BOOK STORE INVENTORY MANAGEMENT SYSTEM

DHESHIEGHAN A/L SARAVANA MOORTHY
PRAVINRAJ A/L SIVABATHI
MIKHAIL BIN YASSIN

PROBLEM STATEMENT

Traditional bookstores in rural areas still rely on manual processes to manage book records, stock levels, customer data, and supplier orders. This leads to inefficiencies such as inventory errors, restocking delays, and poor customer satisfaction due to mismanagement and lack of automation.

PROPOSED SOLUTION

Implement a digital inventory management system that automates book stock tracking, simplifies customer purchases, and streamlines supplier ordering. The system will enhance operational efficiency, reduce manual workload, and improve overall user experience for admins, customers, and suppliers.

A. ENCAPSULATION AND DATA HIDING

```java
class Name{
    private String fName;
    private String lName;

    Name(String f, String l){
        fName = f;
        lName = l;
    }
    //GETTERS
    public String getfName(){
        return fName;
    }
    public String getlName(){
        return lName;
    }


    //SETTERS
    public void setfName(String f){
        fName = f;
    }

    public void setlName(String l){
        lName = l;
    }
}
```

Encapsulation involves bundling attributes and methods within a class, controlling access to the data through methods while hiding the internal implementation details. In the inventory system, classes encapsulate data related to books, orders, and user information. Public methods enable controlled access to this data, maintaining data integrity and security.

H. FILE OPERATION (ADD, DELETE, EDIT,VIEW)

```java
public  Vector<Book> getBooksfromFile() throws FileNotFoundException {
    Vector<Book> books = new Vector<Book>();
    Scanner sc = new Scanner(new File(pathname:"booksDatabase.txt"));

    while(sc.hasNext()){
        String bookId = sc.next();
        String title = sc.next();
        String mainAuthor = sc.next();
        int genre = sc.nextInt();
        int quantityInStock = sc.nextInt();
        double bookPrice = sc.nextDouble();
        Book book = new Book(bookId, title, mainAuthor, genre, quantityInStock,bookPrice);
        books.add(book);
    }
    return books;
}
```

```java
public void addBooksIntoFile(Vector<Book> bk) throws IOException{
    Scanner scan = new Scanner(System.in);
    System.out.println(x:"===========ADD BOOK===========");
    System.out.print(s:"\nEnter Book ID: ");
    String id = scan.nextLine();

    Vector<Book> bkList = new Vector<Book>();
    bkList = getBooksfromFile();

    for(Book c:bkList){
        while (c.getBookID().equals(id.toUpperCase())) {
            System.out.println(x:"Book ID already exists. Please choose a different Book ID.");
            System.out.print(s:"\nEnter a new Book ID : ");
            id = scan.nextLine().toUpperCase();
        }
    }

    System.out.print(s:"\nEnter Title: ");
    String ttl = scan.nextLine();

    System.out.print(s:"\nEnter Main Author: ");
    String mainAuthor = scan.nextLine();
    int genreOption = 0;
    do{
        System.out.print(s:"\nEnter new book genre (1-Romance , 2-Mystery , 3-Fantasy , 4-Comedy , 5-Thriller) : ");
        try {
            genreOption = scan.nextInt();
            if(genreOption <1 || genreOption>5){
                System.out.println(x:"Invalid option entered. Please enter a number between 1 and 5. Try Again :)");
            }
        } catch (InputMismatchException e) {
            System.out.println(x:"Invalid option entered. Please Enter an appropriate number.\nPress any key to continue...");
            scan.nextLine();
            genreOption = 8;
        }
```

```java
public Boolean editBookDetails(Vector<Book> books,int category, String id, String value) throws IOException{
    value = value.replaceAll(regex:" ", replacement:"_");
    for(Book book:books){
        if(book.getBookID().equals(id.toUpperCase())){
            switch (category) {
                case 1:
                    book.setTitle(value);
                    break;
                case 2:
                    book.setMainAuthor(value);
                    break;
                case 3:
                    book.setGenre(Integer.parseInt(value));
                    break;
                case 4:
                    book.setQuantityInStock(Integer.parseInt(value));
                    break;
                case 5:
                    book.setBookPrice(Double.parseDouble(value));
                    break;
                default:
                    return false;
            }
        }

    }
    if(books !=null){
        FileWriter file = new FileWriter(fileName:"booksDatabase.txt",append:false);
        for (Book bk : books) {
            file.write(bk.getBookID().toUpperCase()+ " "+bk.getTitle()+ " "+bk.getMainAuthor()+ " "+bk.getGenre()+ " "+bk.getQuantityInStoc
        }
        file.close();
    }
    return true;
}
```

```java
public void removeBookFromFile(Vector<Book> bk2) throws IOException{
    Scanner scan = new Scanner(System.in);

    if(bk2.size() != 0){
        viewAllBooks(bk2,role:1);
        String bookId = "";
        int index =0;

        System.out.print(s:"\nEnter Book ID : ");
        bookId = scan.nextLine().toUpperCase();
        boolean validation  = false;

        for (Book bk : bk2) {
            if (bk.getBookID().equals(bookId.toUpperCase())) {
                validation = true;
                break;
            }
        }

        while(validation == false){
                System.out.println(x:"Book ID not exists in our database. Please Try Again!");
                System.out.print(s:"\nEnter Book ID : ");
                bookId = scan.nextLine().toUpperCase();
                for (Book bk : bk2) {
                    if (bk.getBookID().equals(bookId.toUpperCase())) {
                        validation = true;
                        break;
                    }
                }
        }
        PrintWriter outputFile = new PrintWriter(new FileWriter(fileName:"booksDatabase.txt"),autoFlush:false);
        for(Book book:bk2){
            if(book.getBookID().equals(bookId.toUpperCase())){
                break;
```

