

**Subject: Object Oriented Programming**

**Subject code: IT602**

**Date of submission:5-2-2022**

**Due date of submission:5-2-2022**

**Group Name: Elite**

**Group No: 3**

**Teaching Asst: Hetvi Suthar**

**Student Names:**

**Dev Adnani 202212012**

**Rudrik Prajapati 202212013**

**Sanket Acharya 202212100**

**Yash Khatri 202212086**

**Topic: Develop a console-based application for a retailstore.**

**Question Statement**

**Problem Statement:** Construct a console-based application for a **retail store application**.

**Part I: User Login/ Register**

* Application should start by showing two options: (1) Sign In (2) Register
* In case of Sign In, ask the user to provide his “username” and “password”. If they match, let her in, not otherwise.
* In the case of Register, ask the user to provide his “full name”, “username”, and “password”. In case of password, echo *asterisk* symbol (\*) in place of the typed character. E.g., a user types “1@4”, you should display “\*\*\*”, but you will store the actual password to match when she logs in again.

**Hint:** You can use **java.io.Console.readPassword()** method. Carefully read its description from the Java 8 docs online.

* In case of Sign In, provide an option of “Forgot Password”. Ask the user for her name, if it matches then, let her set a new password and redirect her to login again with the new password.

**Part II: Add Product Information**

* After the successful login, you need to make an interface only for an admin account to add product information.
* In which admin will pass the product information like product id, product name, and their status of availability (e.g., In-store, Out-of-Stock, five copies available, discounted price if any, decide the period of time a customer can return/replace the product).
* Create an interface to compute the total number of products available at the store. Calculate the maximum profit they can make from the sale of the products.

**Part III: Display/Issue/Return Products**

* For a customer account, you need to display the catalog for available products after successful login.
* Add membership functionally with abstract methods: Member and Nonmember. (Provide the user an additional discount if they are a member.)
* Then the application should show three options: (1) Purchase product, (2) Cancel purchased product, (3) Display Profile.
* For the first two options, the customer needs to specify the product id which he wants to issue or return.
* If the product is available, make a child class and its method to store the issue date/return date in the customer’s profile.
* And for the third option, the application should display the customer’s profile using the parent class and its methods.

**Part IV: Calculate Fine for Late Return**

* Make an interface for the admin to calculate the fine for customers.
* If the duration between the issue date and the return date is more than the limits displayed on products, then calculate fine.
* Implement the interface using access modifiers.

**File Structure**

src

* admin(folder/package)
  + Admin.java
  + AdminHelper.java
  + AdminInterface.java
* product(folder/package)
  + Avaliabilty.java
  + Product.java
  + ProductHelper.java
  + ProductSale.java
* user(folder/package)
  + User.java
  + UserHelper.java
  + Membership.java
* utils(folder/package)
  + Utils.java

main.java

Admin Package :

**Admin.java**

import products.Product;

import products.ProductHelper;

public class Admin implements AdminInterface {

// Add A New Product In System - Only To Be Done By Admin

@Override

public void addProducts(Product product,ProductHelper productHelper)

{

productHelper.products.add(product);

}

// Get Total Quantity - All Products

@Override

public int getTotalProducts(ProductHelper productHelper) {

int total = 0;

for (Product pd : productHelper.products)

{

total += pd.getQuantityLeft();

}

return total;

}

// Get A Estimate Of How Much Profit / Money We Can Make If We Sell All Products

@Override

public double getMaxProfit(ProductHelper productHelper) {

//Assuming We Have Margin Of 50% Average (Including All Products) , Hence We After Selling At It Double Price , We Will Make 50% Profit

final double PROFIT\_MARGIN = 0.5;

double total = 0;

for (Product pd : productHelper.products)

{

total += (pd.getQuantityLeft()) \*(pd.getPrice());

}

return total\*PROFIT\_MARGIN;

}

}

Description :

The Admin class has three methods:

* addProducts(Product product, ProductHelper productHelper): This method takes a Product object and adds it to the products list in a ProductHelper object, which contains all the products in the system. This method is used to add new products to the system, and it can only be performed by an admin user.
* getTotalProducts(ProductHelper productHelper): This method returns the total number of products available in the system, by iterating over the products list in the ProductHelper object and summing up the quantityLeft field of each Product object.
* getMaxProfit(ProductHelper productHelper): This method returns an estimate of the maximum profit that can be made if all the products in the system are sold, assuming a profit margin of 50% on each product. This method iterates over the products list in the ProductHelper object, multiplies the quantityLeft and price fields of each Product object, and sums up the results. Finally, it multiplies the total by the profit margin (0.5) to get the estimated maximum profit.

