



CSE4006

Object Oriented Programing

Dilshan Gunasekara

CL/HDCSE/CMU/120/05

Assignment Cover Sheet

Qualification		Module Number and Title
HD in Computing/HD in Software Engineering		Object Oriented Programming- CSE4006
Student Name & No.		Assessor
Student Name: Dilshan Gunasekara Student No: CL/HDCSE/CMU/120/05		Ms. Upeka
Hand out date		Submission Date
2024-01-16		2024-03-01
Assessment type WRIT1- Coursework	Duration/Length of Assessment Type 3000 words equivalent	Weighting of Assessment 100%

Learner declaration
I Dilshan Gunasekara CL/HDCSE/CMU/120/05 certify that the work submitted for this assignment is my own and research sources are fully acknowledged.

Marks Awarded			
First assessor			
IV marks			
Agreed grade			
Signature of the assessor		Date	

Feedback Form

International College of Business & Technology

Module: Object Oriented Programming

Student: Dilshan Gunasekara

Assessor: Ms. Upeka

Assignment: Object Oriented Programming

Strong features of your work:

Areas for improvement:

Marks Awarded:

Contents

Acknowledgment	4
Introduction	4
Purpose and Scope.....	5
Task 01.....	6
Use Case Diagram	6
Class Diagram	7
Sequeunce Diagram	8
Cashier Diagram	9
Manager Diagram	9
Manager Account Diagram	10
Task 02.....	11
Object-Oriented Programming Concepts	11
Uses of OOP Concepts	12
Significance	12
Codes Snippets of The Project	13
Task 03.....	16
User Manul for The Little Bag Shop	16
References.....	23

Acknowledgment

I would want to thank Ms. Upeka from ICBT Campus from the bottom of my heart for all of his help and advice in teaching us the fundamentals of object-oriented programming (OOP). I would not be able to comprehend and use these ideas in real-world situations without his understanding, commitment, and patience. The information and abilities he taught in his classes were essential to finishing this job successfully.

I also want to sincerely thank my family and friends for their constant encouragement and support. Their inspiration and understanding have been invaluable in supporting my ability to remain committed and focused during this journey. Their confidence in my skills has always motivated me, and I am very appreciative of their existence in my life.

Introduction

The goal of the assignment "The Little Bag Shop" is to create a fully automated system for a retail establishment that sells only bags. The purpose of this system is to improve customer service and operational efficiency by streamlining the purchase process. It has capabilities that enable managers and cashiers to efficiently handle user accounts and inventories.

Cashiers can see, add, and search for bags based on categories, and managers can create new cashier accounts to provide secure access control. The ideas of Object-Oriented Programming (OOP), which are essential for creating reliable and maintainable software, are incorporated into the system's design. These ideas—abstraction, encapsulation, inheritance, and polymorphism—all support the development of modular and expandable system designs.

All things considered, the "The Little Bag Shop" project shows how to apply OOP concepts to build a productive, user-friendly system that improves both the operational workflow and the shopping experience in a retail setting.

Purpose and Scope

The main goal of this assignment is to create an extensive application that supports the key components of a bag shop's retail management system. The system offers a streamlined and effective method to manage daily operations by managing multiple business components, including user accounts, inventory, and bag data.

Two user roles are distinguished by the application: Cashier and Manager. Basic inventory management duties for cashiers include viewing every bag in store, adding new bags to the stock, and conducting category searches for bags. This makes it possible for them to efficiently help clients and guarantees that they will locate the bags they need, be it backpacks, shoulder bags, tote bags, or any other kind.

In contrast, the responsibilities of managers are more varied. They can handle user accounts in addition to all the duties that cashiers can complete. In order to guarantee that only authorised workers can access particular system capabilities, this includes creating new cashier accounts. The separation of user roles contributes to the upkeep of a system that is orderly, secure, and has well-defined roles and access levels.

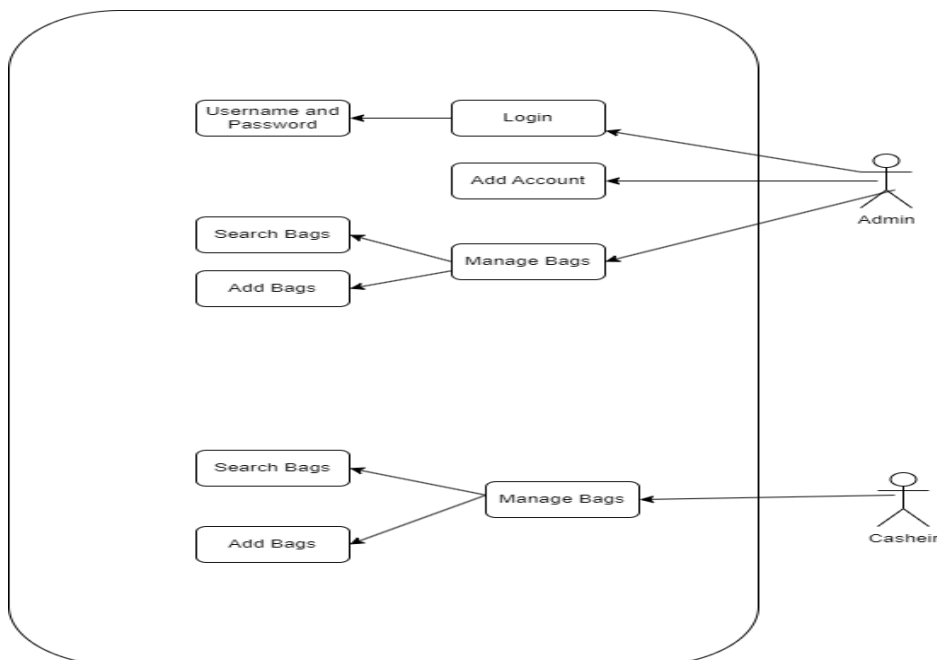
The design of the program makes use of Object-Oriented Programming (OOP) ideas to produce a scalable and modular solution. These ideas include inheritance, which permits the system to handle various user inputs and actions dynamically; abstraction, which streamlines complex operations and presents them in an approachable way; encapsulation, which protects sensitive data and limits access to it; and polymorphism, which allows the system to handle various user inputs and actions dynamically.

