**TDD Stub Examples**

**Summary of Stubbing Usage**  
For every example, a hand-crafted stub class replaces real external dependencies, providing controlled, test-specific behavior. This enables deterministic, focused logic testing of the service in JUnit 5, where the unit interacts with external dependencies whose responses are simulated by the stub object. Each test method's comments explain its precise role and the stub's use within.

**Example 1: User Notification Filtering**

**Task Requirement:**  
Implement a NotificationService that filters notifications for a given user based on their user settings. The filtering logic should skip notifications marked as "Do Not Disturb" in settings. User settings come from a dependency UserSettingsProvider. Create unit tests using a hand-crafted stub for UserSettingsProvider.

**Test Class**

import org.junit.jupiter.api.Test;  
import static org.junit.jupiter.api.Assertions.\*;  
import java.util.\*;  
  
class NotificationServiceTest {  
 // Purpose: Verifies filtering out "DND" notifications according to stubbed settings.  
 @Test  
 void filtersNotificationsBasedOnUserSettings() {  
 // Stub simulates user settings (with "Do Not Disturb" enabled).  
 UserSettingsProviderStub stub = new UserSettingsProviderStub(true);  
  
 NotificationService service = new NotificationService(stub);  
  
 List<String> allNotifications = Arrays.asList("Meeting at 10am", "DND: Lunch Break", "DND: Focus Time", "General Update");  
 List<String> filtered = service.filterNotifications("alice", allNotifications);  
  
 assertEquals(Arrays.asList("Meeting at 10am", "General Update"), filtered,   
 "Should filter out notifications marked DND when 'Do Not Disturb' is enabled");  
 }  
  
 // Purpose: Ensures all notifications are shown if "Do Not Disturb" is off in stub.  
 @Test  
 void doesNotFilterIfDndOff() {  
 UserSettingsProviderStub stub = new UserSettingsProviderStub(false);  
 NotificationService service = new NotificationService(stub);  
  
 List<String> allNotifications = Arrays.asList("Meeting at 10am", "DND: Lunch Break");  
 List<String> filtered = service.filterNotifications("alice", allNotifications);  
  
 assertEquals(allNotifications, filtered, "All notifications should be shown when DND is off");  
 }  
}  
  
// Stub Explanation:  
// 'UserSettingsProviderStub' is used instead of the real settings provider.   
// It's a simple Java class that allows the test to simulate different user settings.  
// Where: Used directly in test methods, injected into service.  
// How: Constructor parameter determines simulated DND setting.  
  
// STUB: Hardcoded to simulate settings  
class UserSettingsProviderStub implements UserSettingsProvider {  
 private final boolean doNotDisturb;  
  
 public UserSettingsProviderStub(boolean doNotDisturb) {  
 this.doNotDisturb = doNotDisturb;  
 }  
 @Override  
 public boolean isDoNotDisturbEnabled(String username) {  
 return doNotDisturb;  
 }  
}

**Service and Stub Implementation**

// Service to be tested  
class NotificationService {  
 private final UserSettingsProvider settingsProvider;  
  
 public NotificationService(UserSettingsProvider settingsProvider) {  
 this.settingsProvider = settingsProvider;  
 }  
  
 public List<String> filterNotifications(String username, List<String> notifications) {  
 boolean doNotDisturb = settingsProvider.isDoNotDisturbEnabled(username);  
 if (!doNotDisturb) {  
 return notifications;  
 }  
 List<String> filtered = new ArrayList<>();  
 for (String note : notifications) {  
 if (!note.startsWith("DND:")) {  
 filtered.add(note);  
 }  
 }  
 return filtered;  
 }  
}  
  
// Interface for collaboration  
interface UserSettingsProvider {  
 boolean isDoNotDisturbEnabled(String username);  
}

**Example 2: Payment Processing Fee Calculation**

**Task Requirement:**  
Write PaymentProcessor that adds a transaction fee to payments based on user's account type from an external dependency AccountTypeProvider. Write tests using a stub for AccountTypeProvider to simulate different account types.

