Practical Work 2: Integration Test

Huynh Vinh, Nam
huynh-vinh.nam@usth.edu.vn

Kieu Quoc, Viet
kieu-quoc.viet@usth.edu.vn

1 InventorySystem Class Specification

The `InventorySystem` class models a system for managing and tracking items in stock. This builds on your solution from Practical Work 1, with the following modifications for integration between systems.

1.1 Attributes

- **Items**: A dictionary to store item data:
  - `Keys`: Unique item identifiers (product codes provided by the user).
  - `Values`: Dictionaries containing detailed information about each item:
    - `name`: The name of the item (string).
    - `quantity`: The current quantity in stock (non-negative) (integer).
    - `price`: The price per item (float).

1.2 Methods

- `create_item(identifier, name, quantity, price)`: Creates a new item to the system.
  - **Parameters**:
    - `identifier`: The product code for the item (string).
    - `name`: The name of the item (string).
    - `quantity`: The initial quantity of the item to add. (integer).
    - `price`: The price per item (float).
- `display()`: Displays the information of all the items in the system (in a formatted way).
- `remove_item(identifier)`: Removes an item from the system.

- `update_item(identifier, **kwargs)`: Updates specific attributes of an existing item.
- `get_item(identifier)`: Retrieves information for a specific item.
- `check_availability(identifier, quantity)`: Checks if requested quantity if available.
  - **Parameters**:
    - `identifier`: The product code for the item (string).
    - `quantity`: The requested quantity (integer).

1.3 Integration with ShoppingCart

The `ShoppingCart` class from Practical Work 1 needs to be modified to work with the `InventorySystem`. Specifically:

- Initialize `ShoppingCart` with a reference to an `InventorySystem` instance
- Modify add_item() to check inventory availability before adding items
- Modify remove_item() to update inventory when items are removed from cart

Example relationship between classes:

```
# Example of how the classes should interact
inventory = \texttt{InventorySystem}()
inventory.create_item("APPLE1", "Apple", 50, 1.25)
inventory.create_item("BANANA1", "Banana", 30, 0.75)

cart = \texttt{ShoppingCart}(inventory, max_quantity=10)
# This should check inventory and reduce inventory quantity
cart.add_item("APPLE1", 5)
```

2 Testing Scenarios

In this section, you can still use the Python unit testing framework like **unittest** or **pytest** for integration test. However, you'll likely need to use mocking libraries (like **unittest.mock** or **pytest-integration**) to simulate the behavior of external systems.

2.1 Adding an Item to Cart Updates Inventory

- **Test Setup**: Initialize both `ShoppingCart` and `InventorySystem` with a sample item.
- **Action**: Add the item to the shopping cart with a specific quantity.
- **Assertion**: Verify that:

- The item is added to the cart with the correct quantity and price.
- The `InventorySystem`'s quantity for the item is reduced by the amount added to the cart.

2.2 Removing an Item from Cart Updates Inventory

- **Test Setup**: Similar to the previous scenario, but start with the item already in the cart.
- **Action**: Remove the item from the shopping cart.
- **Assertion**: Verify that:
  - The item is no longer present in the cart.
  - The `InventorySystem`'s quantity for the item is restored to its original value.

2.3 Out-of-Stock Items Cannot Be Added

- **Test Setup**: Set the quantity of an item in the `InventorySystem` to 0.
- **Action**: Attempt to add the item to the cart.
- **Assertion**: Verify that:
  - The item is not added to the cart.
  - An appropriate error message or exception is raised (e.g., "Item out of stock").

2.4 Partial Quantity Available

- **Test Setup**: Set the quantity of an item in the `InventorySystem` to less than the requested amount.
- **Action**: Attempt to add the item to the cart with a quantity greater than what's available.
- **Assertion**: Verify appropriate handling (either adding only available quantity or rejection).

2.5 Price Changes in Inventory

- **Test Setup**: Add an item to the cart, then change its price in inventory.
- **Action**: Add more of the same item to the cart.
- **Assertion**: Verify consistent pricing behavior (either maintaining original price or updating to new price).

2.6 Checkout Process

- **Test Setup**: Add multiple items to cart from inventory.
- **Action**: Call the checkout() method.
- **Assertion**: Verify that inventory quantities are permanently updated and cart is emptied.

3 System Integration Design

When implementing the integration between `ShoppingCart` and `InventorySystem`, use the following design pattern:

3.1 Component Relationship

The `ShoppingCart` and `InventorySystem` classes should interact as follows:

- `ShoppingCart` contains a reference to an `InventorySystem` instance
- `ShoppingCart` methods check with `InventorySystem` before performing operations
- Operations on `ShoppingCart` trigger corresponding updates in `InventorySystem`

3.2 Class Relationship Diagram

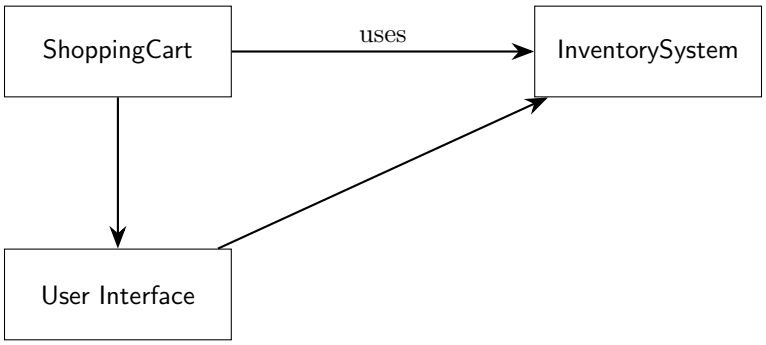

Figure 1: Integration between `ShoppingCart` and `InventorySystem`

3.3 Implementation Guidelines

- The `ShoppingCart` should never modify inventory data directly, but should always call `InventorySystem` methods.
- Both systems should validate data before performing operations.

- Proper error handling should be implemented for cases like out-of-stock items.
- Design for transactional integrity (e.g., if an operation fails halfway, ensure data consistency).