This assignment essentially shows how OOP ideas may be used to create a useful and intuitive retail management system. It demonstrates how to design a safe, well-organised application that can be readily expanded and maintained as the company expands. This solution gives "The Little Bag Shop" a more dependable and fluid purchasing experience, which not only improves the operational workflow but also the entire consumer experience.

Task 01

Use Case Diagram

The functional needs of a system are represented by a Use Case Diagram in UML, which shows interactions between the system and its external actors—users, other systems, or hardware—through interactions. The graphic presents a number of use examples that detail particular features or services that the system offers to its users. Important elements include use cases (shown as ovals), actors (stick figures), associations (lines joining actors to use cases), and the system boundary (a rectangle containing all use cases). Use case relationships that exhibit dependence and inheritance include **Include**, **Extend**, and **Generalisation**. Use case diagrams are helpful for defining the project scope, gathering requirements, designing systems, communicating with stakeholders, validating and verifying information, and creating test cases.

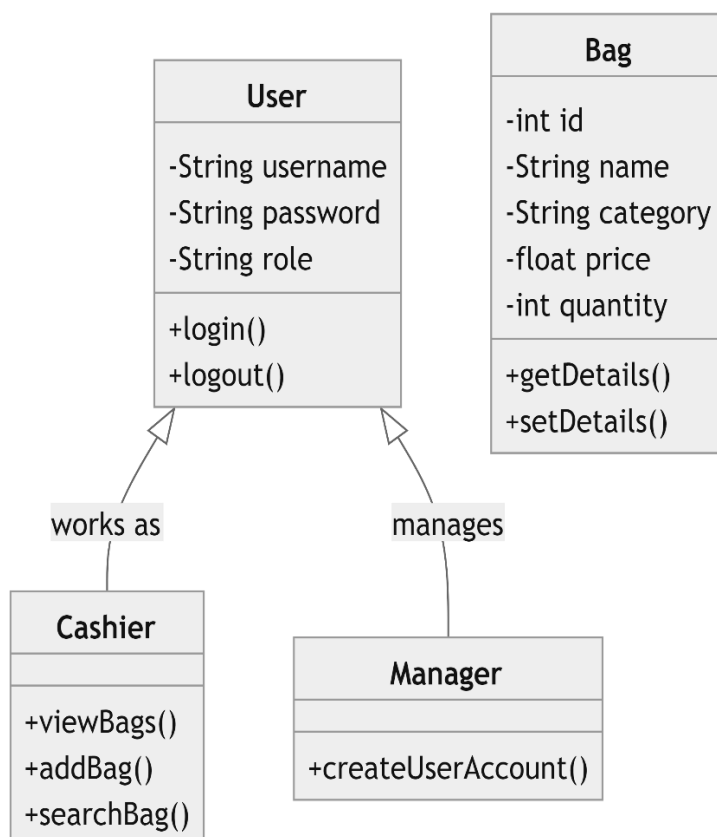


Class Diagram

One essential UML component that shows a system's structural structure is the class diagram. It describes the classes in the system, along with their properties, functions, and connections. A rectangle with the class name at the top, attributes in the middle, and methods at the bottom is used to symbolise each class. A class's properties are defined by its attributes, which also indicate access levels with visibility modifiers like + (public), - (private), and # (protected). Methods are the means by which a class can carry out operations.

Class diagram relationships are represented by the following relationships: aggregation, which indicates a whole-part relationship where parts can exist independently; composition, a stronger form where parts cannot exist without the whole; associations, which show interactions between classes; and inheritance (generalisation), which illustrates a parent-child class relationship.

Class diagrams help clarify the roles and responsibilities of various components in a system, acting as a blueprint for system design and execution. They provide a clear picture of the structure of the system and aid in stakeholder communication, making them crucial to both the analysis and design stages of software development.



Sequence Diagram

UML diagrams called sequence diagrams show how items interact in a specified order to accomplish a given task. Their significance lies in their emphasis on the sequential exchange of messages between objects in time, which makes them indispensable for comprehending the dynamic behaviour of a system. A sequence diagram shows how things work together to accomplish a goal and how events unfold in a system over time.

Key Elements

Actors and Objects: The people involved in the interaction are shown as vertical lines known as lifelines. Objects represent instances of classes within the system, and actors represent users or other systems.

Lifelines: Vertical lines that are dashed and stretch from actors or objects to show their presence over time. A lifeline indicates an object's lifespan while it is being used.

Messages: The interactions or communication between actors and objects is represented by horizontal arrows between lifelines. They show the order in which messages are sent and received, frequently involving data exchanges or method calls.

Activation Bars: The duration of an object's active message processing is shown by thin rectangles on a lifeline. The action's duration is indicated by the bar's length.

Asynchronous and Synchronous Messages: Asynchronous messages are indicated with solid arrows and require a response in order for the interaction to proceed. Asynchronous messages are indicated with open arrows and don't wait for a response.