Admin Package :

**AdminHelper.java**

package admin;

import products.Availability;

import products.Product;

import products.ProductHelper;

import static utils.Utils.\*;

public class AdminHelper {

boolean isLogged = false;

Admin admin = new Admin();

public void showAdminMenu(ProductHelper productHelper)

{

while (true)

{

while (!isLogged)

{

*print*("Enter 1 For Login");

*print*("Enter 2 For Exit");

int ch = *userInputInt*("Enter Your Choice :");

switch (ch)

{

case 1 ->

{

String uname = *userInputString*("username");

String password = *hashedPassword*();

if(uname.equals("admin") && password.equals("daiict"))

{

isLogged=true;

}

else

{

*print*("Oops Wrong Password / Username");

}

}

case 2->

{

return;

}

default -> *print*("Enter Correct Choice");

}

while (true)

{

*print*("Enter 1 For Adding Product");

*print*("Enter 2 For Getting Total Products(Quantity)");

*print*("Enter 3 For Getting Max Profit After Selling All Products");

*print*("Enter 4 Going Back To Main Screen");

*print*("Enter 5 For Clearing Screen");

ch = *userInputInt*("Enter Your Choice :");

switch (ch)

{

case 1->

{

//Add Product

String productName = *userInputString*("Product Name");

int quantityLeft = *userInputInt*("Enter Quantity Left :");

double price = *userInputDouble*("Enter Price :") ;

double discountedPrice = *userInputDouble*("Enter Discounted Price");

if(price < 0 && quantityLeft < 0 && discountedPrice < 0)

{

*print*("Please Enter Correct Information , (In Terms Of Price,Quantity Left,Discounted Price)");

*print*("These Values Cant Be Less Than 0");

}

else

{

Product product = new Product(productName,price,quantityLeft,Availability.*IN\_STORE*,discountedPrice);

admin.addProducts(product,productHelper);

*print*("Product Added In System");

}

}

case 2-> *print*("The Total Quantity Available Is " + admin.getTotalProducts(productHelper));

case 3-> *print*("The Total Profit We Will Make After Selling All Goods Is " + admin.getMaxProfit(productHelper));

case 4->

{

return;

}

case 5-> *clearTerminal*();

default -> *print*("Incorrect input");

}

}

}

}

}

}

Description :

* This code is to be implementing an "Admin" class and an "AdminHelper" class. The "Admin" class provides methods to add new products to the system, get the total quantity of all products, and estimate the maximum profit that could be made if all products were sold. The "AdminHelper" class provides a menu system for an administrator to interact with the "Admin" class.
* Overall, the code is to be implementing a basic management system with a simple menu-based interface for administrators.

Admin Package :

**AdminInterface.java**

package admin;

import products.Product;

import products.ProductHelper;

public interface AdminInterface {

public void addProducts(Product product, ProductHelper productHelper);

public int getTotalProducts(ProductHelper productHelper) ;

public double getMaxProfit(ProductHelper productHelper);

}

Description :

* The AdminInterface is an interface that defines three methods: addProducts, getTotalProducts, and getMaxProfit. Any class that implements the AdminInterface interface will be required to implement these methods.
* The addProducts method takes in a Product object and a ProductHelper object as parameters and adds the Product to the list of products in the ProductHelper.
* The getTotalProducts method takes in a ProductHelper object as a parameter and returns the total quantity of all the products in the ProductHelper.
* The getMaxProfit method takes in a ProductHelper object as a parameter and returns an estimate of the maximum profit that can be made by selling all the products in the ProductHelper.

