Jason Walker

Jason.walker@blackpool.ac.uk

App structure

Warehouse Application

Contents

[InventoryManagement.py 2](#_Toc182579627)

[Step 1: Define the InventoryManager Class 2](#_Toc182579628)

[Step 2: Add a Section 2](#_Toc182579629)

[Step 3: Retrieve a Section 2](#_Toc182579630)

[Step 4: Add an Item to a Section 2](#_Toc182579631)

[Step 5: Manage Stock 3](#_Toc182579632)

[**Add Stock** 3](#_Toc182579633)

[**Remove Stock** 4](#_Toc182579634)

[**Move Stock** 4](#_Toc182579635)

[Step 6: Get Inventory Overview 4](#_Toc182579636)

[Key Points to Remember 5](#_Toc182579637)

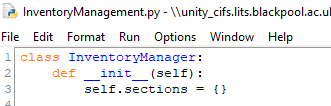
[Integration 5](#_Toc182579638)

# InventoryManagement.py

This document provides an in-depth explanation of the InventoryManagement script, focusing on the `InventoryManager` class and its role in managing warehouse sections and inventory operations.

# Step 1: Define the InventoryManager Class

The `InventoryManager` class acts as the central controller for managing sections and inventory operations across the warehouse.



**Explanation**:

1. `class InventoryManager:`: Declares the `InventoryManager` class.

2. `\_\_init\_\_(self)`: Initializes the class with:

- `sections`: A dictionary to store warehouse sections, where keys are section names and values are `InventorySection` objects.

# Step 2: Add a Section

This method allows adding a new section to the warehouse.



Explanation:

1. `add\_section(self, section)`: Adds the provided `section` object to the `sections` dictionary using its name as the key.

# Step 3: Retrieve a Section

This method retrieves a section by its name.

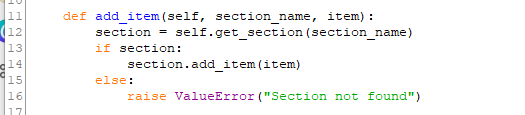


Explanation:

1. `get\_section(self, name)`: Returns the section object if it exists in the `sections` dictionary; otherwise, returns `None`.

# Step 4: Add an Item to a Section

This method adds an inventory item to a specified section.



Explanation:

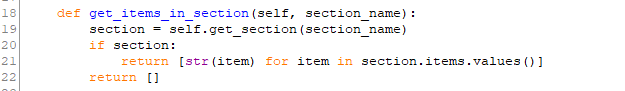
1. Retrieves the section using `get\_section`.

2. If the section exists, calls the `add\_item` method of the section.

3. If the section does not exist, raises a `ValueError`.

# Step 5: Manage Stock

The class includes methods to add, remove, and move stock within the warehouse.

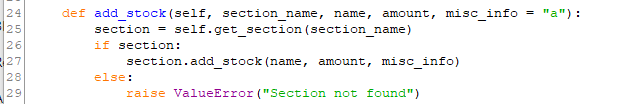
**Get Items in section**  


**Explanation**:

1. Retrieves the section items using `get\_items\_in\_section`.

2. Returns the values of sections selected

### **Add Stock**



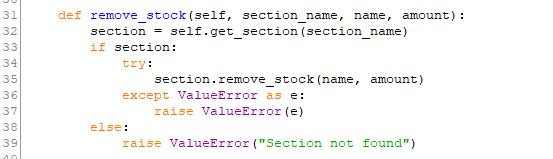
Explanation:

1. Retrieves the section using `get\_section`.

2. Calls the `add\_stock` method of the section to update the item's stock.

3. If the section does not exist, raises a `ValueError`.

### **Remove Stock**



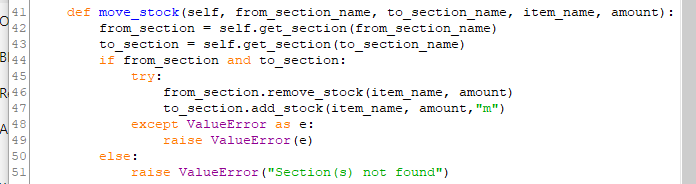
Explanation:

1. Retrieves the section using `get\_section`.

2. Calls the `remove\_stock` method of the section to decrease the item's stock, handling any errors that occur.

3. If the section does not exist, raises a `ValueError`.

### **Move Stock**



Explanation:

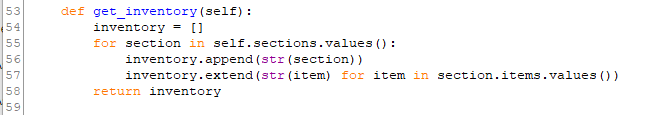
1. Retrieves the source and destination sections using `get\_section`.

2. Attempts to move stock by removing it from the source section and adding it to the destination section.

3. If either section does not exist, raises a `ValueError`.

# Step 6: Get Inventory Overview

This method returns a list of all sections and their inventory items.



Explanation:

1. Iterates through all sections in the `sections` dictionary.

2. Adds the string representation of each section and its items to the `inventory` list.

3. Returns the complete inventory as a list of strings.

# Key Points to Remember

1. The `InventoryManager` class serves as the central hub for managing warehouse sections and their items.

2. It provides methods for adding, retrieving, and managing inventory items and stock.

3. The class integrates closely with `InventorySection`, `RegularItem`, and `PerishableItem` to enable seamless operations.

# Integration

This class can be used in the main application to manage all inventory-related operations. It interacts with the user interface or other components to handle inventory task