Jason Walker

Jason.walker@blackpool.ac.uk

App structure

Warehouse Application

Contents

[Sections.py 2](#_Toc182579058)

[Step 1: Import Dependencies 2](#_Toc182579059)

[Step 2: Define the InventorySection Class 2](#_Toc182579060)

[Step 3: Add an Item to the Section 2](#_Toc182579061)

[Step 4: Retrieve an Item 3](#_Toc182579062)

[Step 5: Add Stock 3](#_Toc182579063)

[Step 6: Remove Stock 3](#_Toc182579064)

[Step 7: Define the String Representation 4](#_Toc182579065)

[Key Points to Remember 4](#_Toc182579066)

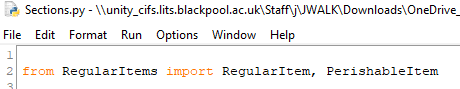
[Integration 4](#_Toc182579067)

# Sections.py

This document provides an in-depth explanation of the Sections script, detailing the functionality of the `InventorySection` class and its interaction with inventory items.

# Step 1: Import Dependencies

The script begins by importing the `RegularItem` and `PerishableItem` classes. These classes represent specific types of inventory items and will be used in the `InventorySection` class.

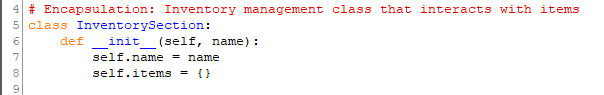


Explanation:

1. `from RegularItems import RegularItem, PerishableItem`: Imports the item classes defined in the RegularItems.py file.

# Step 2: Define the InventorySection Class

The `InventorySection` class manages a collection of inventory items within a specific section of the warehouse.



Explanation:

1. `class InventorySection`: Declares the `InventorySection` class.

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

- `name`: The name of the section.

- `items`: A dictionary to store inventory items, where the keys are item names and the values are item objects.

# Step 3: Add an Item to the Section

This method allows adding an inventory item to the section.



Explanation:

1. `add\_item(self, item)`: Adds the provided `item` object to the `items` dictionary using its name as the key.

# Step 4: Retrieve an Item

This method retrieves an item object by its name.

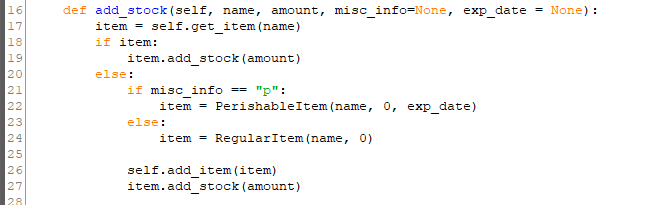


Explanation:

1. `get\_item(self, name)`: Returns the item object if it exists in the `items` dictionary; otherwise, returns `None`.

# Step 5: Add Stock

This method adds stock to an existing item or creates a new item if it does not exist.



Explanation:

1. Checks if the item exists in the section using `get\_item`.

2. If it exists, adds the specified `amount` of stock to the item.

3. If it does not exist, creates a new item:

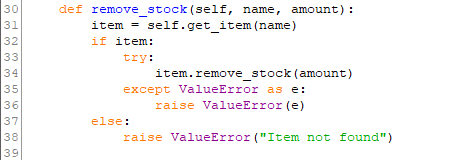
- Creates a `PerishableItem` if `misc\_info` is 'p' and sets its expiry date.

- Otherwise, creates a `RegularItem`.

4. Adds the new item to the section using `add\_item` and updates its stock.

# Step 6: Remove Stock

This method reduces the stock of an existing item. If the item does not exist, it raises an error.



Explanation:

1. Retrieves the item using `get\_item`.

2. If the item exists, attempts to reduce its stock by calling `remove\_stock`.

3. If the item does not exist, raises a `ValueError` with the message 'Item not found'.

# Step 7: Define the String Representation

This method provides a string representation of the section, displaying its name.



Explanation:

1. `\_\_str\_\_(self)`: Returns a string in the format 'Section: SectionName', where `name` is the section's name.

# Key Points to Remember

1. The `InventorySection` class acts as a container for managing inventory items within a specific section.

2. It provides methods to add, retrieve, and manage stock for inventory items.

3. The class integrates seamlessly with `RegularItem` and `PerishableItem` to handle different types of inventory.

# Integration

This class can be used within the inventory management system to represent distinct sections of the warehouse, such as 'Electronics' or 'Perishables'. It interacts with the `InventoryManager` class to enable seamless inventory operations.