DATA STRUCTURES

PROJECT REPORT

ONLINE SHOPPING CART

Presented to
Sir Abul Salam

Drafted by

- Rida Fatima [BAP-221009]
- Neil Kamal [BAP-221024]

INTRODUCTION

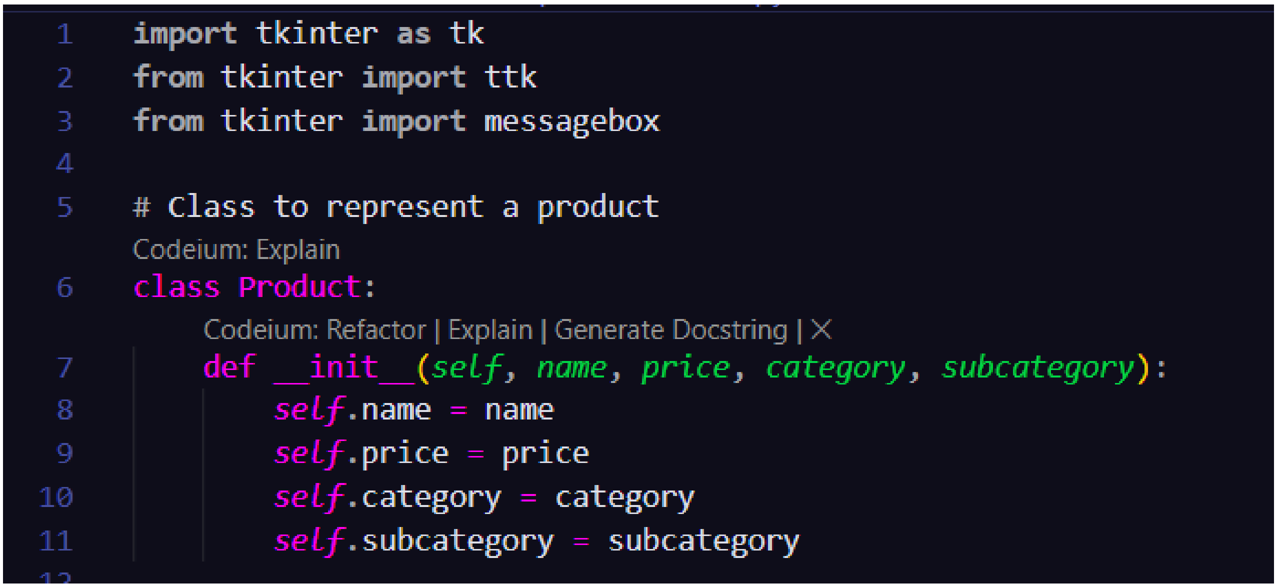
In the modern world of e-commerce and online shopping, the effective organization and management of product catalogs and shopping carts play a pivotal role in providing a seamless and enjoyable user experience. This report showcases the design and implementation of a dynamic shopping cart application, through the effective use of data structures and basic graphical user interface (GUI) implemented through Tkinter.

Project Overview

The project revolves around the creation of a shopping cart system that incorporates a binary search tree (BST) data structure to organize and manage a catalog of products. Each product is represented by a class, and a binary search tree (BST) is employed for efficient insertion, searching, and retrieval of products. The shopping cart itself is implemented using a list to store items, allowing for easy addition and removal of products from the shopping cart.

1. Product Class:

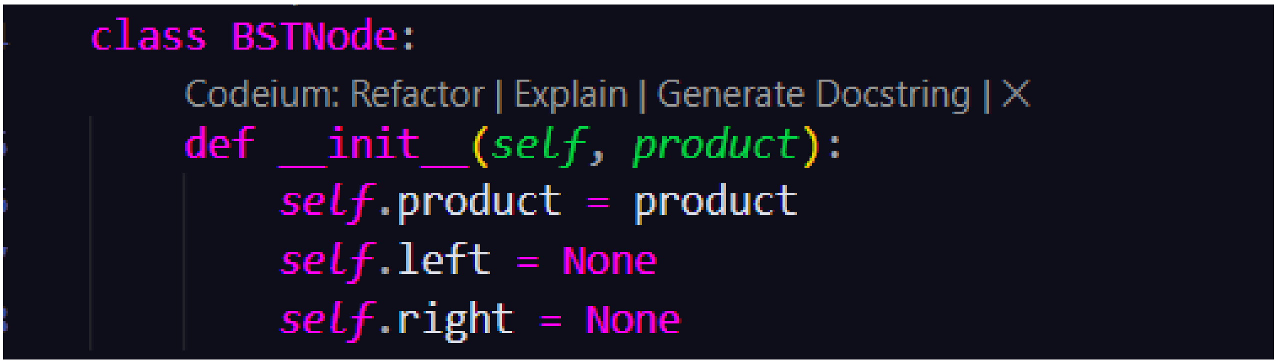
- Functionality: Represents a product with attributes like name, price, category, and subcategory.
- Purpose: To store information about a product.



```python
import tkinter as tk
from tkinter import ttk
from tkinter import messagebox

# Class to represent a product
class Product:
    def __init__(self, name, price, category, subcategory):
        self.name = name
        self.price = price
        self.category = category
        self.subcategory = subcategory
```