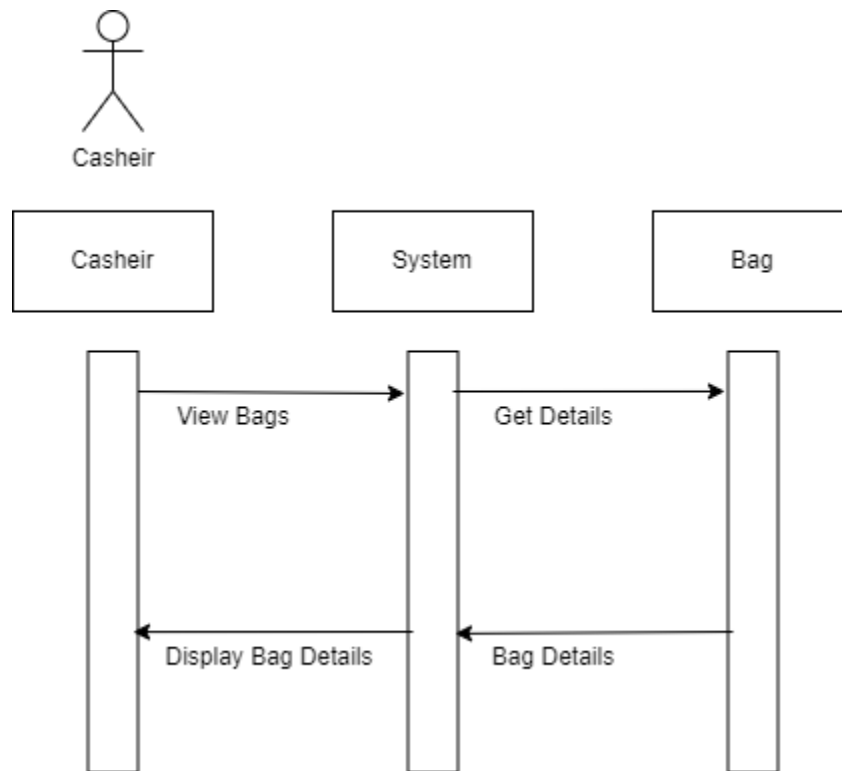
Return Messages: Dashed lines that show when control is transferred from one object to another; they frequently contain method call results.

Use

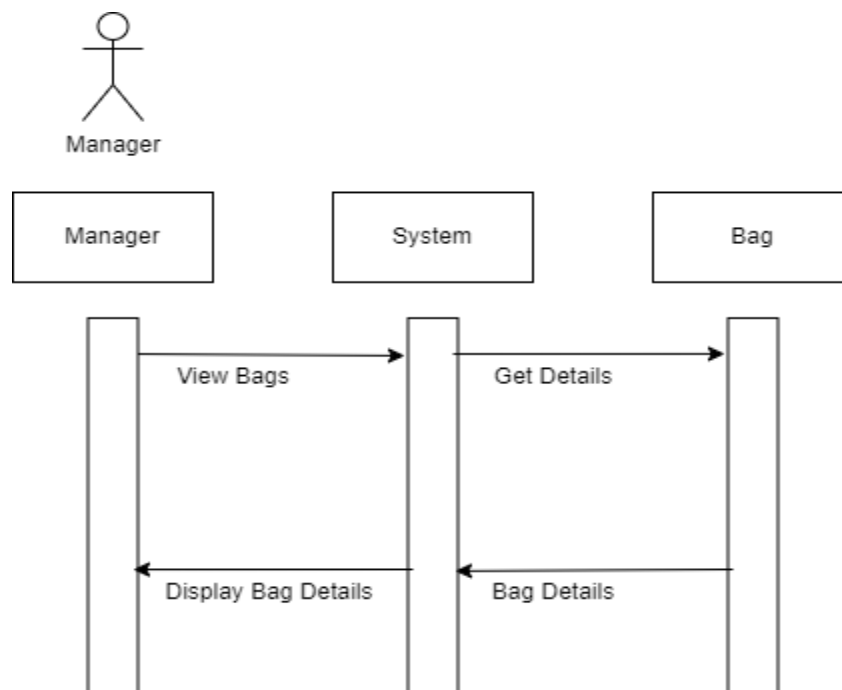
Sequence diagrams are very helpful during the system development process' analysis and design stages. They facilitate the understanding of intricate processes and the detection of possible problems by helping to make the order of actions and interactions clearer. They are especially helpful in figuring out how items work together and making sure that all required interactions are taken into account during the design process.

Sequence diagrams can show steps in systems like "The Little Bag Shop," such as a management registering a new user account or a cashier evaluating bag details. By assisting stakeholders in visualising and verifying the system's flow of operations, these diagrams make that the system satisfies its requirements and operates as intended. Sequence diagrams facilitate improved team communication and system design refinement by offering a distinct temporal sequence of interactions.

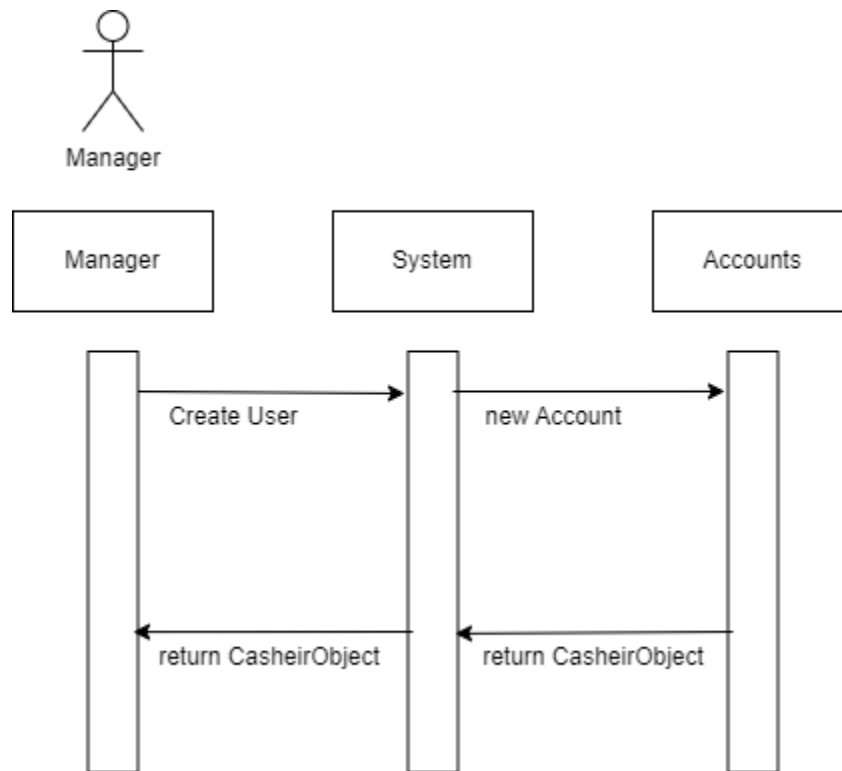
Cashier Diagram



Manager Diagram



Manager Account Diagram



Task 02

Object-Oriented Programming Concepts

The fundamental ideas of OOP, which provide an organised and modular approach to software development, serve as the foundation for the system's design and execution. The following are the main OOP ideas used in this project:

1. Classes and Objects

Several classes that reflect real-world things including users, cashiers, managers, and bags make up the system. Every class contains characteristics and actions specific to the entity it represents. For instance, the Bag class has methods to manage the following data points: id, name, category, price, and quantity.