**Test Class**

import org.junit.jupiter.api.Test;  
import static org.junit.jupiter.api.Assertions.\*;  
  
class PaymentProcessorTest {  
 // Tests correct fee for premium account  
 @Test  
 void appliesLowFeeForPremium() {  
 AccountTypeProviderStub stub = new AccountTypeProviderStub("premium");  
 PaymentProcessor processor = new PaymentProcessor(stub);  
 assertEquals(102.0, processor.processPayment("u1", 100.0), 0.01);  
 }  
  
 // Tests correct fee for standard account  
 @Test  
 void appliesStandardFee() {  
 AccountTypeProviderStub stub = new AccountTypeProviderStub("standard");  
 PaymentProcessor processor = new PaymentProcessor(stub);  
 assertEquals(105.0, processor.processPayment("u2", 100.0), 0.01);  
 }  
}  
  
// Stub used where fee depends on account type. The stub class returns the type set in its constructor.  
  
class AccountTypeProviderStub implements AccountTypeProvider {  
 private final String type;  
 public AccountTypeProviderStub(String type) { this.type = type; }  
 @Override public String getAccountType(String userId) { return type; }  
}

**Service and Stub**

class PaymentProcessor {  
 private final AccountTypeProvider accountTypeProvider;  
 public PaymentProcessor(AccountTypeProvider accountTypeProvider) {  
 this.accountTypeProvider = accountTypeProvider;  
 }  
 public double processPayment(String userId, double amount) {  
 String type = accountTypeProvider.getAccountType(userId);  
 if ("premium".equals(type)) {  
 return amount \* 1.02; // 2% fee  
 } else {  
 return amount \* 1.05; // 5% fee  
 }  
 }  
}  
interface AccountTypeProvider {  
 String getAccountType(String userId);  
}

**Example 3: Item Restocking Logic**

**Task Requirement:**  
Create an InventoryManager that checks if items need to be restocked using the SupplierService. If stock is below threshold, it should request restock. Use a stub for SupplierService to simulate item availability.

**Test Class**

import org.junit.jupiter.api.Test;  
import static org.junit.jupiter.api.Assertions.\*;  
  
class InventoryManagerTest {  
 // Ensures restocking occurs when supply is available  
 @Test  
 void restocksWhenLowAndSupplierHasStock() {  
 SupplierServiceStub stub = new SupplierServiceStub(true);  
 InventoryManager manager = new InventoryManager(stub);  
 boolean restocked = manager.restockIfNeeded("itemX", 2, 5);  
 assertTrue(restocked, "Should restock if below threshold and supplier has stock");  
 }  
  
 // Ensures no restock if supplier is out of stock  
 @Test  
 void doesNotRestockIfSupplierOut() {  
 SupplierServiceStub stub = new SupplierServiceStub(false);  
 InventoryManager manager = new InventoryManager(stub);  
 boolean restocked = manager.restockIfNeeded("itemX", 2, 5);  
 assertFalse(restocked, "Should not restock if supplier has no stock");  
 }  
}  
  
// Stub Explanation: SupplierServiceStub's constructor sets supplied availability. Used directly to inject simulated supplier status.  
  
class SupplierServiceStub implements SupplierService {  
 private final boolean inStock;  
 public SupplierServiceStub(boolean inStock) { this.inStock = inStock; }  
 @Override public boolean hasStock(String item) { return inStock; }  
 @Override public void order(String item, int quantity) { /\* do nothing \*/ }  
}

**Service and Stub**

class InventoryManager {  
 private final SupplierService supplierService;  
 public InventoryManager(SupplierService supplierService) {  
 this.supplierService = supplierService;  
 }  
 // Returns true if restocked  
 public boolean restockIfNeeded(String item, int currentStock, int threshold) {  
 if (currentStock < threshold && supplierService.hasStock(item)) {  
 supplierService.order(item, threshold - currentStock);  
 return true;  
 }  
 return false;  
 }  
}  
interface SupplierService {  
 boolean hasStock(String item);  
 void order(String item, int quantity);  
}

**Example 4: Email Validation Service**

**Task Requirement:**  
Develop an EmailChecker that validates emails and checks if the domain is allowed by consulting an external AllowedDomainsProvider. Test using a stub for domains.