H. FILE OPERATION (ADD, DELETE, EDIT,VIEW)

- SYSTEM IMPLEMENTS FILE OPERATION CONCEPT TO PERFORM NEW BOOK ADDITION INTO FILE, REMOVING A CERTAIN BOOK DETAIL FROM FILE, READING ALL THE DETAILS FROM THE FILE AND MODIFYING CERTAIN BOOK DETAILS AND UPDATING IT INTO THE FILE.

- ADDBOOKSINTOFILE() METHOD THAT ADDS NEW BOOK DETAILS ENTERED BY THE USER INTO THE BOOKSDATABASE.TXT TEXT FILE.

G. ASSOCIATION

```java
class Admin extends User { //subclass
    private Vector<Customer> customers;
    private static Vector<Book> books;
    private static Vector<OrderManagement> orderList = new Vector<OrderManagement>();
    private Report report;

    public Admin(String id, String name ,String pw, String mail, int roleID, String fName, String lName, Vector<Book> bks){
        super(id, name, pw, mail, roleID,fName,lName);
        customers = new Vector<Customer>();
        books = bks; //Aggregation done
    }

    public Admin(){}
```

Admin.java > Admin > manageBookOperation(Book, int, int)

```java
94
95      public void generateSalesReport(Report r) throws FileNotFoundException{
96          report = r;
97          report.generateReport(roleID:1);
98      }
99
100     public void orderBooks(OrderManagement o) {
101         orderList.add(o);
102     }
103
```

- AN ASSOCIATION CAN BE ONE-TO-ONE, ONE-TO-MANY, OR MANY-TO-MANY.

THE ADMIN CLASS HAS ASSOCIATIONS WITH THE REPORT AND ORDERMANAGEMENT CLASSES.

FIGURE SHOWS THE ADMIN CLASS IMPLEMENTING ASSOCIATION BY DECLARING ORDERLIST VARIABLE THAT REPRESENTS ORDERMANAGEMENT CLASS AND REPORT VARIABLE THAT REPRESENTS REPORT CLASS WITHIN THE CLASS.