2. Abstraction

Abstraction is used to define essential characteristics of an object while hiding the unnecessary details. In this system, abstract classes or interfaces can be utilized to define common behaviours among different user types, ensuring that specific details are implemented only in derived classes.

3. Inheritance

The system can establish a hierarchical link between classes thanks to inheritance. For example, the Manager class derives from the Cashier class, which enables it to add new features like the creation of user accounts and reuse and extend the operations of a cashier. This encourages the logical organisation of similar functionalities and the reuse of code.

4. Encapsulation

Access modifiers are used to limit access to a class's internal operations, which results in encapsulation. Public methods are offered to interact with attributes, which are often kept private or protected. This guarantees that an object's internal state is shielded against unwanted access and alteration.

5. Polymorphism

The ability to consider objects as instances of their parent class thanks to polymorphism makes it possible to build methods that can work uniformly on objects belonging to different classes. For instance, the system can incorporate a technique for showing user data that is applicable to several user roles, even if each role may have different characteristics and actions.

Uses of OOP Concepts

The programming paradigm known as object-oriented programming, or OOP, uses classes and objects to create and develop applications. OOP has the following main applications and advantages:

Modularity: Object-oriented programming (OOP) enables developers to decompose intricate issues into more manageable modules or objects. The code can be made more modular and structured by having each object represent a real-world thing with distinct characteristics and behaviours.

Reusability: OOP makes it possible to reuse previously written code through inheritance. In order to reduce duplication and encourage code reuse, developers might design new classes that inherit properties and methods from preexisting ones.

Scalability: OOP facilitates the addition of new features and the modification of current ones. The scalability of the system is improved by the fact that objects encapsulate data and behaviour, allowing changes to be made to one component with little to no effect on other components.

Maintainability: In OOP, the encapsulation principle aids in concealing an object's internal state and implementation specifics. Code maintenance and updating become simpler as a result of the decrease in dependencies.

Data Abstraction: Using OOP, programmers can represent complicated real-world items abstractly. The system is made simpler via abstraction, which emphasises the important aspects while disregarding the minutiae.

Significance

By putting these OOP ideas into effect, the codebase becomes scalable, modular, and easy to maintain, which guarantees long-term adaptability while simultaneously satisfying the functional requirements of the system. Individual parts, including inventory control and user account management, can be independently designed, tested, and updated thanks to a modular framework. This division of responsibilities improves the overall robustness of the system and makes troubleshooting easier.

Another important advantage of applying OOP principles is scalability. The system is easily expandable to support new features, such more user roles or intricate inventory categorisations, as "The Little Bag Shop" evolves. Because of its adaptability, the system can grow with the company and meet new needs without needing to be completely redesigned.

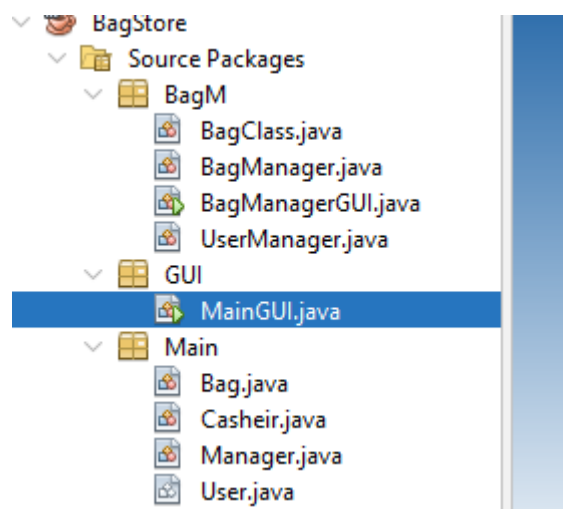
The system can be effectively updated and maintained throughout time thanks to the emphasis on maintainability. The system's structure is clear and consistent due to the adherence to OOP principles, such as inheritance and encapsulation, which facilitates code comprehension and modification by developers. This makes it possible to implement new features more quickly and lowers the chance of introducing issues during upgrades.

The assignment also places a strong emphasis on moral principles like safe coding procedures and understandable documentation. By employing secure coding techniques, consumers' faith in the system is maintained and important user and corporate data is shielded from potential attacks. Thorough documentation clarifies the architecture and functionality of the system, making it easier for future engineers to grasp. Since transparency encourages accountability and makes knowledge exchange easier, it is essential for the development of ethics.

All things considered; this project offers a useful illustration of how OOP concepts may be applied to create a functional application. The work exhibits a profound comprehension of theoretical principles and their practical implementation, offering significant perspectives on software development. The project emphasises how crucial it is to have a carefully thought-out and organised system that can easily adjust to new developments and modifications in the future. Establishing a benchmark for conscientious and proficient software development, the project incorporates ethical considerations while emphasising maintainability.