2. BSTNode Class:

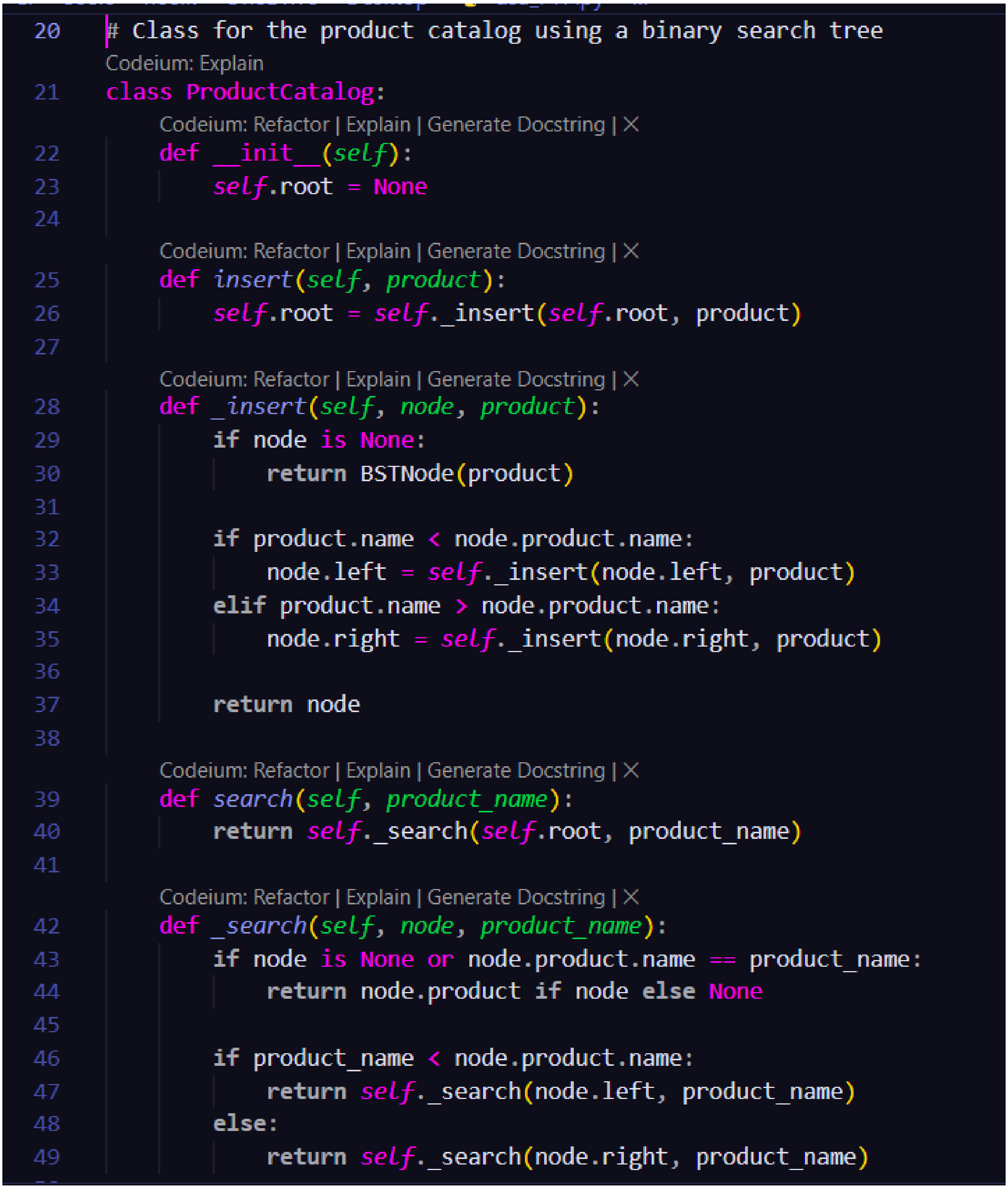
- Functionality: Represents a node in a binary search tree (BST) with a product as its data.
- Data Structure Used: Binary Search Tree (BST) for organizing and searching products efficiently.
- Purpose: Forms the nodes of the binary search tree in the ProductCatalog Class.



```python
class BSTNode:
    def __init__(self, product):
        self.product = product
        self.left = None
        self.right = None
```

3. ProductCatalog Class:

- Functionality: Represents a product catalog using a binary search tree
- Data Structure Used: Binary Search Tree (BST) for efficient insertion and searching of products.
- Purpose: To organize products in a way that allows for efficient searching and insertion.

