GROCERY SHOP MANAGEMENT SYSTEM

Code:

```
import java.io.*;
import java.util.ArrayList;
import java.util.Scanner;

// Fundamental Product Class
class Product {
    private int id;
    private String name;
    private double price;
    private int stock;

    public Product(int id, String name, double price, int stock) {
        this.id = id;
        this.name = name;
        this.price = price;
        this.stock = stock;
    }

    public int getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    public double getPrice() {
        return price;
    }

    public int getStock() {
        return stock;
    }

    public void setStock(int stock) {
        this.stock = stock;
    }
    @Override
    public String toString() {
        return String.format("ID: %d | Name: %s | Price: $%.2f | Stock: %d", id, name,
price, stock);
    }
}

// Inheritance, Interfaces & Packages
interface Billable {
    double calculateTotal();
}

class CartItem extends Product implements Billable {
    private int quantity;

    public CartItem(int id, String name, double price, int stock, int quantity) {
        super(id, name, price, stock);
        this.quantity = quantity;
    }

    public int getQuantity() {
        return quantity;
    }

    @Override
    public double calculateTotal() {
        return getQuantity() * getPrice();
    }

    @Override
    public String toString() {
        return super.toString() + String.format(" | Quantity: %d | Total: $%.2f", quantity,
calculateTotal());
    }
}

// Exceptions Handling
class InsufficientStockException extends Exception {
    public InsufficientStockException(String message) {
        super(message);
    }
}

// Store Class with Collections, File Handling, and Multithreading
```

```
class Store {
   private ArrayList<Product> productList;
   private ArrayList<CartItem> cart;

   public Store() {
      productList = new ArrayList<>();
      cart = new ArrayList<>();
      loadProducts(); // Load products from file
   }

   // Add new product to store
   public synchronized void addProduct(Product product) {
      productList.add(product);
   }

   // Display all products
   public void displayProducts() {
      for (Product product : productList) {
         System.out.println(product);
      }
   }

   // Search product by ID
   public Product searchProductById(int id) {
      for (Product product : productList) {
         if (product.getId() == id) {
            return product;
         }
      }
      return null;
   }

   // Add product to cart
   public void addToCart(int productId, int quantity) throws
InsufficientStockException {
      Product product = searchProductById(productId);
      if (product != null) {
         if (product.getStock() >= quantity) {
            CartItem cartItem = new CartItem(product.getId(), product.getName(),
product.getPrice(), product.getStock(), quantity);
            cart.add(cartItem);
            product.setStock(product.getStock() - quantity);
            System.out.println("Added to cart: " + cartItem);
         } else {
                throw new InsufficientStockException("Not enough stock available.");
            }
        } else {
            System.out.println("Product not found!");
        }
    }

    // Display Cart Items
    public void displayCart() {
        if (cart.isEmpty()) {
            System.out.println("Cart is empty.");
        } else {
            for (CartItem item : cart) {
                System.out.println(item);
            }
        }
    }

    // Save products to file
    public void saveProducts() {
        try (FileWriter fw = new FileWriter("products.txt", false);
             BufferedWriter bw = new BufferedWriter(fw)) {
            for (Product product : productList) {
                bw.write(product.getId() + "," + product.getName() + "," + product.getPrice()
+ "," + product.getStock() + "\n");
            }
            System.out.println("Products saved successfully.");
        } catch (IOException e) {
            System.out.println("Error saving products: " + e.getMessage());
        }
    }

    // Load products from file
    public void loadProducts() {
        try (BufferedReader br = new BufferedReader(new FileReader("products.txt")))
{
            String line;
            while ((line = br.readLine()) != null) {
                String[] data = line.split(",");
                int id = Integer.parseInt(data[0]);
                String name = data[1];
                double price = Double.parseDouble(data[2]);
                int stock = Integer.parseInt(data[3]);
                addProduct(new Product(id, name, price, stock));
            }
        } catch (FileNotFoundException e) {
            System.out.println("Product file not found. Starting with an empty store.");
        } catch (IOException e) {
            System.out.println("Error loading products: " + e.getMessage());
        }
    }

    // Thread to simulate adding new products
    public void simulateAddingProducts() {
        Runnable addTask = () -> {
            try {
                Thread.sleep(2000);
                addProduct(new Product(101, "Banana", 0.99, 100));
                System.out.println("New product added to the store!");
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        };
        new Thread(addTask).start();
    }
}

// Main Class for User Interaction
public class GroceryStore {
    public static void main(String[] args) {
        Store store = new Store();
        Scanner scanner = new Scanner(System.in);
        int choice;

        store.simulateAddingProducts(); // Start background thread

        do {
            System.out.println("\n===== Grocery Store Menu =====");
            System.out.println("1. Display Products");
            System.out.println("2. Add Product to Cart");
            System.out.println("3. View Cart");
            System.out.println("4. Add New Product to Store");
            System.out.println("5. Save Products to File");
            System.out.println("6. Exit");
            System.out.print("Enter your choice: ");
            choice = scanner.nextInt();

            switch (choice) {
```

```
case 1:
    store.displayProducts();
    break;

case 2:
    System.out.print("Enter product ID to add to cart: ");
    int id = scanner.nextInt();
    System.out.print("Enter quantity: ");
    int quantity = scanner.nextInt();

    try {
        store.addToCart(id, quantity);
    } catch (InsufficientStockException e) {
        System.out.println("Error: " + e.getMessage());
    }
    break;

case 3:
    store.displayCart();
    break;

case 4:
    System.out.print("Enter new product ID: ");
    int newId = scanner.nextInt();
    scanner.nextLine(); // consume the newline character
    System.out.print("Enter new product name: ");
    String name = scanner.nextLine();
    System.out.print("Enter new product price: ");
    double price = scanner.nextDouble();
    System.out.print("Enter new product stock: ");
    int stock = scanner.nextInt();

    Product newProduct = new Product(newId, name, price, stock);
    store.addProduct(newProduct);
    System.out.println("New product added: " + newProduct);
    break;

case 5:
    store.saveProducts();
    break;

case 6:
    System.out.println("Exiting the store. Goodbye!");
    break;
```

```
            default:
                System.out.println("Invalid choice! Please try again.");
        }
    } while (choice != 6);

    scanner.close();
  }
}
```