Codes Snippets of The Project

Use Of Classes and Objects



Use of Abstraction

```
// BagClass.java
public class BagClass {
    private int id;
    private String name;
    private String category;

    public BagClass(int id, String name, String category) {
        this.id = id;
        this.name = name;
        this.category = category;
    }

    // Getter methods for accessing properties
    public int getId() { return id; }
    public String getName() { return name; }
    public String getCategory() { return category; }

    // toString method to provide a string representation
    @Override
    public String toString() {
        return "BagClass{id=" + id + ", name='" + name + "', category='" + category + "'}";
    }
}
```

Use of Inheritance

```
// Manager.java (Superclass)
public class Manager {
    protected int id;
    protected String username;

    public Manager(int id, String username) {
        this.id = id;
        this.username = username;
    }

    public int getId() { return id; }
    public String getUsername() { return username; }

    public abstract void manage();
}

// BagManager.java (Subclass)
public class BagManager extends Manager {

    public BagManager(int id, String username) {
        super(id, username);
    }

    @Override
    public void manage() {
        System.out.println("Managing bags...");
    }
}
```

Use of Encapsulation

```
// BagManager.java
public class BagManager {
    private List<BagClass> bags = new ArrayList<>();

    // Public method to add a bag
    public void addBag(BagClass bag) {
        bags.add(bag);
    }

    // Public method to get all bags
    public List<BagClass> getAllBags() {
        return new ArrayList<>(bags);
    }

    // Public method to search bags by category
    public List<BagClass> searchBagsByCategory(String category) {
        return bags.stream()
            .filter(bag -> bag.getCategory().equalsIgnoreCase(category))
            .collect(Collectors.toList());
    }
}
}
```

Use of Polymorphism

```
// MainGUI.java
public class MainGUI extends JFrame {
    private User loggedInUser;

    private void showManagerPanel() {
        // Some code for manager panel
    }

    private void showCashierPanel() {
        // Some code for cashier panel
    }

    private class LoginButtonListener implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent e) {
            String username = usernameField.getText();
            String password = new String(passwordField.getPassword());

            // Example of polymorphism in action
            if ("manager".equals(username) && "pass".equals(password)) {
                loggedInUser = new Manager(1, "Manager", username, password);
                showManagerPanel();
            } else if ("cashier".equals(username) && "pass".equals(password)) {
                loggedInUser = new Cashier(2, "Cashier", username, password);
                showCashierPanel();
            } else {
                JOptionPane.showMessageDialog(null, "Invalid username or password");
            }
        }
    }
}
}
```


Task 03

User Manul for The Little Bag Shop

Overview

"The Little Bag Shop" is a quaint little business situated in a busy city on a side street. The store provides a wide selection of bags to meet the demands of a wide spectrum of customers, from stylish briefcases to vibrant totes. Owing to increased demand, the store has chosen to automate its transaction process in order to increase productivity. The Cashier and Manager user levels are the two separate ones in the system. The main characteristics of the system and the functionalities accessible to each user level are summarised in this guidebook.

System Overview

Designing an automated system for "The Little Bag Shop" involves applying the concepts of object-oriented programming, or OOP. It has two main user levels, each with different features:

Cashier:

View Bag Details: All of the bags that are available in the store can be viewed by cashiers. They can see bag names, categories, and other pertinent details thanks to this function.

Add New Bags: Cashiers have the ability to add new bags to the system, giving specifics like the name and category of the bag. Maintaining the inventory requires this feature.

Search Bags: Cashiers have the option to look for bags by category, such as backpacks, shoulder bags, and tote bags. This feature helps with customer service by making it easier to quickly retrieve specified items.

Manager:

All Cashier Features: Managers can view, add, and search for bags, among other features that are available to cashiers.

establish New User Accounts: In order to control who has access to the system, managers can establish new user accounts for cashiers. Safe and regulated system access is guaranteed by this feature.

Key Functionalities

Bag Management:

Adding Bags: By entering the bag's name and category, users can add new bags. This data is shown in the bag list and stored in the system.

Viewing Bags: Users are able to view every bag in the inventory using the system. This tool helps with inventory management and client enquiries by offering a detailed list of all bags that are available.

Bags: Users have the option to search for bags based on categories. This feature makes it easier for customers to find particular types of bags fast, which enhances their shopping experience.

User Management:

Account Creation: New cashiers can have their accounts created by managers. With the help of this function, the business can control system access and make sure that only authorised workers are able to add or see bag details.

Data Storage

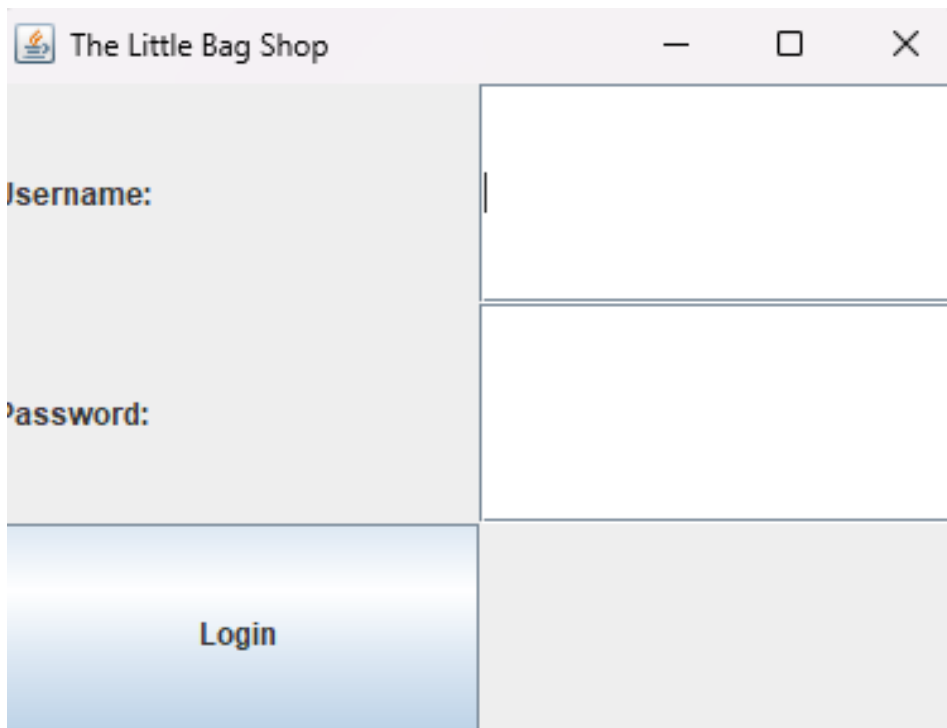
All information is stored and retrieved via text files, including user accounts and bag details. By using this method, data persistence is guaranteed, even in the event of a system restart. When the system starts up, it reads the data from the files and writes any updated or new data back to them as things change. This configuration offers an easy-to-maintain and backup method for handling data in an efficient yet straightforward manner.

