Jason Walker

Jason.walker@blackpool.ac.uk

App structure

Warehouse Application

Contents

[Step 1: Set Up the Project Directory 2](#_Toc176942160)

[Step 2: Version Control Repository 3](#_Toc176942161)

[Step 3: Create a New Repository on GitHub 3](#_Toc176942162)

[Step 4: Upload Python Files to the Repository 3](#_Toc176942163)

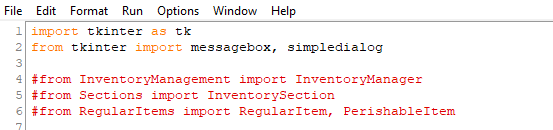
[Step 5: Verify Your Code 3](#_Toc176942164)

# Step 1: Understand the Structure

The project uses Python's Tkinter library to create a GUI-based warehouse management system.

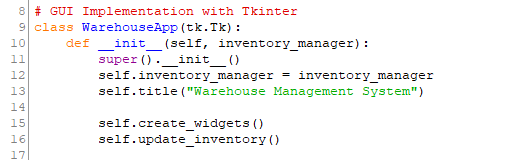
* Import library files
* Create blueprint for GUI with ability to:
  + Add items to the warehouse (with optional expiry for perishable items).
  + Manage stock (add, remove, move).
  + Display the current inventory.
* Call the GUI

# Step 2: Import the files

It’s a good idea to create modular and maintainable programs so we break the structure down.  
  
In Main.py we can import our library files

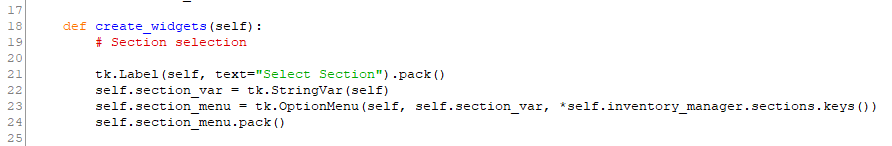
Tkinter is a popular graphical user interface library.  
  
Import tkinter as tk basically imports this library and assigns an alias of tk for short hand.  
  
We are going to import some classes that we create in the future so these are commented out for now.  
  
Line 4 above basically means, Import InventoryManager class from the InventoryManagement.py file.

# Step 3: Class Definition and Initialisation



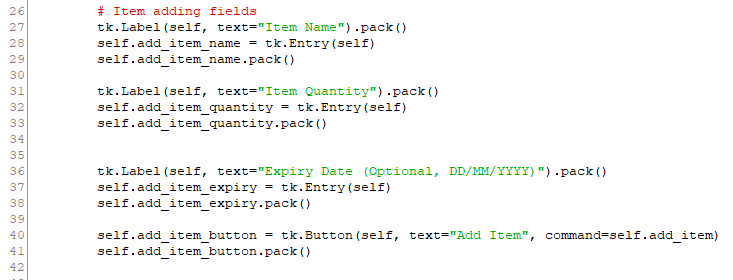
1. WarehouseApp is a subclass of tk.Tk, making it the main window of the application.
2. def \_\_\_init\_\_\_ and super().\_\_init\_\_ function is used to call a method from a parent class. When used with the \_\_init\_\_ method, it allows you to initialise the attributes of the parent class, in addition to any attributes defined in the child class.
3. self.inventory\_manager: This is an instance of InventoryManager that will handle all the inventory operations.
4. self.create\_widgets() initialises all the buttons, labels, and input fields.
5. self.update\_inventory() will refresh the display after every change.

# Step 4: Creating Widgets (UI Components)



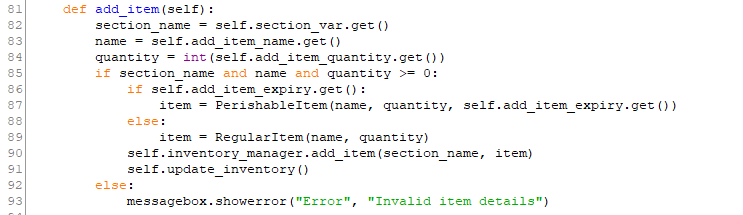
* + def create\_widgets method is part of the class and allows objects initialised to the class to call the method to perform the child actions.
  + This block creates a drop-down menu (Tkinter OptionMenu) for selecting which section of the warehouse you’re interacting with.
  + self.inventory\_manager.sections.keys() gives a list of all sections (e.g., "Electronics", "Automotive").