- THE GENERATESALESREPORT() METHOD REPRESENTS AN ASSOCIATION BETWEEN THE ADMIN CLASS WITH THE REPORT CLASS, WHERE AN ADMIN CAN GENERATE SALES REPORTS USING A REPORT OBJECT

MEANWHILE, THE ORDERBOOKS() METHOD REPRESENTS AN ASSOCIATION BETWEEN ADMIN CLASS AND ORDERMANAGEMENT CLASS, WHERE AN ADMIN CAN ADD ORDERS TO ITS ORDERLIST VARIABLE USING AN ORDERMANAGEMENT OBJECT IN FIGURE

C. COMPOSITION

```java
class Name{
    private String fName;
    private String lName;

    Name(String f, String l){
        fName = f;
        lName = l;
    }
    //GETTERS
    public String getfName(){
        return fName;
    }
    public String getlName(){
        return lName;
    }


    //SETTERS
    public void setfName(String f){
        fName = f;
    }

    public void setlName(String l){
        lName = l;
    }
}
```

```java
class User extends Menu{ //Parent class
    protected String userID;
    protected String userName;
    protected String password;
    protected String email;
    protected int userRole;
    private Name name;

    public User( String id, String names ,String pw, String mail, int roleID,String fName, String lName){
        userID = id;
        userName = names;
        password = pw;
        email = mail;
        userRole = roleID;
        name = new Name(fName,lName); //composition done
    }

    public User(){}
```

A strong relationship in which one class contains objects of another class as part of its structure. Composition is a way to design classes where one class contains an object of another class, establishing a "has-a" relationship.

- Figure shows the User class and its composition relationship with a name class. The User class has a Name object in its structure, according to this composition which demonstrates how the User class integrates and depends on the attributes and functions of the Name object. This implies that the User class has a Name object, which leads to an interconnection of those two classes.

D. AGGREGATION

```java
class Book{ //aggregate to Admin
    private String bookID;
    private String title;
    private String mainAuthor;
    private int genre;
    private int quantityInStock;
    private double bookPrice;


    public Book(){}

    public Book(String id, String titleBook, String author, int genreCat, int stock,double price){
        bookID = id;
        title = titleBook;
        mainAuthor = author;
        genre = genreCat;
        quantityInStock = stock;
        bookPrice = price;
    }

    tabnine: test | explain | document | ask
    public double getBookPrice() {
        return bookPrice;
    }
    tabnine: test | explain | document | ask
    public void setBookPrice(double bookPrice) {
        this.bookPrice = bookPrice;
    }
```

```java
class Admin extends User {
    private Vector<Customer> customers;
    private static Vector<Book> books;
    private static Vector<OrderManagement> orderList = new Vector<OrderManagement>();
    private Report report;

    public Admin(String id, String name ,String pw, String mail, int roleID, String fName, String lName, Vector<Book> bks){
        super(id, name, pw, mail, roleID,fName,lName);
        customers = new Vector<Customer>();
        books = bks; //Aggregation done
    }

    public Admin(){}

    tabnine: test | explain | document | ask
    public void viewAllCustomers(Customer c) throws FileNotFoundException{
        System.out.print(s:"\033[H\033[2J");
        System.out.flush();
        InventorySystem.header();
        customers = c.getCustomersfromFile();
        c.viewAllCustomers(customers);
    }
```

- A LOOSE RELATIONSHIP IN WHICH ONE CLASS IS ASSOCIATED WITH ANOTHER CLASS BUT DOES NOT OWN ITS OBJECTS.
- AN AGGREGATE RELATIONSHIP, WHICH INCLUDES A STATIC VECTOR CALLED BOOKS, IS SHOWN IN FIGURE OF THE ADMIN CLASS. THE ADMIN CLASS CONSTRUCTOR RECEIVES A BOOK CLASS VECTOR AS AN ARGUMENT, INDICATING THAT THE ADMIN CLASS IS MERGING THE BOOK OBJECTS. WE HAVE DEVELOPED THE GETTER AND SETTER METHODS FOR A BOOK VECTOR TO SHOW THAT WE CAN RETRIEVE AND MODIFY ANY COMBINATION OF BOOKS. FIGURE SHOWS THE BOOK CLASS IMPLEMENTATION.