**Test Class**

import org.junit.jupiter.api.Test;  
import static org.junit.jupiter.api.Assertions.\*;  
  
class EmailCheckerTest {  
 // For allowed domain in stub  
 @Test  
 void acceptsAllowedDomain() {  
 AllowedDomainsProviderStub stub = new AllowedDomainsProviderStub(Set.of("example.com"));  
 EmailChecker checker = new EmailChecker(stub);  
 assertTrue(checker.isValid("john@example.com"));  
 }  
  
 // For disallowed domain in stub  
 @Test  
 void rejectsDisallowedDomain() {  
 AllowedDomainsProviderStub stub = new AllowedDomainsProviderStub(Set.of("example.com"));  
 EmailChecker checker = new EmailChecker(stub);  
 assertFalse(checker.isValid("john@blocked.com"));  
 }  
}  
  
// Stub simulates allowed domains, receiving them at construction and making them available to the service.  
  
class AllowedDomainsProviderStub implements AllowedDomainsProvider {  
 private final Set<String> domains;  
 public AllowedDomainsProviderStub(Set<String> domains) { this.domains = domains; }  
 @Override public Set<String> getAllowedDomains() { return domains; }  
}

**Service and Stub**

class EmailChecker {  
 private final AllowedDomainsProvider domainsProvider;  
 public EmailChecker(AllowedDomainsProvider domainsProvider) {  
 this.domainsProvider = domainsProvider;  
 }  
 public boolean isValid(String email) {  
 String[] parts = email.split("@");  
 return parts.length == 2 && domainsProvider.getAllowedDomains().contains(parts[^1\_1]);  
 }  
}  
interface AllowedDomainsProvider {  
 Set<String> getAllowedDomains();  
}

**Example 5: Discount Eligibility Checker**

**Task Requirement:**  
Implement DiscountService to apply discount based on loyalty status from an external dependency. Use a stub to simulate the loyalty status.

**Test Class**

import org.junit.jupiter.api.Test;  
import static org.junit.jupiter.api.Assertions.\*;  
  
class DiscountServiceTest {  
 // With eligible loyalty status in stub  
 @Test  
 void appliesDiscountForLoyalUsers() {  
 LoyaltyStatusProviderStub stub = new LoyaltyStatusProviderStub(true);  
 DiscountService service = new DiscountService(stub);  
 assertEquals(90.0, service.applyDiscount("u1", 100.0), 0.01);  
 }  
  
 // With ineligible status  
 @Test  
 void doesNotApplyDiscountForNonLoyalUsers() {  
 LoyaltyStatusProviderStub stub = new LoyaltyStatusProviderStub(false);  
 DiscountService service = new DiscountService(stub);  
 assertEquals(100.0, service.applyDiscount("u2", 100.0), 0.01);  
 }  
}  
  
// Stub simulates user loyalty, injected at construction. Tests can thus assert discounting logic in isolation.  
  
class LoyaltyStatusProviderStub implements LoyaltyStatusProvider {  
 private final boolean loyal;  
 public LoyaltyStatusProviderStub(boolean loyal) { this.loyal = loyal; }  
 @Override public boolean isLoyal(String userId) { return loyal; }  
}

**Service and Stub**

class DiscountService {  
 private final LoyaltyStatusProvider provider;  
 public DiscountService(LoyaltyStatusProvider provider) {  
 this.provider = provider;  
 }  
 public double applyDiscount(String userId, double amount) {  
 if (provider.isLoyal(userId)) {  
 return amount \* 0.9; // 10% off  
 }  
 return amount;  
 }  
}  
interface LoyaltyStatusProvider {  
 boolean isLoyal(String userId);  
}