# TDD Lab Test - Test Execution Summary

## Complete Test Suite Results

### Overall Summary

- **Total Tests:** 49
- **Passed:** 49 
- **Failed:** 0
- **Execution Time:** < 0.1 seconds

---

## Requirement A - Product Model & Catalog

### RED Phase - Failing Test Example

```
def test_create_product_fails_when_price_missing(self):
    """Test that product creation fails without price."""
    with pytest.raises(ValueError, match="Price is required"):
        Product(sku="SKU001", name="Laptop", price=None)
```

### Test Output (RED)

```
ModuleNotFoundError: No module named 'src.product'
```

### GREEN Phase - Implementation

```
class Product:
    def __init__(self, sku: str, name: str, price: float):
        if sku is None:
            raise ValueError("SKU is required")
        if name is None:
            raise ValueError("Name is required")
        if price is None:
            raise ValueError("Price is required")
        if price < 0:
            raise ValueError("Price must be non-negative")
        self.sku = sku
        self.name = name
        self.price = price
```

### Test Output (GREEN)

```
 tests/test_product.py::TestProduct::test_create_product_with_valid_data PASSED
tests/test_product.py::TestProduct::test_create_product_fails_when_price_missing PASSED
tests/test_product.py::TestProduct::test_create_product_fails_when_price_negative PASSED
tests/test_product.py::TestProduct::test_create_product_fails_when_sku_missing PASSED
tests/test_product.py::TestProduct::test_create_product_fails_when_name_missing PASSED
tests/test_product.py::TestCatalog::test_catalog_add_product PASSED
tests/test_product.py::TestCatalog::test_catalog_search_by_sku_returns_product PASSED
tests/test_product.py::TestCatalog::test_catalog_search_missing_sku_returns_none PASSED
tests/test_product.py::TestCatalog::test_catalog_can_add_multiple_products PASSED

============================== 9 passed in 0.01s ===============================
```

### REFACTOR

- Extracted Product as a value object
- Used **eq** for product comparison
- Catalog uses dictionary for O(1) lookups

## Requirement B - Shopping Cart

### RED Phase - Failing Test Example

```
def test_add_item_not_in_catalog_raises_error(self):
    """Test that adding a product not in catalog raises an error."""
    cart = Cart(self.catalog)
    with pytest.raises(ValueError, match="Product .* not found in catalog"):
        cart.add_item("INVALID_SKU", 1)
```

### Test Output (RED)

```
ModuleNotFoundError: No module named 'src.cart'
```

### GREEN Phase - Implementation

```
class Cart:
    def add_item(self, sku: str, quantity: int) -> None:
        if quantity <= 0:
            raise ValueError("Quantity must be greater than 0")

        product = self._catalog.get_product_by_sku(sku)
        if product is None:
            raise ValueError(f"Product {sku} not found in catalog")

        if sku in self._items:
            self._items[sku].quantity += quantity
        else:
            self._items[sku] = LineItem(sku, quantity, product.price)
```

### Test Output (GREEN)

```
tests/test_cart.py::TestCart::test_cart_starts_empty PASSED
tests/test_cart.py::TestCart::test_add_item_to_cart PASSED
tests/test_cart.py::TestCart::test_add_item_with_quantity PASSED
tests/test_cart.py::TestCart::test_add_multiple_different_items PASSED
tests/test_cart.py::TestCart::test_add_item_not_in_catalog_raises_error PASSED
tests/test_cart.py::TestCart::test_add_item_with_zero_quantity_raises_error PASSED
tests/test_cart.py::TestCart::test_add_item_with_negative_quantity_raises_error PASSED
tests/test_cart.py::TestCart::test_remove_item_from_cart PASSED
tests/test_cart.py::TestCart::test_remove_item_not_in_cart_raises_error PASSED
tests/test_cart.py::TestCart::test_cart_total_calculation_with_multiple_items PASSED
tests/test_cart.py::TestCart::test_add_same_item_twice_increases_quantity PASSED

============================== 11 passed in 0.02s ==============================
```

### REFACTOR

- Extracted LineItem class for better separation
- Added get_subtotal() method to LineItem
- Cart delegates to catalog for validation

---

## Requirement C - Inventory Reservation

### RED Phase - Failing Test Example

```
def test_add_item_with_insufficient_inventory_raises_error(self):
    """Test that adding more than available inventory fails."""
    self.inventory_service.get_available.return_value = 3
    cart = Cart(self.catalog, self.inventory_service)
    with pytest.raises(ValueError, match="Insufficient inventory.*only 3 available"):
        cart.add_item("SKU001", 5)
```

### Test Output (RED)

```
ModuleNotFoundError: No module named 'src.inventory'
```

### GREEN Phase - Implementation

```
class Cart:
    def add_item(self, sku: str, quantity: int) -> None:
        # ... validation ...

        # Check inventory if service is available
        if self._inventory_service:
            current_quantity = self._items[sku].quantity if sku in self._items else 0
            total_quantity = current_quantity + quantity
            available = self._inventory_service.get_available(sku)

            if total_quantity > available:
                raise ValueError(
                    f"Insufficient inventory for {sku}. "
                    f"Requested {total_quantity}, only {available} available"
                )
```