B. INHERITANCE

- IT IS USED TO CREATE A HIERARCHY IN WHICH CLASSES DECLARED AS SUBCLASSES INHERIT PROPERTIES AND BEHAVIOUR FROM SUPERCLASSES. IT ESTABLISHED AN “IS-A” RELATIONSHIP BETWEEN CLASSES. FIGURE 2.0 SHOWS THE USER CLASS, WHICH REPRESENTS THE PARENT CLASS IN THE SYSTEM. MEANWHILE, ADMIN, CUSTOMER AND BOOKSUPPLIER CLASSES ARE SUBCLASSES FOR THE PARENT CLASS. THESE SUBCLASSES INHERIT THE METHODS AND VARIABLES OF THEIR PARENT CLASS, WHICH ARE DECLARED AS PUBLIC OR PROTECTED AS THEIR ACCESS MODIFIERS.

B. INHERITANCE

```java
class Admin extends User { //subclass
    private Vector<Customer> customers;
    private static Vector<Book> books;
    private static Vector<OrderManagement> orderList = new Vector<OrderManagement>();
    private Report report;

    public Admin(String id, String name ,String pw, String mail, int roleID, String fName, String lName, Vector<Book> bks){
        super(id, name, pw, mail, roleID,fName,lName);
        customers = new Vector<Customer>();
        books = bks; //Aggregation done
    }

    public Admin(){}

    public void viewAllCustomers(Customer c) throws FileNotFoundException{
        System.out.print(s:"\033[H\033[2J");
        System.out.flush();
        InventorySystem.header();
        customers = c.getCustomersfromFile();
        c.viewAllCustomers(customers);
    }

    public Vector<Customer> getCustomers() {
        return customers;
    }

    public static Vector<Book> getBooks() {
        return books;
    }
```

```java
class User extends Menu{ //Parent class
    protected String userID;
    protected String userName;
    protected String password;
    protected String email;
    protected int userRole;
    private Name name;

    public User( String id, String names ,String pw, String mail, int roleID,String fName, String lName){
        userID = id;
        userName = names;
        password = pw;
        email = mail;
        userRole = roleID;
        name = new Name(fName,lName); //composition done
    }

    public User(){}

    public String getUserID() {
        return userID;
    }
    public void setUserID(String userID) {
        this.userID = userID;
    }
    public String getUserName() {
        return userName;
    }
    public void setUserName(String userName) {
        this.userName = userName;
    }
```

```java
public class Customer extends User  { //subclass
    private Vector<Book> bookList;
    private Vector<OrderManagement> order = new Vector<OrderManagement>();

    public Customer(){}

    public Customer(String id, String name ,String pw, String mail, int roleID, String fName, String lName){
        super(id, name, pw, mail, roleID,fName,lName);
    }

    public static void registration() throws IOException{
        Vector<Customer> cust = new Vector<Customer>();
        cust = getCustomersfromFile();
        Scanner sc = new Scanner(System.in);
        System.out.print(s:"\033[H\033[2J");
        System.out.flush();
        InventorySystem.header();
        System.out.print(s:"\n\nEnter your username : ");
        String username = sc.nextLine();

        for(Customer c:cust){
            while (c.getUserName().equals(username)) {
                System.out.println(x:"Username already exists. Please choose a different username.");
                System.out.print(s:"\nEnter a new username : ");
                username = sc.nextLine();
            }
        }
```

```java
ss BookSupplier extends User{ //Subclass
 private Vector<OrderManagement> orderList;
 public BookSupplier(String id, String name ,String pw, String mail, int roleID,String fName,String lName){
     super(id, name, pw, mail, roleID,fName,lName);
 }

 public BookSupplier(){
 }


 public static void displaySupplier(User u){
     System.out.println(x:"┌──────────────┬──────────────┬──────────────┬──────────────┐");
     System.out.println(x:"│ Supplier ID  │ Username     │ Full Name    │ Email        │");
     System.out.println(x:"├──────────────┼──────────────┼──────────────┼──────────────┤");

     System.out.printf(format:"│ %-15s │ %-19s │ %-19s │ %-19s │%n", u.getUserID(), u.getUserName(), u.getName().getfName() + " " + u.get

     System.out.println(x:"└──────────────┴──────────────┴──────────────┴──────────────┘");
```