Concepts:

This Java program utilizes several key programming concepts and features, which are outlined below:

1. Object-Oriented Programming (OOP) Concepts:

- Classes and Objects:
  - Product, CartItem, Store, and GroceryStore are defined as separate classes to encapsulate related data and behaviors.
- Inheritance:
  - CartItem inherits from the Product class, allowing it to reuse the properties and methods defined in Product.
- Polymorphism:
  - CartItem overrides the toString() method from Product to customize its string representation.
- Encapsulation:
  - Class fields like id, name, price, and stock are marked as private and are accessed through public getters and setters to control visibility and maintain data integrity.
- Interfaces:
  - The Billable interface is used to define the calculateTotal() method, which is then implemented in the CartItem class, demonstrating polymorphic behavior.

2. Exception Handling:

- A custom exception class, InsufficientStockException, is created to handle cases where the requested quantity of a product exceeds its available stock.
- try-catch blocks are used to manage exceptions when adding items to the cart, ensuring that invalid operations are handled gracefully.

3. File Handling:

- The program uses FileReader, BufferedReader, FileWriter, and BufferedWriter to read from and write to a text file (products.txt), demonstrating basic file I/O operations.
- This feature enables persistence, allowing the program to load products from a file when the store is initialized and save changes to the file before exiting.

4. Collections Framework:

- The ArrayList class is used to store collections of products (productList) and cart items (cart).
- It provides dynamic resizing and various utility methods for managing and manipulating collections.

5. Multithreading:

- The simulateAddingProducts() method in the Store class spawns a new thread using Runnable to add a product asynchronously, simulating concurrent operations in the store.
- This showcases basic usage of threads in Java.