Product Package :

**Product.java**

package products;

public class Product {

private static int *lastID* = 0;

private final int productId;

private int quantityLeft;

private final String productName;

private final double price;

private Availability availability;

private final double discountedPrice;

// Constructor For Product Class

public Product(String productName, double price,int quantityLeft , Availability availability, double discountedPrice) {

this.productId = ++*lastID*;

this.productName = productName;

this.quantityLeft = quantityLeft;

this.price = price;

this.availability = availability;

this.discountedPrice = discountedPrice;

}

// Get Discounted Price

public double getDiscountedPrice() {

return discountedPrice;

}

//Get : Product ID

public int getProductId() {

return productId;

}

// Get : Quantity Left

public int getQuantityLeft() {

return quantityLeft;

}

// Decrease Quantity Per Product

public void decreaseQuantity() {

--quantityLeft;

}

// Increase Quantity Per Product

public void increaseQuantity() {

++quantityLeft;

}

// Get : Product Name

public String getProductName() {

return productName;

}

// Get : Product Price

public double getPrice() {

return price;

}

// Get - Product Availability

public Availability getAvailability() {

return availability;

}

//Set - Setting Product Availability

public void setAvailability(Availability availability) {

this.availability = availability;

}

}

Description :

* This code defines a class named "Product" that represents a product with various properties such as product ID, name, quantity left, price, availability, and discounted price.
* The class has a constructor that takes in parameters such as product name, price, quantity left, availability, and discounted price to initialize the product properties. It also has getters and setters for all the properties of the product.
* The product ID is generated automatically using a static variable named "lastID" that keeps track of the last assigned ID. The quantity of the product can be decreased or increased using the "decreaseQuantity" and "increaseQuantity" methods respectively.

Product Package :

**Availability.java**

package products;

public enum Availability {

*IN\_STORE*,

*OUT\_OF\_STOCK*,

}

Description :

* This code defines an enum named "Availability" which represents the availability status of a product. It has two possible values: "IN\_STORE" and "OUT\_OF\_STOCK".
* The "IN\_STORE" value indicates that the product is available in the store, while the "OUT\_OF\_STOCK" value indicates that the product is not available and is currently out of stock.

Product Package :

**ProductHelper.java**

package products;

import java.util.ArrayList;

import static utils.Utils.*print*;

public class ProductHelper {

public ArrayList<Product> products = new ArrayList<>();

//Adding Some Demo Data!!

public ProductHelper() {

products.add(new Product("IPhone 7s",999.99, 5,Availability.*IN\_STORE*, 899.99));

products.add(new Product("IPhone 8s",1100.99,6,Availability.*IN\_STORE*, 1000.99));

products.add(new Product("IPhone 9s",1399.99,4,Availability.*IN\_STORE*, 1200.22));

products.add(new Product("IPhone 10s",1766.99,3,Availability.*IN\_STORE*, 1660.99));

}

// Gets Specific Product

public Product getSpecificProduct(int index) {

return products.get(index);

}

// Display All Products

public void showProducts() {

*print*("Product ID\t| Product Name\t\t| Product Price\t\t| Discounted Price\t\t| Quantity Left\t\t| Product Availability");

for (Product pd : products) {

*print*(pd.getProductId() + "\t\t| " + pd.getProductName() + "\t\t| " + pd.getPrice() + "\t\t| " + pd.getDiscountedPrice() + "\t\t\t| " + pd.getQuantityLeft() + "\t\t\t| " + pd.getAvailability());

}

}

/// Getting Total Products Length

public int getProductLength() {

return products.size();

}

// Decrease Products Quantity After We User's It

public void decreaseProductQuantity(int productId) {

if (products.get(productId).getQuantityLeft() > 1) {

products.get(productId).decreaseQuantity();

} else {

products.get(productId).setAvailability(Availability.*OUT\_OF\_STOCK*);

products.get(productId).decreaseQuantity();

}

}

// Increase Products Quantity After User Returns It

public void increaseProductQuantity(int productId)

{

if(products.get(productId).getQuantityLeft() <= 0)

{

products.get(productId).increaseQuantity();

products.get(productId).setAvailability(Availability.*IN\_STORE*);

}

else

{

products.get(productId).increaseQuantity();

}

}

public int calculateFine(int days)

{

if (days < 0)

{

return (days\*10); // 100$ Fine For Negative Value

}

if (days > 5) {

return (days \* 10); // $10 per day fine

} else {

return 0;

}

}

}

Description :