E. POLYMORPHISM

```java
abstract class  Menu { //abstract class that is used by the user supeclass and its subclasses to perform polymorphism
    abstract protected int viewMenu(); //abstract function
}
```

Admin.java > Admin > manageBookOperation(Book, int, int)

```java
    @Override
    protected int viewMenu(){
        Scanner sc = new Scanner(System.in);
        int option = 0;
        do{
            System.out.print("\033[H\033[2J");
            System.out.flush();
            InventorySystem.header();
            System.out.println("                                        ");
            System.out.println("              Admin Menu                ");
            System.out.println("----------------------------------------");
            System.out.println(" 1. View Books Catalog                  ");
            System.out.println(" 2. Order Book                          ");
            System.out.println(" 3. View Customer Orders                ");
            System.out.println(" 4. View Supplier Orders                ");
            System.out.println(" 5. View All Orders                     ");
            System.out.println(" 6. Manage Book                         ");
            System.out.println(" 7. Generate Report                     ");
            System.out.println(" 8. View All Customers                  ");
            System.out.println(" 9. Exit                                ");
            System.out.println("                                        ");

            System.out.print("\n\n Enter the option (1-9) : ");
            try {
                option = sc.nextInt();
                if(option < 1 || option > 9){
                    System.out.println("Invalid option entered. Please enter a number between 1 and 9. Try Again :)");
                    sc.nextLine();
                    sc.nextLine();

                }
```

User.java > User > setPassword(String)

```java
                    String fName = ary[0];
                    String lName = ary[1];
                    User temp = new User(id, name, pw, mail, roleID,fName,lName);
                    cust.add(temp);
                }
            }

            sc.nextLine();
            sc.close();

            return cust;

        }

        protected int viewMenu(){
            return 0;
        } //Polymorphism done using abstraction.
    }
```

THE ABILITY TO TREAT DIFFERENT CLASSES AS INSTANCES OF THE SAME CLASS THROUGH INHERITANCE AND METHOD OVERRIDING.
- THE MENU CLASS IN FIGURE 5.0 DECLARES AN ABSTRACT METHOD CALLED VIEWMENU(). THE METHOD VIEWMENU() IS IMPLEMENTED WITH POLYMORPHISM FEATURES. THE ADMIN CLASS AND USER CLASS IN FIGURE 5.1 AND 5.2 INHERITS FROM THE MENU CLASS AND DEMONSTRATES POLYMORPHISM BY OFFERING A DISTINCT IMPLEMENTATION FOR THE VIEWMENU() METHOD. THE METHOD'S IMPLEMENTATION IN BOTH CLASSES OVERRIDES THE ABSTRACT METHOD DECLARED IN THE BASE CLASS, SHOWCASING THE FLEXIBILITY AND ADAPTABILITY OF POLYMORPHISM WITHIN THE STRUCTURE OF INHERITANCE AND METHOD OVERRIDING.