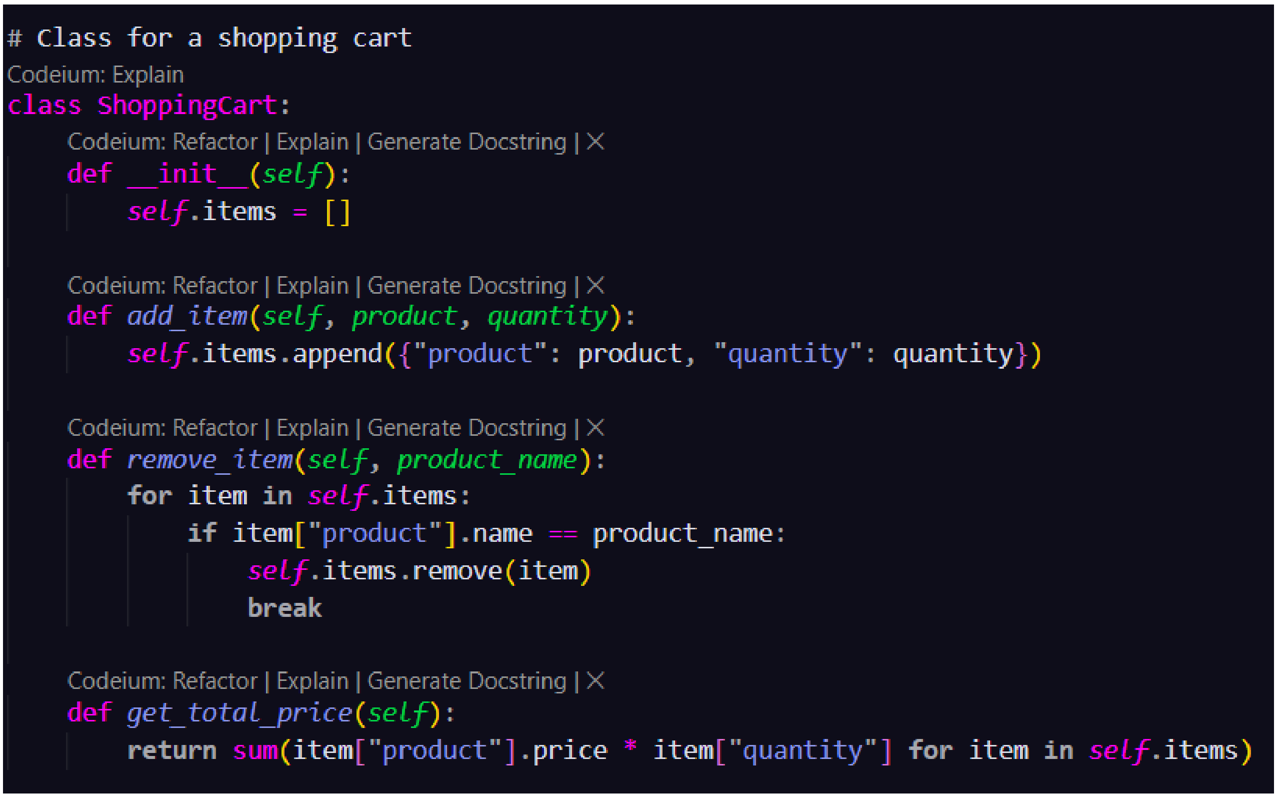


```
# Class for the product catalog using a binary search tree
class ProductCatalog:
    def __init__(self):
        self.root = None

    def insert(self, product):
        self.root = self._insert(self.root, product)

    def _insert(self, node, product):
        if node is None:
            return BSTNode(product)

        if product.name < node.product.name:
            node.left = self._insert(node.left, product)
        elif product.name > node.product.name:
            node.right = self._insert(node.right, product)

        return node

    def search(self, product_name):
        return self._search(self.root, product_name)

    def _search(self, node, product_name):
        if node is None or node.product.name == product_name:
            return node.product if node else None

        if product_name < node.product.name:
            return self._search(node.left, product_name)
        else:
            return self._search(node.right, product_name)
```

Codeium: Explain
Codeium: Refactor | Explain | Generate Docstring | ×
Codeium: Refactor | Explain | Generate Docstring | ×
Codeium: Refactor | Explain | Generate Docstring | ×
Codeium: Refactor | Explain | Generate Docstring | ×
Codeium: Refactor | Explain | Generate Docstring | ×

4. ShoppingCart Class:

- Functionality: Represents a shopping cart that can add and remove items and calculate the total price.
- Data Structure Used: List for storing items in the cart.
- Purpose: To manage the items in the shopping cart.



Codeium: Explain
Codeium: Refactor | Explain | Generate Docstring | ×
Codeium: Refactor | Explain | Generate Docstring | ×
Codeium: Refactor | Explain | Generate Docstring | ×
Codeium: Refactor | Explain | Generate Docstring | ×

```python
# Class for a shopping cart
class ShoppingCart:
    def __init__(self):
        self.items = []

    def add_item(self, product, quantity):
        self.items.append({"product": product, "quantity": quantity})

    def remove_item(self, product_name):
        for item in self.items:
            if item["product"].name == product_name:
                self.items.remove(item)
                break

    def get_total_price(self):
        return sum(item["product"].price * item["quantity"] for item in self.items)
```

5. ShoppingCartGUI Class:

- Functionality: Represents the graphical user interface (GUI) for the shopping cart.
- Data Structure Used: Tkinter widgets for GUI elements. Lists and variables for managing cart items.
- Purpose: To provide an interactive interface for users to add/remove items from the cart.

6. Methods in ShoppingCartGUI Class:

i. add_to_cart Method:

- Functionality: Adds selected products to the cart and updates the GUI.
- Data Structure Usage: Listbox, Combobox, and Labels for displaying and updating cart information.
- Time Complexity: O(n) - Where n is the number of items in the cart (linear time for updating GUI).
- Space Complexity: O(1) - Constant space used for GUI elements.

ii. on_cart_select Method:

- Functionality: Handles the event when an item in the cart is selected and removes it.
- Data Structure Usage: Listbox for displaying cart items.
- Time Complexity: O(n) - Where n is the number of items in the cart (linear time for updating GUI).
- Space Complexity: O(1) - Constant space used for GUI elements.