* This code defines a class named "ProductHelper" which helps to manage a list of products. It contains an ArrayList of Product objects and methods to perform various operations on the list of products.
* The constructor of the "ProductHelper" class adds some demo data to the list of products for testing purposes.
* The "getSpecificProduct" method returns a specific product from the list based on the index passed as a parameter.
* The "showProducts" method displays all the products in the list with their respective properties such as product ID, name, price, discounted price, quantity left, and availability.
* The "getProductLength" method returns the number of products in the list.
* The "decreaseProductQuantity" method decreases the quantity of a specific product by one, and if the quantity becomes zero, it sets the availability of the product to "OUT\_OF\_STOCK".
* The "increaseProductQuantity" method increases the quantity of a specific product by one and sets the availability of the product to "IN\_STORE" if it was previously out of stock.
* The "calculateFine" method calculates the fine to be charged to a user for returning a product after the due date. It takes in the number of days the product was late and calculates the fine as $10 per day,

Product Package :

**ProductSale.java**

package products;

import java.util.Date;

public class ProductSale {

private final Product product;

private final Date dateBought;

public ProductSale(Product product, Date dateBought) {

this.product = product;

this.dateBought = dateBought;

}

public Product getProduct() {

return product;

}

public Date getDateBought() {

return dateBought;

}

}

Description :

* The code defines a ProductSale class that represents a single sale transaction of a Product. The ProductSale has two fields, a Product object representing the sold product, and a Date object representing the date the product was sold.
* The ProductSale class has a constructor that takes two parameters, a Product object and a Date object, and initializes its corresponding fields with the provided values.
* The class has two getter methods, getProduct() and getDateBought(), that return the Product object and the Date object, respectively.
* Overall, the ProductSale class is used to keep track of the sales made for a particular product and the date on which they were made.

User Package :

**User.java**

package user;

import products.ProductSale;

import java.util.ArrayList;

public class User {

private static int *lastUid* = 0;

private String username, password,fullName;

private double points = 99999;

private final int uid;

private Membership membership = Membership.*NOT\_A\_MEMBER*;

private final ArrayList<ProductSale> productsBought = new ArrayList<>();

public ArrayList<ProductSale> getProductsBought() {

return productsBought;

}

// Constructor

public User() {

this.uid = ++*lastUid*;

}

// Buy Product

public boolean buyProduct(ProductSale product) {

productsBought.add(product);

return true;

}

// Get Quantity Of Product Bought

public int getProductBoughtSize()

{

return productsBought.size();

}

public void removeProduct(int index) {

productsBought.remove(index);

}

// Get - FullName

public String getFullName() {

return fullName;

}

// Set - FullName

public void setFullName(String fullName) {

this.fullName = fullName;

}

//Get - UserName

public String getUsername() {

return username;

}

// Set Username

public void setUsername(String username) {

this.username = username;

}

// Get - Password

public String getPassword() {

return password;

}

// Set - Password

public void setPassword(String password) {

this.password = password;

}

// Get - User Points

public double getPoints() {

return points;

}

// Set - User Points

public void setPoints(double points) {

this.points = points;

}

// Get User ID

public int getUid() {

return uid;

}

// Get Membership

public Membership getMembership() {

return membership;

}

//Set : Set Membership

public void setMembership(Membership membership) {

this.membership = membership;

}

}

Description :

* It has fields to store the user's username, password, full name, and points.
* It also has a unique ID and a membership status.
* The class provides methods to buy products, get the number of products bought, remove a product from the list of bought products, and get/set user's username, password, full name, and points. Additionally, it includes a list of ProductSale objects representing the products bought by the user.

User Package :

**Membership.java**

package user;

public enum Membership {

*NOT\_A\_MEMBER*,

*IS\_MEMBER*,

}

Description :

* "Membership" which has two values defined: "NOT\_A\_MEMBER" and "IS\_MEMBER
* The "NOT\_A\_MEMBER" value represents that the user is not a member, while the "IS\_MEMBER" value represents that the user is a member

User Package :

**UserHelper.java**

package user;

import products.ProductSale;

import java.util.ArrayList;

import java.util.HashMap;

import static utils.Utils.\*;

public class UserHelper{

//Forget Password - Just Checking If Name Is Same As Old Or Not , If Yes We Change The Password , Else We Return False

public boolean forgetPassword(HashMap<String, User> users) {

String userName,password;

userName = *userInputString*("Username");

password = *hashedPassword*();

boolean userNameCheck = users.containsKey(userName);

if (userNameCheck) {

users.get(userName).setPassword(password);

return true;