Conclusion

The automated "The Little Bag Shop" system streamlines the handling of user accounts and bag details, improving the shop's operating efficiency. Utilising OOP principles guarantees a system that is scalable, modular, and manageable. This handbook provides guidance on utilising the system's features and comprehending its architecture. Please use the provided contact information if you require any additional information or technical assistance.

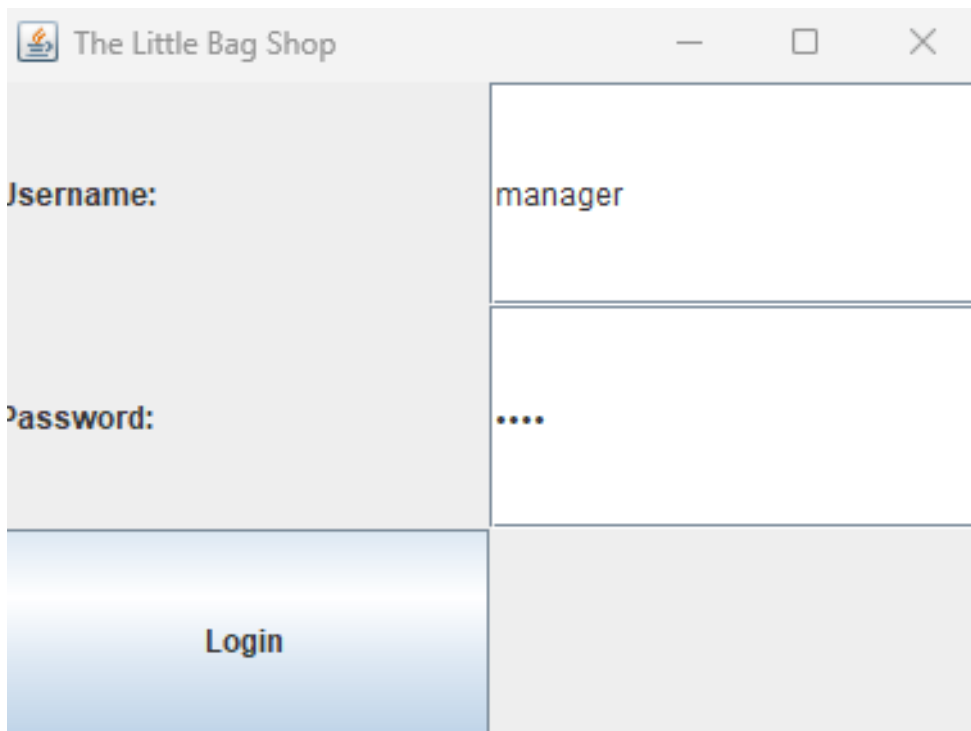
Login Page

User has to login with the given information and press the login button to access the User Menu.



The screenshot shows a window titled "The Little Bag Shop". Inside the window, there is a login form. The form consists of two input fields: one for "Username:" and one for "Password:". Below these fields is a blue button labeled "Login".

Login (Filled)



The Little Bag Shop

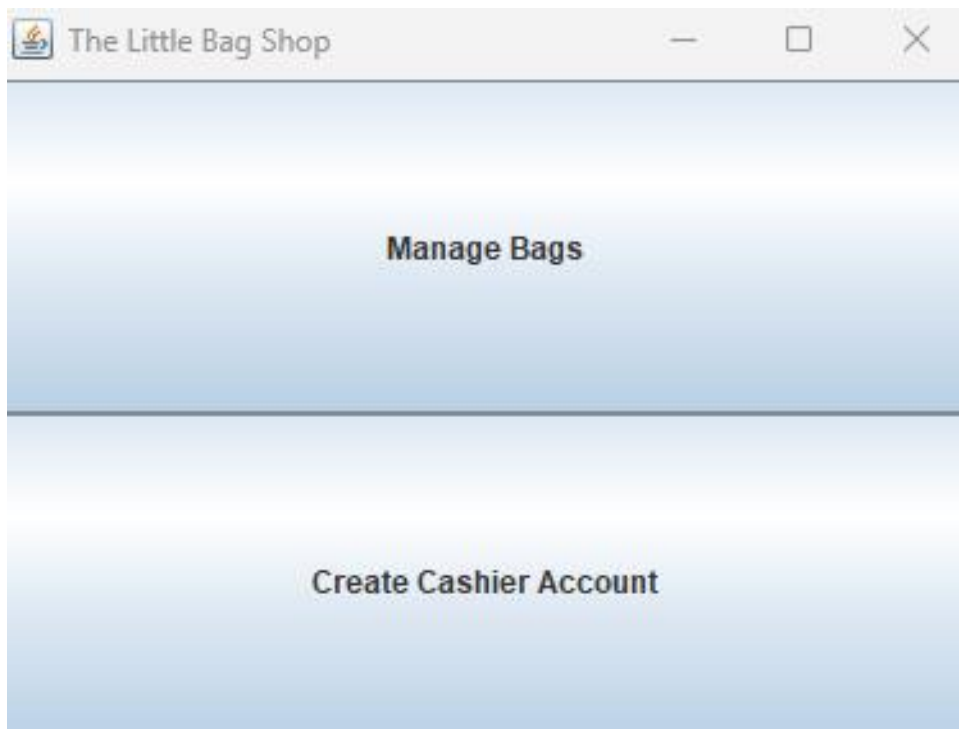
Username: manager

Password:

Login

Main Menu (Manager)

After Putting the Logging Information, The User gets Greeted with The Main Menu, then the user can choose between Manage Bags and Create Cashier Account.