6. Methods in ShoppingCartGUI Class:

iii. update_product_indices Method:

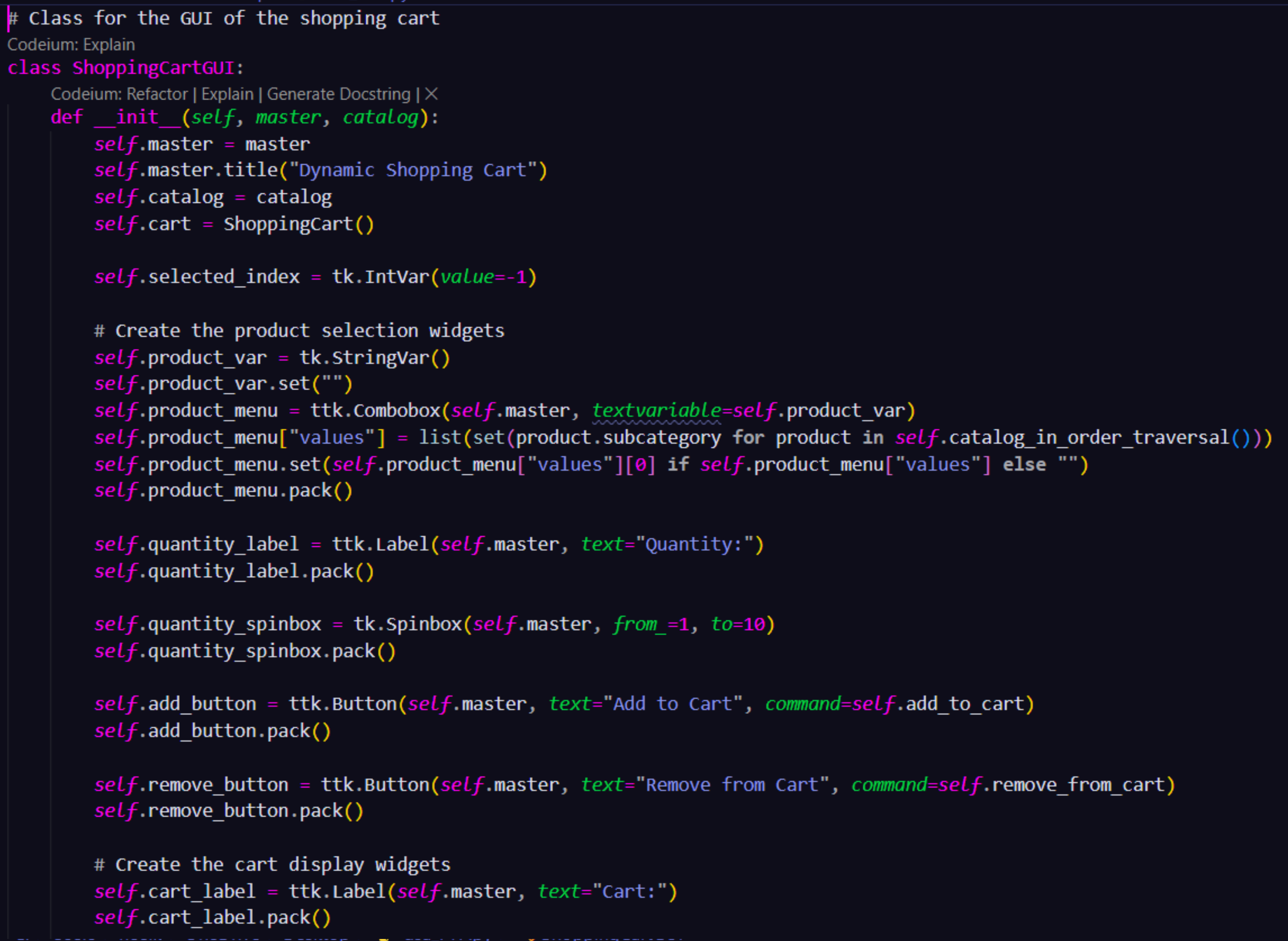
- Functionality: Updates the mapping between product names and their indices in the Listbox.
- Data Structure Usage: Dictionary for storing product indices.
- Time Complexity: O(n) - Where n is the number of items in the cart (linear time for updating indices).
- Space Complexity: O(n) - Linear space for storing product indices.

iv. remove_from_cart Method:

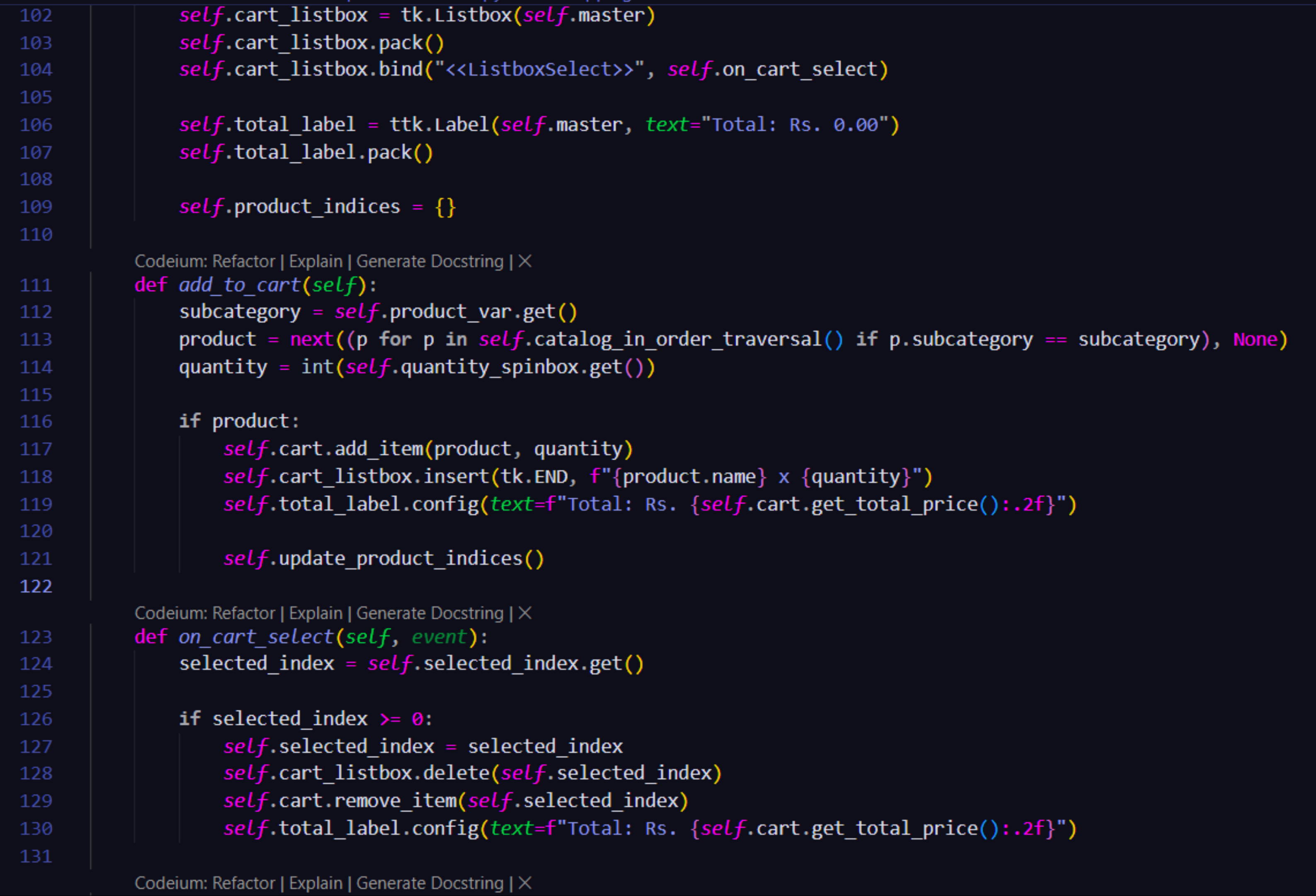
- Functionality: Removes selected items from the cart and updates the GUI.
- Data Structure Usage: Listbox for displaying cart items.
- Time Complexity: O(n) - Where n is the number of items in the cart (linear time for updating GUI).
- Space Complexity: O(1) - Constant space used for GUI elements.

v. catalog_in_order_traversal Method:

- Functionality: Performs an in-order traversal of the product catalog.
- Data Structure Usage: Binary Search Tree (BST) for organizing and searching products.
- Time Complexity: O(m) - Where m is the number of products in the catalog (linear time for traversal).
- Space Complexity: O(1) - Constant space used for traversal.