# Step 5: GUI for Adding Items

****

* + The label fields are created to provide a text label for the entry fields.
  + Entry fields are created to allow the user to input the name, quantity, and optional expiry date for the item.
  + The pack() function is part of tkinter and forces a refresh of the UI with the element included.
  + A button labeled "Add Item" will trigger the self.add\_item() method when clicked.

**Step 6: Method for Adding Items**

Add some space for the other GUI items then create the code for the add\_item method which is triggered when you click the self.add\_item\_button from above.  
  


* + section\_name = self.section\_var.get() retrieves the selected section. Self is the object initialised with the class, section\_var is the variable property that stores the selected value and .get() is a built in function to return the value.
  + If the expiry date is filled in, it creates a Perishable item by calling the method from a different file; otherwise, a RegularItem which are both methods accessed via line 6 import. You may want to comment this line out to avoid getting a defined error and replace it with a temporary print(“You tried to add a perishable/regular item”)
  + The item is added to the specified section in the inventory, and self.update\_inventory() is called to refresh the inventory display.

**Step 4: Managing Stock**

1. **UI for Managing Stock**

python

Copy code

tk.Label(self, text="Stock Amount").pack()

self.stock\_amount = tk.Entry(self)

self.stock\_amount.pack()

self.add\_stock\_button = tk.Button(self, text="Add Stock", command=self.add\_stock)

self.add\_stock\_button.pack()

self.remove\_stock\_button = tk.Button(self, text="Remove Stock", command=self.remove\_stock)

self.remove\_stock\_button.pack()

* + Input fields and buttons are created for adding or removing stock quantities.

1. **Methods for Adding and Removing Stock**

python

Copy code

def add\_stock(self):

section\_name = self.section\_var.get()

name = self.add\_item\_name.get()

amount = int(self.stock\_amount.get())

try:

self.inventory\_manager.add\_stock(section\_name, name, amount)

self.update\_inventory()

except ValueError as e:

messagebox.showerror("Error", str(e))

* + Retrieves the section, item name, and amount to add stock.
  + Calls self.inventory\_manager.add\_stock() to perform the operation and updates the inventory afterward.

python

Copy code

def remove\_stock(self):

section\_name = self.section\_var.get()

name = self.add\_item\_name.get()

amount = int(self.stock\_amount.get())

try:

self.inventory\_manager.remove\_stock(section\_name, name, amount)

self.update\_inventory()

except ValueError as e:

messagebox.showerror("Error", str(e))

* + Similar to adding stock, but for removing stock. It ensures proper error handling with try/except.

**Step 5: Moving Stock Between Sections**

1. **UI for Moving Stock**

python

Copy code

tk.Label(self, text="Destination Section").pack()

self.move\_to\_var = tk.StringVar(self)

self.move\_to\_section = tk.OptionMenu(self, self.move\_to\_var, \*self.inventory\_manager.sections.keys())

self.move\_to\_section.pack()

tk.Label(self, text="What do you want to move?").pack()

self.move\_item\_name = tk.Entry(self)

self.move\_item\_name.pack()

tk.Label(self, text="QTY to move").pack()

self.move\_amount = tk.Entry(self)

self.move\_amount.pack()

self.move\_stock\_button = tk.Button(self, text="Move Stock", command=self.move\_stock)

self.move\_stock\_button.pack()

* + A second section drop-down (self.move\_to\_var) is created for the destination section. This lets the user select where to move stock from one section to another.
  + The user also enters the item name and quantity to move.

1. **Method for Moving Stock**

python

Copy code

def move\_stock(self):

from\_section\_name = self.section\_var.get()

to\_section\_name = self.move\_to\_var.get()

item\_name = self.move\_item\_name.get()

amount = int(self.move\_amount.get())

try:

self.inventory\_manager.move\_stock(from\_section\_name, to\_section\_name, item\_name, amount)

self.update\_inventory()

except ValueError as e:

messagebox.showerror("Error", str(e))

* + Calls the move\_stock() method from the InventoryManager to move items from one section to another.
  + The update\_inventory() method refreshes the inventory list after successful stock movement.

**Step 6: Display Inventory**

1. **UI for Inventory Display**

python

Copy code

self.inventory\_text = tk.Text(self, height=15, width=50)

self.inventory\_text.pack()

* + A large text box is created to display the entire inventory. It will list all items in the current sections.

1. **Method to Update Inventory**