F. EXCEPTION HANDLING

```java
protected int viewMenu(){
    Scanner sc = new Scanner(System.in);
    int option = 0;
    do{
        System.out.print(s:"\033[H\033[2J");
        System.out.flush();
        InventorySystem.header();
        System.out.println(x:"╔═══════════════════════════════╗");
        System.out.println(x:"║          Supplier Menu        ║");
        System.out.println(x:"╠═══════════════════════════════╣");
        System.out.println(x:"║ 1. View all Orders            ║");
        System.out.println(x:"║ 2. Manage Orders              ║");
        System.out.println(x:"║ 3. Manage Account             ║");
        System.out.println(x:"║ 4. Exit                       ║");
        System.out.println(x:"╚═══════════════════════════════╝");

        System.out.print(s:"\n\n Enter the option (1-4) : ");
        try {
            option = sc.nextInt();
            if(option < 1 || option > 4){
                System.out.println(x:"Invalid option entered. Please enter a number between 1 and 4. Try Again :)");
                sc.nextLine();
                sc.nextLine();
            }
        } catch (InputMismatchException e) {
            System.out.println(x:"Invalid option entered. Please Enter an appropriate number.\nPress any key to continue...");
            sc.nextLine();
            sc.nextLine();
            option = 10;
        }
    }while(option < 1 || option > 4);
    return option;
}
```

```java
public void generateReport(int roleID) throws FileNotFoundException{
    System.out.print(s:"\033[H\033[2J");
    System.out.flush();
    InventorySystem.header();
    Scanner sc = new Scanner(System.in);
    OrderManagement orders = new OrderManagement();
    orderList = orders.getOrderFromFile(roleID);
    if(orderList.size() == 0){
        System.out.println(x:"No Order Found.\nPress any key to continue..");
        return;
    }
    System.out.println(
        x:"╔═══════════════════════════════════════════════════╗");
    System.out.println(
        x:"║              REPORT KEDAI BUKU KAMAL              ║");
    System.out.println(
        x:"╠═══════════════════════════════════════════════════╣");
    System.out.println(
        x:"║                  TOP 10 BOOKS SOLD                ║");
    System.out.println(
        x:"╟---------------------------------------------------╢");
    System.out.printf(
        format:"╟%-8s | %-30s  | %-10s╢\n", ...args:"Book ID", "Title", "Quantity");
    System.out.println(
        x:"╟---------------------------------------------------╢");

    double totalAmount = 0;
    double totalAmountOrder = 0;
```

F. EXCEPTION HANDLING

```
public boolean validation(String bookId, int bookQuantity, int roleId) throws FileNotFoundException{
    Vector<Book> bkList = new Vector<Book>();
    Book book = new Book();
    Boolean check = false;
    bkList = book.getBooksfromFile();

    try {
        for (Book b : bkList) {
            if (b.getBookID().equals(bookId)) {
                check = true;
                break;  // Exit the loop if the book is found
            }
        }

        if (!check) {
            System.out.println(x:"Sorry, the book id you entered is not found.");
        }
    } catch (InputMismatchException e) {
        System.out.println(e.getMessage());
        check = false;
    }
    if(roleId == 2){
        for(Book b:bkList){
            if(b.getBookID().equals(bookId)){
                if(b.getQuantityInStock() >= bookQuantity){
                    check = true;
                    break; // Exit the loop if the book is found
                }else{
                    System.out.println(x:"Sorry, the quantity you order is more than the quantity in stock.");
                    check = false;
```

- IT IS USED TO MANAGE EXCEPTIONS THAT MAY OCCUR DURING PROGRAM EXECUTION AND ENSURE THAT ERRORS OR UNEXPECTED SITUATIONS ARE PROPERLY HANDLED. IN AN INVENTORY SYSTEM, THIS MIGHT INVOLVE HANDLING SCENARIOS SUCH AS INCORRECT USER INPUT DURING ORDER PROCESSING OR DATA ENTRY ABOUT PRODUCTS.
- IN ORDER TO EFFECTIVELY MANAGE CERTAIN EXCEPTIONS, THE VIEWMENU(), GENERATEREPORT(), AND VALIDATION() METHODS HAVE BEEN INTEGRATED WITH "TRY" AND "CATCH" BLOCKS AS SHOWN IN FIGURES 6.0, 6.1 AND 6.2 RESPECTIVELY. EXCEPTIONS SUCH AS FILENOTFOUNDEXCEPTION AND INPUTMISMATCHEXCEPTION ARE USED TO PROVIDE SPECIAL INFORMATION ABOUT THE TYPE OF EXCEPTION THAT OCCURRED. MOREOVER, THIS FEATURE IS USED TO UPDATE AND INCLUDE ERROR MESSAGES THAT PROVIDE MORE INFORMATION IN ORDER TO IMPROVE CLARITY AND FACILITATE THE PROCESS OF UNDERSTANDING AND RESOLVING ISSUES.

THANK YOU!