer = new Customer("Jane Smith", "jane@email.com", "0987654321");  
   
 // Mockito stubbing using when() and thenReturn() - sets return value for stubbed method call  
 when(eligibilityService.checkEligibility(customer)).thenReturn("APPROVED");  
   
 // Act: Call the eligibility check method on our service  
 String eligibilityStatus = onboardingService.checkEligibility(customer);  
   
 // Assert: Verify the result matches expected outcome  
 assertEquals("APPROVED", eligibilityStatus);  
   
 // Verify that the mock was called exactly once with correct parameter  
 verify(eligibilityService, times(1)).checkEligibility(customer);  
 }  
   
 // Test case for Requirement 3: Account Creation with Error Handling  
 @Test  
 void shouldThrowExceptionWhenAccountServiceUnavailable() {  
 // Arrange: Create a customer object for account creation  
 Customer customer = new Customer("Bob Wilson", "bob@email.com", "5555555555");  
   
 // Mockito stubbing using when() and thenThrow() - throws exception when stubbed method called  
 when(accountService.createAccount(customer))  
 .thenThrow(new ServiceUnavailableException("Account service is currently unavailable"));  
   
 // Act & Assert: Verify that ServiceUnavailableException is thrown  
 ServiceUnavailableException exception = assertThrows(ServiceUnavailableException.class, () -> {  
 onboardingService.createAccount(customer);  
 });  
   
 // Assert: Verify the exception message is as expected  
 assertEquals("Account service is currently unavailable", exception.getMessage());  
   
 // Verify that the mock was called exactly once with correct parameter  
 verify(accountService, times(1)).createAccount(customer);  
 }  
}

**Expected Result:** All tests pass ✅

**REFACTOR Phase - Final Cleanup**

**Final Test File:** src/test/java/edu/m001/CustomerOnboardingServiceTest.java

package edu.m001;  
  
// Import JUnit 5 testing framework annotations and assertions  
import org.junit.jupiter.api.Test;  
import org.junit.jupiter.api.BeforeEach;  
import static org.junit.jupiter.api.Assertions.\*;  
  
// Import Mockito framework for mocking dependencies  
import org.mockito.Mock;  
import org.mockito.MockitoAnnotations;  
import static org.mockito.Mockito.\*;  
  
// Test class for CustomerOnboardingService using TDD approach with Mockito  
public class CustomerOnboardingServiceTest {  
   
 // Mock object for CustomerValidationService dependency  
 @Mock  
 private CustomerValidationService validationService;  
   
 // Mock object for EligibilityService dependency   
 @Mock  
 private EligibilityService eligibilityService;  
   
 // Mock object for AccountService dependency  
 @Mock  
 private AccountService accountService;  
   
 // System under test - the main service we're testing  
 private CustomerOnboardingService onboardingService;  
   
 // Setup method executed before each test method  
 @BeforeEach  
 void setUp() {  
 // Initialize all mock objects annotated with @Mock  
 MockitoAnnotations.openMocks(this);  
 // Create instance of system under test with mocked dependencies injected  
 onboardingService = new CustomerOnboardingService(validationService, eligibilityService, accountService);  
 }  
   
 // Test case for Requirement 1: Customer Information Validation  
 // Demonstrates when() method stubbing  
 @Test  
 void shouldValidateCustomerInformation() {  
 // Arrange: Create a customer object with valid information  
 Customer customer = new Customer("John Doe", "john@email.com", "1234567890");  
   
 // Mockito method stubbing using when() - specifies method call to be stubbed  
 when(validationService.validateBasicInfo(customer)).thenReturn(true);  
   
 // Act: Call the validation method on our service under test  
 boolean result = onboardingService.validateCustomer(customer);  
   
 // Assert: Verify the result matches expected outcome  
 assertTrue(result, "Customer validation should return true for valid customer");  
   
 // Verify that the mock was called exactly once with correct parameter  
 verify(validationService, times(1)).validateBasicInfo(customer);  
 }  
   
 // Test case for Requirement 2: Eligibility Verification   
 // Demonstrates thenReturn() method for setting return values  
 @Test  
 void shouldCheckCustomerEligibility() {  
 // Arrange: Create a customer object for eligibility check  
 Customer customer = new Customer("Jane Smith", "jane@email.com", "0987654321");  
   
 // Mockito method stubbing using thenReturn() - sets return value for stubbed method call  
 when(eligibilityService.checkEligibility(customer)).thenReturn("APPROVED");  
   
 // Act: Call the eligibility check method on our service under test  
 String eligibilityStatus = onboardingService.checkEligibility(customer);  
   
 // Assert: Verify the result matches expected outcome  
 assertEquals("APPROVED", eligibilityStatus, "Eligible customer should receive APPROVED status");  
   
 // Verify that the mock was called exactly once with correct parameter  
 verify(eligibilityService, times(1)).checkEligibility(customer);  
 }  
   
 // Test case for Requirement 3: Account Creation with Error Handling  
 // Demonstrates thenThrow() method for exception simulation  
 @Test  
 void shouldThrowExceptionWhenAccountServiceUnavailable() {  
 // Arrange: Create a customer object for account creation  
 Customer customer = new Customer("Bob Wilson", "bob@email.com", "5555555555");  
   
 // Mockito method stubbing using thenThrow() - throws exception when stubbed method called  
 when(accountService.createAccount(customer))  
 .thenThrow(new ServiceUnavailableException("Account service is currently unavailable"));  
   
 // Act & Assert: Verify that ServiceUnavailableException is thrown as expected  
 ServiceUnavailableException exception = assertThrows(ServiceUnavailableException.class, () -> {  
 onboardingService.createAccount(customer);  
 }, "Should throw ServiceUnavailableException when account service fails");  
   
 // Assert: Verify the exception message contains expected content  
 assertEquals("Account service is currently unavailable", exception.getMessage());  
   
 // Verify that the mock was called exactly once with correct parameter  
 verify(accountService, times(1)).createAccount(customer);  
 }  
}

**Summary**

This assignment successfully demonstrates TDD with Mockito using three key stubbing techniques:

1. **when()**: Method call stubbing specification
2. **thenReturn()**: Return value configuration for stubs
3. **thenThrow()**: Exception throwing simulation for error scenarios

Each requirement followed the complete RED-GREEN-REFACTOR cycle, building incrementally on the previous functionality while maintaining clean, testable code structure.