Day 6 Task

Case Scenario: Inventory Management System

Create an Inventory Management System using a class named InventoryItem with attributes like name, price, quantity, and category. Implement functions to add, remove, and update items, view the inventory, and generate a low stock report. Utilize class inheritance and polymorphism, abstract sorting and reporting, encapsulate internal details, and implement exception handling for user input errors. Finally, design a simple command-line interface with options for user interaction.

Breakdown:

1. Add Item:
   a. Allow the user to add a new item to the inventory.
   b. Gather information such as item name, price, quantity, and category.
   c. Validate the input, ensuring that the price and quantity are numeric and greater than zero.
   d. Update the inventory list, set of categories, and dictionary with the new item information.

2. Remove Item:
   a. Provide an option for the user to remove an item from the inventory.
   b. Ask for the item name to be removed.
   c. Check if the item exists in the inventory before removal.
   d. Update the inventory list, set of categories, and dictionary after removing the item.

3. Update Item:
   a. Allow the user to update the details of an existing item.
   b. Ask for the item name to be updated.
   c. Check if the item exists in the inventory before proceeding.
   d. Prompt the user to enter the updated information (price, quantity, or category).

e. Update the inventory list, set of categories, and dictionary with the modified item details.

4. View Inventory:

a. Display the current inventory, showing details like item name, price, quantity, and category.

b. Implement sorting options using lambda functions to sort the inventory list based on different criteria (e.g., name, price, quantity).

5. Low Stock Report:

a. Create a list comprehension to generate a sublist of items that are running low on stock (e.g., quantity less than 10).

b. Display a report of items with low stock, including details like item name, current quantity, and a warning message.

6. Class Implementation:

a. Define a class named `InventoryItem` to represent an item with attributes like name, price, quantity, and category.

b. Implement methods within the class to handle actions like updating the item details and displaying item information.

7. Inheritance:

a. Introduce a subclass for special items (e.g., electronics) that inherits from the `InventoryItem` class.

b. Extend the subclass with additional attributes or methods specific to these special items.

8. Polymorphism:

a. Demonstrate polymorphism by creating multiple instances of the `InventoryItem` class and its subclass, each responding to common methods in a way relevant to its type.

9. Abstraction:

a. Abstract away the details of sorting and low stock reporting into separate functions, keeping the main program clean and modular.

10. Encapsulation:

a. Encapsulate the internal details of the `InventoryItem` class, allowing access and modification through well-defined methods while hiding the implementation details.