The Little Bag Shop

Manage Bags

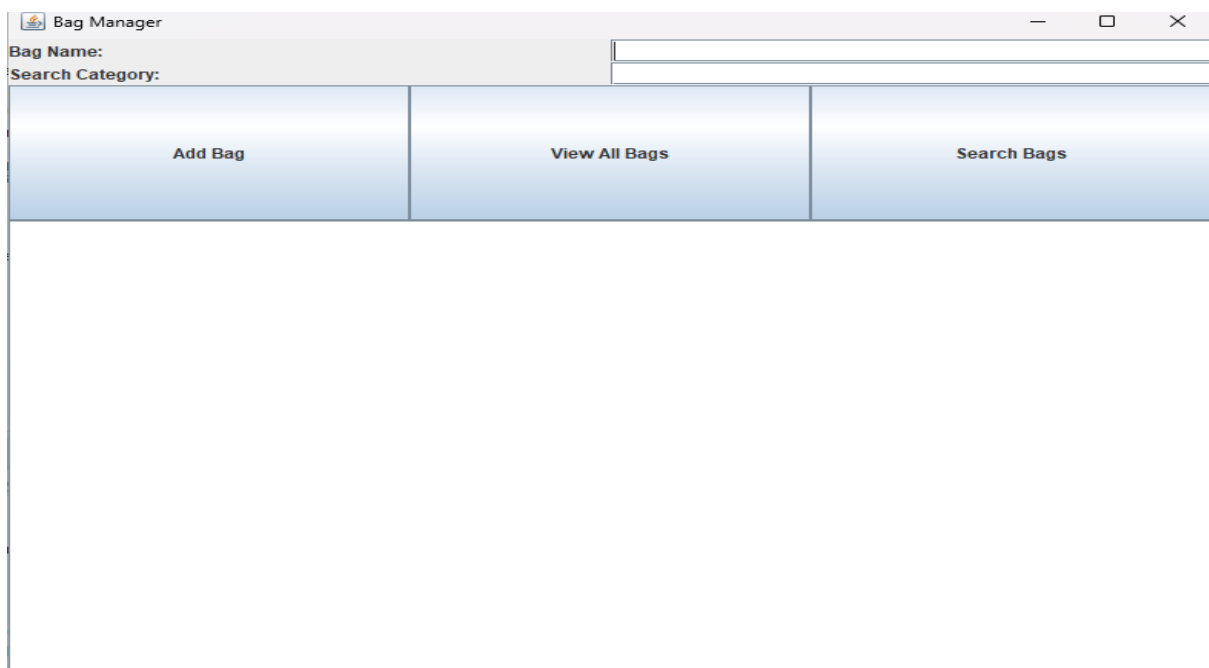
Create Cashier Account

Main Menu (Cashier)

After Putting the Logging Information, The User gets Greeted with The Main Menu, then the user can choose Manage Bags.



Bag Manager



Bag Manager (Add)

User has to fill in Details to the specific Fields and Press Add Bag Button to Save The Details.

Bag Manager

Bag Name:

Balanciaga

Search Category:

B

Add Bag

View All Bags

Search Bags

Bag Manager (Added)

Bag Manager

Bag Name:

Search Category:

Add Bag

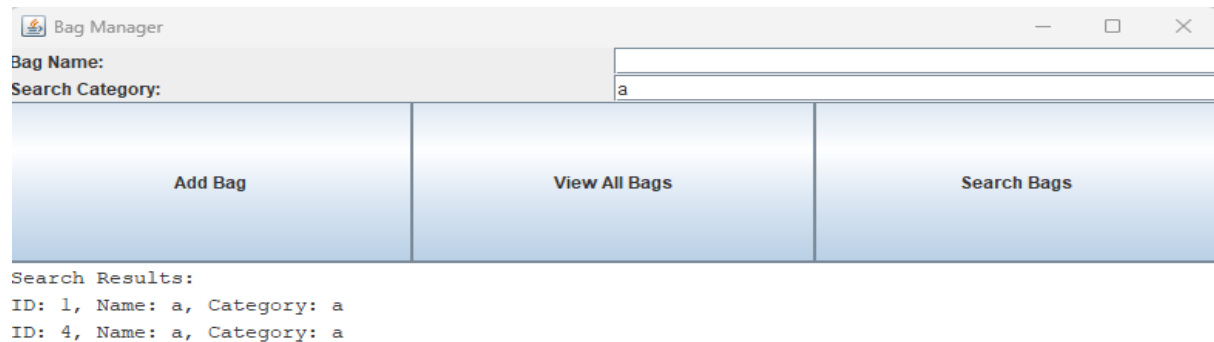
View All Bags

Search Bags

Added: ID: 6, Name: Balanciaga, Category: B

Bag Manager (Search)