6. Synchronization:

- The addProduct() method in the Store class is marked with synchronized to prevent concurrent modifications to the productList when products are being added in a multithreaded environment.

7. User Input Handling:

- The Scanner class is used to read input from the user for selecting options from the menu, adding products to the store, and managing the cart.

8. Looping and Control Structures:

- for loops are used to iterate through the collections when displaying products and cart items.
- A do-while loop is used in the main() method to repeatedly display the menu until the user chooses to exit.

9. String Formatting:

- The String.format() method is used to format strings for displaying product and cart item details in a structured format.

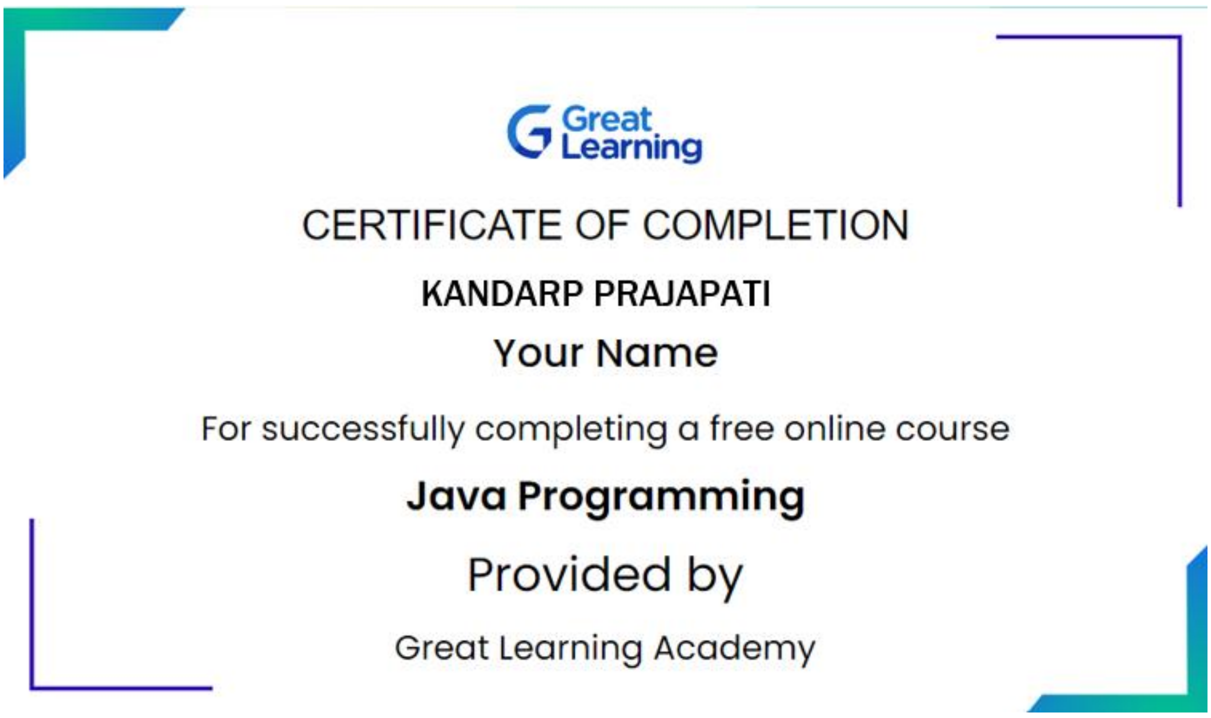
10. Static Typing and Method Overloading:

- The program leverages Java's static typing system to ensure type safety.
- Method overriding is demonstrated in the CartItem class with the toString() and calculateTotal() methods.

These concepts work together to create a well-structured, interactive program that simulates a basic grocery store management system with products and a shopping cart.

BY-23DCS 114,101,103

CERTIFICATE:



Great Learning

CERTIFICATE OF COMPLETION

KANDARP PRAJAPATI

Your Name

For successfully completing a free online course

Java Programming

Provided by

Great Learning Academy