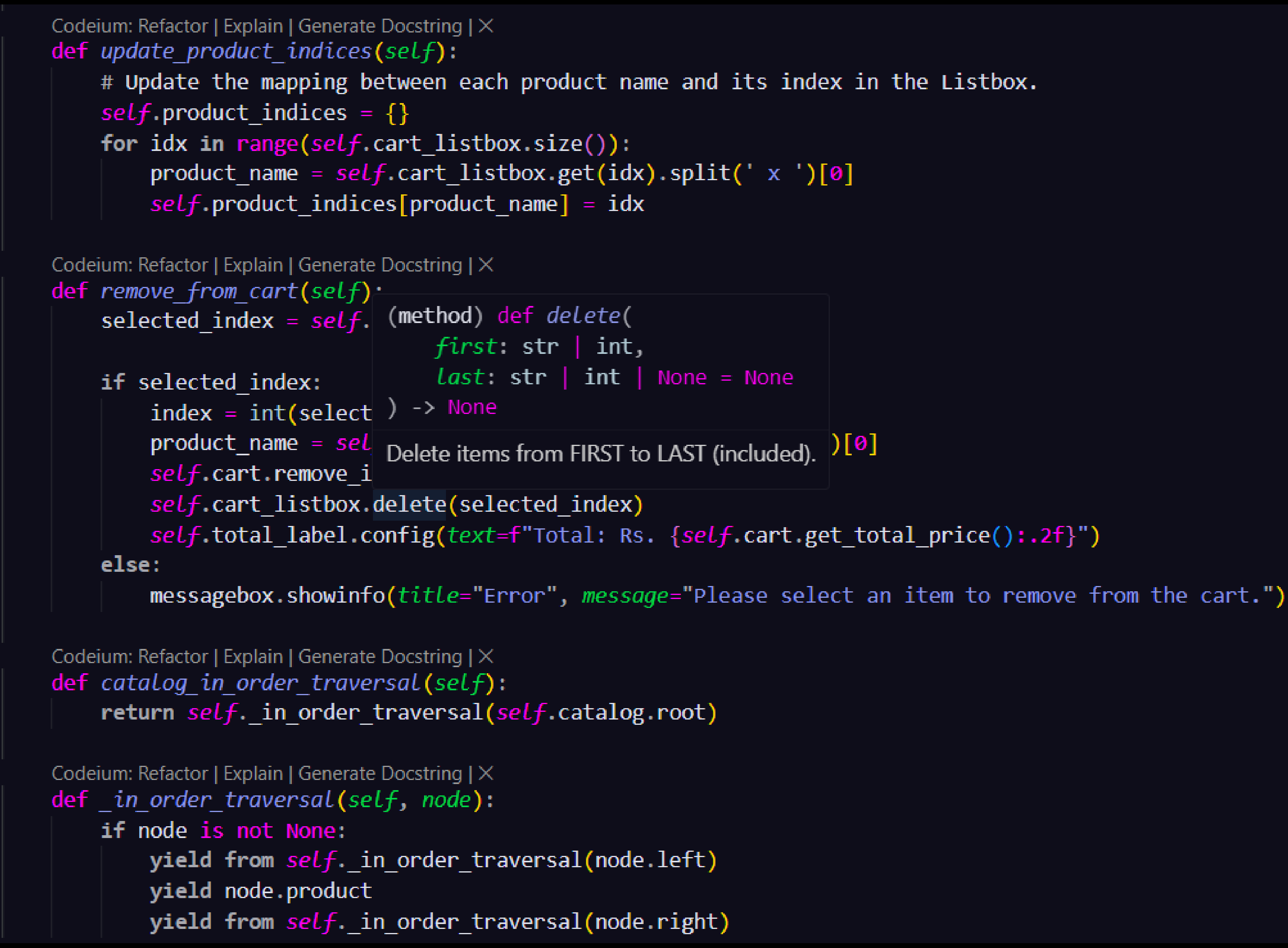


```python
# Class for the GUI of the shopping cart
class ShoppingCartGUI:
    def __init__(self, master, catalog):
        self.master = master
        self.master.title("Dynamic Shopping Cart")
        self.catalog = catalog
        self.cart = ShoppingCart()

        self.selected_index = tk.IntVar(value=-1)

        # Create the product selection widgets
        self.product_var = tk.StringVar()
        self.product_var.set("")
        self.product_menu = ttk.Combobox(self.master, textvariable=self.product_var)
        self.product_menu["values"] = list(set(product.subcategory for product in self.catalog_in_order_traversal()))
        self.product_menu.set(self.product_menu["values"][0] if self.product_menu["values"] else "")
        self.product_menu.pack()

        self.quantity_label = ttk.Label(self.master, text="Quantity:")
        self.quantity_label.pack()

        self.quantity_spinbox = tk.Spinbox(self.master, from_=1, to=10)
        self.quantity_spinbox.pack()

        self.add_button = ttk.Button(self.master, text="Add to Cart", command=self.add_to_cart)
        self.add_button.pack()

        self.remove_button = ttk.Button(self.master, text="Remove from Cart", command=self.remove_from_cart)
        self.remove_button.pack()

        # Create the cart display widgets
        self.cart_label = ttk.Label(self.master, text="Cart:")
        self.cart_label.pack()
```



Codeium: Refactor | Explain | Generate Docstring | ×
Codeium: Refactor | Explain | Generate Docstring | ×
Codeium: Refactor | Explain | Generate Docstring | ×

```python
        self.cart_listbox = tk.Listbox(self.master)
        self.cart_listbox.pack()
        self.cart_listbox.bind("<<ListboxSelect>>", self.on_cart_select)

        self.total_label = ttk.Label(self.master, text="Total: Rs. 0.00")
        self.total_label.pack()

        self.product_indices = {}

    def add_to_cart(self):
        subcategory = self.product_var.get()
        product = next((p for p in self.catalog_in_order_traversal() if p.subcategory == subcategory), None)
        quantity = int(self.quantity_spinbox.get())

        if product:
            self.cart.add_item(product, quantity)
            self.cart_listbox.insert(tk.END, f"{product.name} x {quantity}")
            self.total_label.config(text=f"Total: Rs. {self.cart.get_total_price():.2f}")

            self.update_product_indices()

    def on_cart_select(self, event):
        selected_index = self.selected_index.get()

        if selected_index >= 0:
            self.selected_index = selected_index
            self.cart_listbox.delete(self.selected_index)
            self.cart.remove_item(self.selected_index)
            self.total_label.config(text=f"Total: Rs. {self.cart.get_total_price():.2f}")