### Test Output (GREEN)

```
 tests/test_inventory.py::TestInventoryReservation::test_add_item_with_sufficient_inventory PASSED
tests/test_inventory.py::TestInventoryReservation::test_add_item_with_insufficient_inventory_raises_error PASSED
tests/test_inventory.py::TestInventoryReservation::test_add_item_with_exact_inventory_succeeds PASSED
tests/test_inventory.py::TestInventoryReservation::test_add_item_with_zero_inventory_raises_error PASSED
tests/test_inventory.py::TestInventoryReservation::test_add_multiple_items_checks_inventory_for_each PASSED
tests/test_inventory.py::TestInventoryReservation::test_add_same_item_twice_checks_total_quantity PASSED

============================== 6 passed in 0.05s ===============================
```

### REFACTOR

- Created InventoryService interface using ABC
- Used dependency injection in Cart
- Tests use unittest.mock for inventory service

---

## Requirement D - Discount Rules

### RED Phase - Failing Test Example

```
def test_bulk_discount_applies_when_quantity_10_or_more(self):
    """Test bulk discount applies 10% off when quantity >= 10."""
    cart = Cart(self.catalog)
    cart.add_item("SKU002", 10)  # 100 * 10 = 1000

    discount_engine = DiscountEngine()
    discount_engine.add_rule(BulkDiscountRule())

    final_total = discount_engine.apply_discounts(cart)
    assert final_total == 900.0  # 10% off
```

### Test Output (RED)

```
ModuleNotFoundError: No module named 'src.discount'
```

### GREEN Phase - Implementation

```
class BulkDiscountRule(DiscountRule):
    def apply(self, cart: Cart, current_total: float) -> float:
        items = cart.get_items()
        total = 0.0

        for line_item in items.values():
            if line_item.quantity >= 10:
                subtotal = line_item.get_subtotal()
                discounted_subtotal = subtotal * 0.9
                total += discounted_subtotal
            else:
                total += line_item.get_subtotal()

        return total

class OrderDiscountRule(DiscountRule):
    def apply(self, cart: Cart, current_total: float) -> float:
        if current_total >= 1000:
            return current_total * 0.95
        return current_total
```

### Test Output (GREEN)

```
 tests/test_discount.py::TestDiscountRules::test_bulk_discount_applies_when_quantity_10_or_more PASSED
tests/test_discount.py::TestDiscountRules::test_bulk_discount_not_applied_when_quantity_less_than_10 PASSED
tests/test_discount.py::TestDiscountRules::test_bulk_discount_applies_per_line_item PASSED
tests/test_discount.py::TestDiscountRules::test_order_discount_applies_when_total_1000_or_more PASSED
tests/test_discount.py::TestDiscountRules::test_order_discount_not_applied_when_total_less_than_1000 PASSED
tests/test_discount.py::TestDiscountRules::test_multiple_discount_rules_can_be_combined PASSED
tests/test_discount.py::TestDiscountRules::test_discount_engine_with_no_rules_returns_original_total PASSED
tests/test_discount.py::TestDiscountRules::test_bulk_discount_exact_boundary_10_items PASSED
tests/test_discount.py::TestDiscountRules::test_order_discount_exact_boundary_1000 PASSED

============================== 9 passed in 0.02s ===============================
```

### REFACTOR

- Used Strategy Pattern for discount rules
- DiscountEngine orchestrates multiple rules
- Rules are easily pluggable and testable

---

## Requirement E - Checkout Validation & Payment

### RED Phase - Failing Test Example

```
def test_checkout_with_payment_failure_returns_error(self):
    """Test that payment failure prevents order creation."""
    self.inventory_service.get_available.return_value = 10
    self.payment_gateway.charge.return_value = {"success": False, "error": "Card declined"}

    cart = Cart(self.catalog, self.inventory_service)
    cart.add_item("SKU001", 1)

    checkout_service = CheckoutService(self.payment_gateway, self.inventory_service)
    result = checkout_service.checkout(cart, "PAYMENT_TOKEN_123")

    assert result.success is False
    assert "Card declined" in result.error_message
```

### Test Output (RED)

```
ModuleNotFoundError: No module named 'src.checkout'
```

### GREEN Phase - Implementation

```
class CheckoutService:
    def checkout(self, cart: Cart, payment_token: Optional[str]) -> CheckoutResult:
        # Validate cart
        if cart.get_total() == 0:
            return CheckoutResult(success=False, error_message="Cart is empty")

        # Validate payment token
        if not payment_token:
            return CheckoutResult(success=False, error_message="Payment token is required")

        # Validate inventory
        items = cart.get_items()
        for sku, line_item in items.items():
            available = self._inventory_service.get_available(sku)
            if line_item.quantity > available:
                return CheckoutResult(success=False, error_message=f"Insufficient inventory...")

        # Apply discounts
        final_total = self._discount_engine.apply_discounts(cart) if self._discount_engine else cart.get_total()

        # Process payment
        payment_result = self._payment_gateway.charge(final_total, payment_token)

        if not payment_result.get("success"):
            return CheckoutResult(success=False, error_message=payment_result.get("error"))

        return CheckoutResult(success=True, total=final_total, transaction_id=payment_result.get("transaction_id"))
```