User can search for specific bag using search bags button

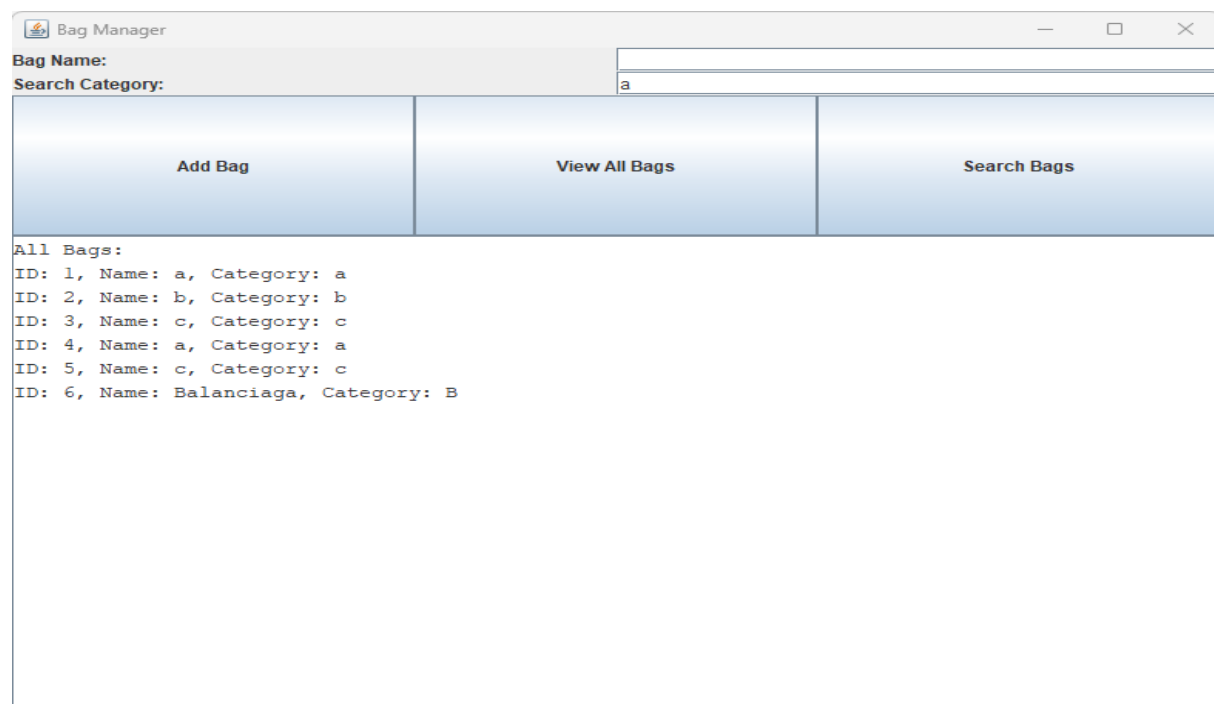


The screenshot shows the Bag Manager application window. At the top, there is a title bar with the text "Bag Manager" and standard window controls. Below the title bar, there are two input fields: "Bag Name:" and "Search Category:". The "Search Category:" field contains the letter "a". Below these fields, there are three buttons: "Add Bag", "View All Bags", and "Search Bags". The "Search Bags" button is highlighted. Below the buttons, the search results are displayed in a text area:

```
Search Results:
ID: 1, Name: a, Category: a
ID: 4, Name: a, Category: a
```

Bag Manager (View All)

User can use the view all bags button to view all the saved bags

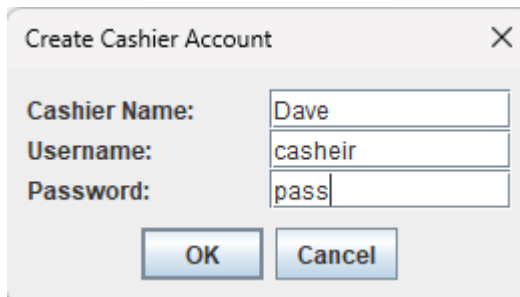


The screenshot shows the Bag Manager application window. At the top, there is a title bar with the text "Bag Manager" and standard window controls. Below the title bar, there are two input fields: "Bag Name:" and "Search Category:". The "Search Category:" field contains the letter "a". Below these fields, there are three buttons: "Add Bag", "View All Bags", and "Search Bags". The "View All Bags" button is highlighted. Below the buttons, the view all bags results are displayed in a text area:

```
All Bags:
ID: 1, Name: a, Category: a
ID: 2, Name: b, Category: b
ID: 3, Name: c, Category: c
ID: 4, Name: a, Category: a
ID: 5, Name: c, Category: c
ID: 6, Name: Balanciaga, Category: B
```

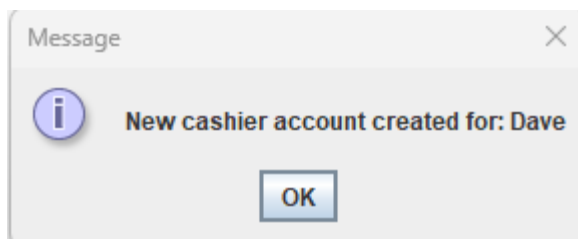
Add Account (Manager)

User can add accounts which will be saved to a txt file



A dialog box titled "Create Cashier Account" with a close button (X) in the top right corner. It contains three input fields: "Cashier Name:" with the text "Dave", "Username:" with the text "casheir", and "Password:" with the text "pass". Below the input fields are two buttons: "OK" and "Cancel".

Add Account Succes (Manager)



A message dialog box titled "Message" with a close button (X) in the top right corner. It features an information icon (i) on the left and the text "New cashier account created for: Dave" in the center. Below the text is an "OK" button.

References

Anna Monus (2023) *Using OOP concepts to write high-performance Java Code (2023)*, Raygun Blog. Available at: <https://raygun.com/blog/oop-concepts-java/> (Accessed: 03 August 2024).

GeeksforGeeks (2024) *Object Oriented Programming (OOPS) concept in Java*, GeeksforGeeks. Available at: <https://www.geeksforgeeks.org/object-oriented-programming-oops-concept-in-java/> (Accessed: 03 August 2024).

Gillis, A.S. and Lewis, S. (2024) *What is object-oriented programming (OOP)?: Definition from TechTarget, App Architecture*. Available at: <https://www.techtarget.com/searchapparchitecture/definition/object-oriented-programming-OOP> (Accessed: 03 August 2024).

Writer, C.B.T. *et al.* (2022) *What is OOP (object oriented programming)?*, Spiceworks Inc. Available at: <https://www.spiceworks.com/tech/devops/articles/object-oriented-programming/> (Accessed: 03 August 2024).