```



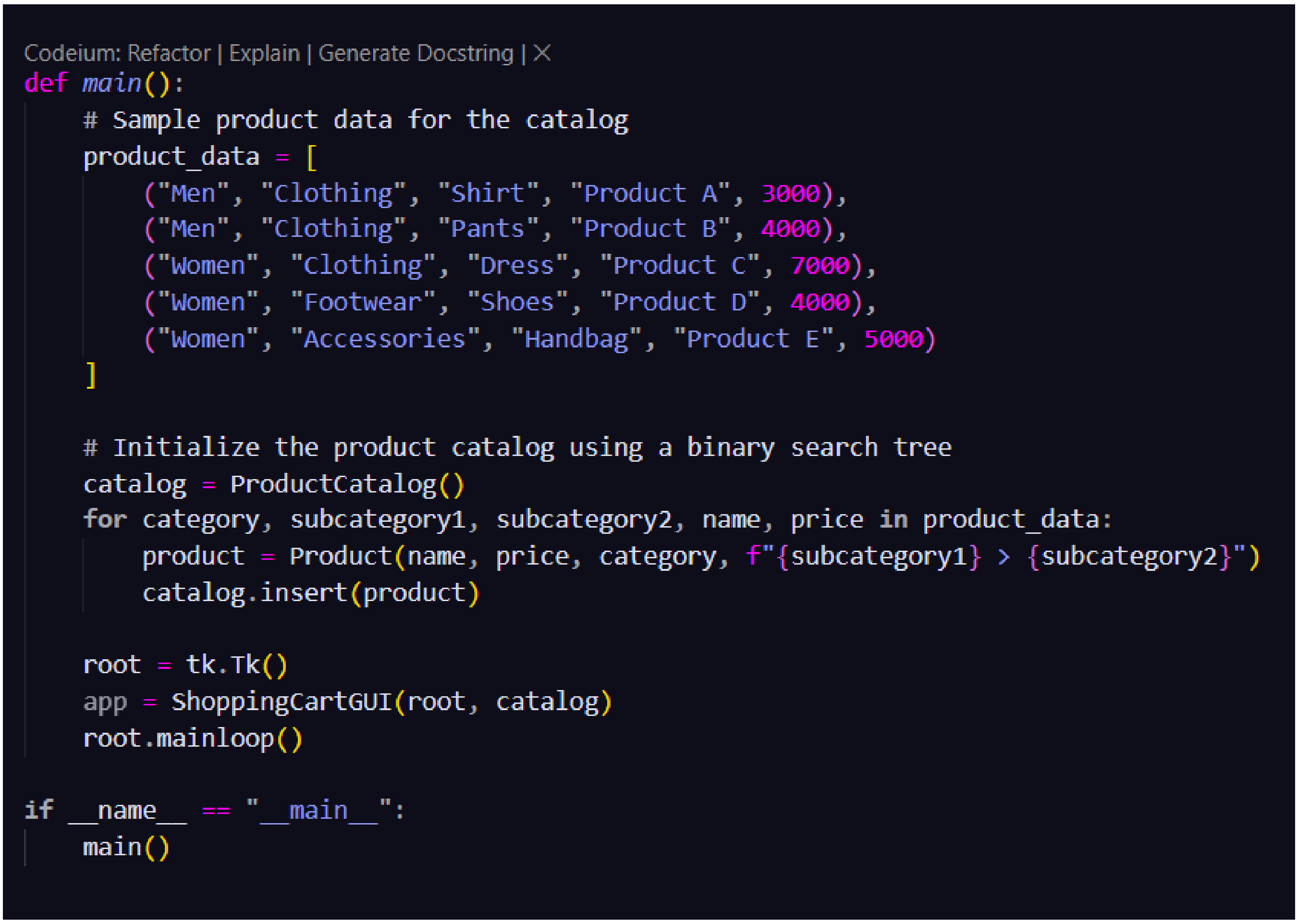
```
Codeium: Refactor | Explain | Generate Docstring | ×
def update_product_indices(self):
    # Update the mapping between each product name and its index in the Listbox.
    self.product_indices = {}
    for idx in range(self.cart_listbox.size()):
        product_name = self.cart_listbox.get(idx).split(' x ')[0]
        self.product_indices[product_name] = idx

Codeium: Refactor | Explain | Generate Docstring | ×
def remove_from_cart(self):
    selected_index = self.

    if selected_index:
        index = int(select
        product_name = sel                                          )[0]
        self.cart.remove_i
        self.cart_listbox.delete(selected_index)
        self.total_label.config(text=f"Total: Rs. {self.cart.get_total_price():.2f}")
    else:
        messagebox.showinfo(title="Error", message="Please select an item to remove from the cart.")

Codeium: Refactor | Explain | Generate Docstring | ×
def catalog_in_order_traversal(self):
    return self._in_order_traversal(self.catalog.root)

Codeium: Refactor | Explain | Generate Docstring | ×
def _in_order_traversal(self, node):
    if node is not None:
        yield from self._in_order_traversal(node.left)
        yield node.product
        yield from self._in_order_traversal(node.right)
```

```
(method) def delete(
    first: str | int,
    last: str | int | None = None
) -> None
```

Delete items from FIRST to LAST (included).