### Test Output (GREEN)

```
 tests/test_checkout.py::TestCheckout::test_successful_checkout_with_payment PASSED
tests/test_checkout.py::TestCheckout::test_checkout_with_payment_failure_returns_error PASSED
tests/test_checkout.py::TestCheckout::test_checkout_validates_inventory_before_payment PASSED
tests/test_checkout.py::TestCheckout::test_checkout_applies_discounts_before_payment PASSED
tests/test_checkout.py::TestCheckout::test_checkout_with_empty_cart_fails PASSED
tests/test_checkout.py::TestCheckout::test_checkout_without_payment_token_fails PASSED
tests/test_checkout.py::TestCheckout::test_checkout_result_includes_transaction_id_on_success PASSED

============================== 7 passed in 0.02s ===============================
```

### REFACTOR

- CheckoutService acts as orchestrator
- Payment gateway abstracted as interface
- CheckoutResult dataclass for clean results

---

## Requirement F - Order History & Persistence

### RED Phase - Failing Test Example

```
def test_successful_checkout_creates_order(self):
    """Test that successful checkout creates an order record."""
    self.inventory_service.get_available.return_value = 10
    self.payment_gateway.charge.return_value = {"success": True, "transaction_id": "TXN123"}

    cart = Cart(self.catalog, self.inventory_service)
    cart.add_item("SKU001", 1)

    checkout_service = CheckoutService(
        self.payment_gateway,
        self.inventory_service,
        order_repository=self.order_repository
    )
    result = checkout_service.checkout(cart, "PAYMENT_TOKEN")

    assert result.success is True

    # Verify order was created
    orders = self.order_repository.get_all_orders()
    assert len(orders) == 1
```

### Test Output (RED)

```
ModuleNotFoundError: No module named 'src.order'
```

### GREEN Phase - Implementation

```
@dataclass
class Order:
    order_id: str
    items: List[dict]
    total: float
    transaction_id: str
    timestamp: datetime = field(default_factory=datetime.now)

class InMemoryOrderRepository(OrderRepository):
    def __init__(self):
        self._orders = {}

    def save_order(self, order: Order) -> None:
        self._orders[order.order_id] = order

    def get_order_by_id(self, order_id: str) -> Optional[Order]:
        return self._orders.get(order_id)

# In CheckoutService
if self._order_repository:
    order = create_order_from_cart(cart, final_total, transaction_id)
    self._order_repository.save_order(order)
```

### Test Output (GREEN)

```
 tests/test_order.py::TestOrderPersistence::test_successful_checkout_creates_order PASSED
tests/test_order.py::TestOrderPersistence::test_failed_checkout_does_not_create_order PASSED
tests/test_order.py::TestOrderPersistence::test_order_contains_line_items PASSED
tests/test_order.py::TestOrderPersistence::test_order_has_timestamp PASSED
tests/test_order.py::TestOrderPersistence::test_repository_can_retrieve_order_by_id PASSED
tests/test_order.py::TestOrderPersistence::test_repository_returns_none_for_nonexistent_order PASSED
tests/test_order.py::TestOrderPersistence::test_multiple_orders_can_be_stored PASSED

============================== 7 passed in 0.02s ===============================
```

### REFACTOR

- Repository Pattern for persistence abstraction
- Order as dataclass with automatic timestamp
- Factory function to create orders from carts

---

## Final Test Suite Run

```
$ python -m pytest tests/ -v

============================== 49 passed in 0.06s ==============================
```

### Test Breakdown by Module

- **test_product.py**: 9 tests 
- **test_cart.py**: 11 tests 
- **test_inventory.py**: 6 tests 
- **test_discount.py**: 9 tests 
- **test_checkout.py**: 7 tests 
- **test_order.py**: 7 tests 

---

## Key TDD Principles Demonstrated

1. **Write tests first** - All tests written before implementation
2. **Red-Green-Refactor cycle** - Each requirement followed TDD workflow
3. **Small increments** - One feature at a time
4. **Test as documentation** - Test names describe behavior
5. **Mock external dependencies** - Payment gateway, inventory service
6. **Fast feedback** - All tests run in < 0.1 seconds
7. **High confidence** - 100% pass rate ensures correct behavior

---

## Conclusion

This lab successfully demonstrates Test-Driven Development by:

- Writing 49 comprehensive tests
- Following Red-Green-Refactor for each requirement
- Using mocks for external dependencies
- Implementing clean, testable architecture
- Achieving 100% test pass rate

All requirements (A-F) completed with full test coverage.