7. Main Function:

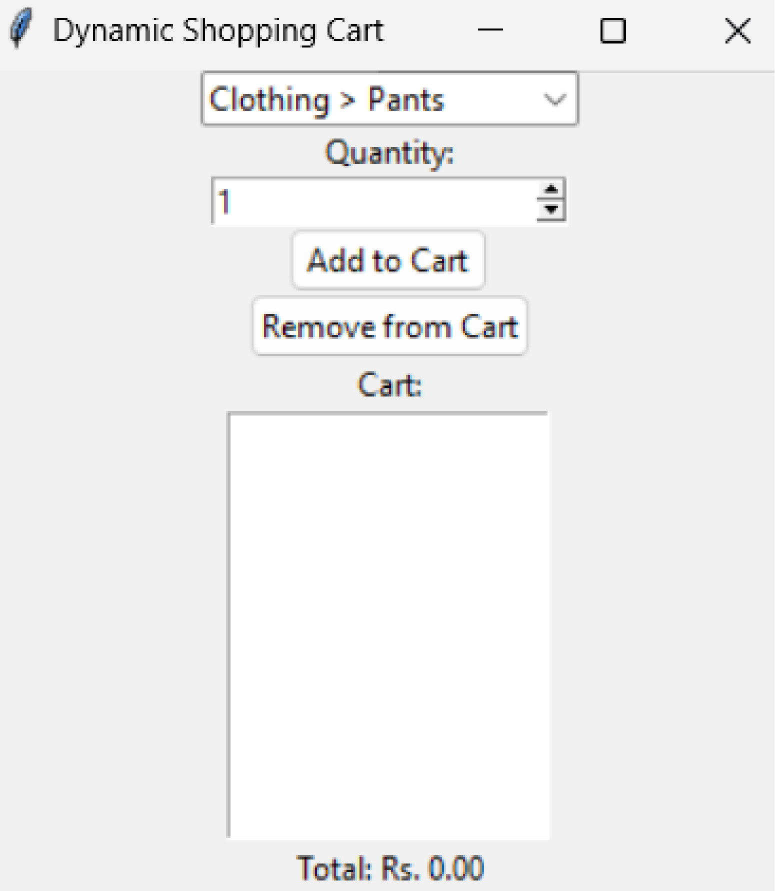
- Functionality: Initializes sample product data and runs the Tkinter main loop.
- Data Structure Usage: None directly, initializes instances of other classes.
- Purpose: To demonstrate the use of the defined classes in creating a shopping cart application.



```python
def main():
    # Sample product data for the catalog
    product_data = [
        ("Men", "Clothing", "Shirt", "Product A", 3000),
        ("Men", "Clothing", "Pants", "Product B", 4000),
        ("Women", "Clothing", "Dress", "Product C", 7000),
        ("Women", "Footwear", "Shoes", "Product D", 4000),
        ("Women", "Accessories", "Handbag", "Product E", 5000)
    ]

    # Initialize the product catalog using a binary search tree
    catalog = ProductCatalog()
    for category, subcategory1, subcategory2, name, price in product_data:
        product = Product(name, price, category, f"{subcategory1} > {subcategory2}")
        catalog.insert(product)

    root = tk.Tk()
    app = ShoppingCartGUI(root, catalog)
    root.mainloop()

if __name__ == "__main__":
    main()
```

Concluding Statement:

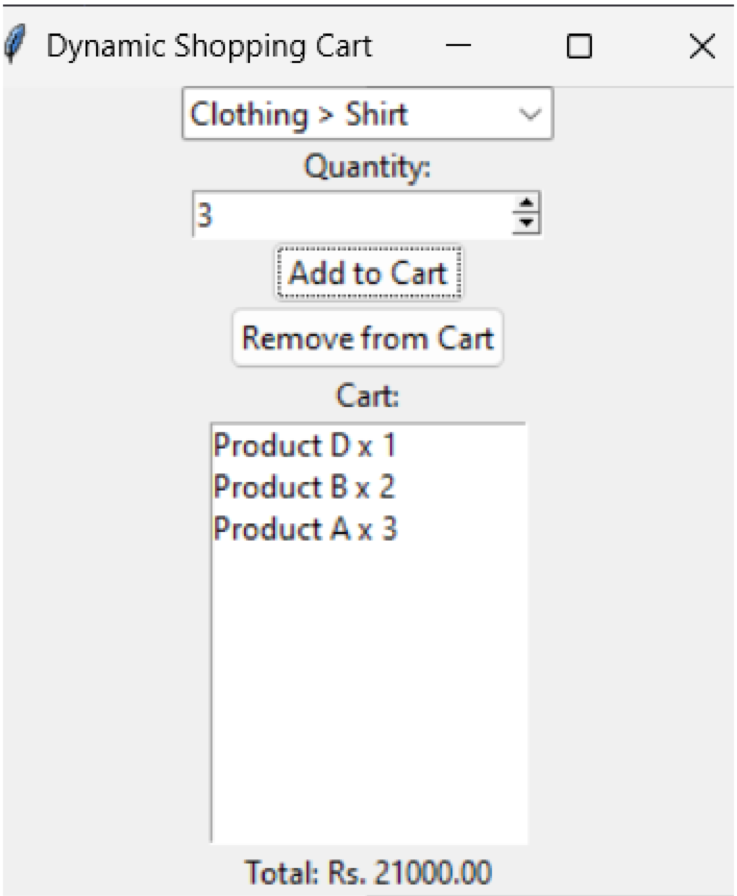
- This project aims to showcase the implementation of a shopping cart using a binary search tree for product organization, linked with a graphical user interface built using the Tkinter library. The choice of data structures reflects the need for efficient product searching and GUI interaction. While the current implementation is functional, further optimizations could be explored based on specific use cases and requirements.



Clothing > Pants

Quantity: 1

Total: Rs. 0.00



Clothing > Shirt

Quantity: 3

Product D x 1
Product B x 2
Product A x 3

Total: Rs. 21000.00

-----THANKYOU-----