} else {

return false;

}

}

public boolean registerUser(HashMap<String, User> users)

{

String userName,password,fullName,isMember;

boolean isRegistered = false;

fullName = *userInputString*("Full name");

userName = *userInputString*("Username");

isMember = *userInputString*("Membership Status , 'Yes' Or 'NO' ");

password = *hashedPassword*();

if(fullName.isEmpty() || userName.isEmpty() || isMember.isEmpty() || password.isEmpty())

{

*print*("Please Enter Some Data !xD");

}

else

{

//Checking If Username Already Exists - Not Permitting Multiple Users With Same Username

boolean allowUser = users.containsKey(userName);

if (!allowUser) {

User createUser = new User();

createUser.setFullName(fullName);

createUser.setUsername(userName);

createUser.setPassword(password);

if(isMember.equalsIgnoreCase("yes"))

{

createUser.setMembership(Membership.*IS\_MEMBER*);

}

users.put(userName, createUser);

isRegistered = true;

}

}

return isRegistered;

}

public User loginUser(HashMap<String,User> users)

{

String userName,password;

User loggedUser = null;

userName = *userInputString*("Username");

boolean userNameCheck = users.containsKey(userName);

if (userNameCheck) {

password = *hashedPassword*();

String userPass = users.get(userName).getPassword();

if (userPass.equals(password)) {

loggedUser = users.get(userName);

}

}

return loggedUser;

}

public void displayUserProfile(User user) {

*print*("--------------------------- USER PROFILE-------------------------------- ");

*print*("ID : " + user.getUid());

*print*("FullName : " + user.getFullName());

*print*("Username : " + user.getUsername());

*print*("Points : " + user.getPoints());

*print*("Membership : " + user.getMembership());

ArrayList<ProductSale> productsBought = user.getProductsBought();

if(productsBought.size() == 0)

{

*print*("You Have Not Bought Any Product");

}

else

{

*print*("Product Bought : ");

if (user.getMembership() == Membership.*IS\_MEMBER*)

{

System.*out*.println("Index\t| Product Name\t| Product Price(Discounted)\t| Product Bought On");

for (int i = 0; i < productsBought.size(); i++) {

System.*out*.println((i+1) + "\t| " + productsBought.get(i).getProduct().getProductName() + "\t| " + productsBought.get(i).getProduct().getDiscountedPrice() + "\t\t\t| " + productsBought.get(i).getDateBought());

}

}

else

{

System.*out*.println("Index\t| Product Name\t| Product Price\t| Product Bought On");

for (int i = 0; i < productsBought.size(); i++) {

System.*out*.println((i+1) + "\t| " + productsBought.get(i).getProduct().getProductName() + "\t| " + productsBought.get(i).getProduct().getDiscountedPrice() + "\t\t\t| " + productsBought.get(i).getDateBought());

}

}

}

*print*("----------------------END OF USER PROFILE-------------------------------- ");

}

}

Description :

* This UserHelper class provides several helper methods for managing user data in a system.
* The forgetPassword() method takes a HashMap of user data as input, prompts the user for their username and a new password, checks if the username exists in the HashMap, and if it does, updates the user's password and returns true. Otherwise, it returns false.
* The registerUser() method takes a HashMap of user data as input, prompts the user for their full name, username, membership status, and a new password, creates a new User object, sets its attributes to the user input, and adds it to the HashMap. It also performs input validation to ensure that required fields are not empty and that no two users have the same username. The method returns true if the user is successfully registered, otherwise false.
* The loginUser() method takes a HashMap of user data as input, prompts the user for their username and password, checks if the username exists in the HashMap, and if it does and the password matches, returns the corresponding User object. Otherwise, it returns null.
* The displayUserProfile() method takes a User object as input, displays the user's profile information, including their ID, full name, username, points, and membership status. It also lists any products the user has bought, along with the date of purchase, if applicable. The method formats the output differently depending on the user's membership status.
* Overall, it provides functionality for managing user accounts in a system, including registration, login, password reset, and displaying user information.

Utils Package :

**Utils.java**

package utils;

import java.io.Console;

import java.util.Scanner;

public class Utils {

Scanner scanner = new Scanner(System.*in*);

public static void print(String msg) {

System.*out*.println(msg);

}

public static String hashedPassword() {

Console console = System.*console*();

String tempPass = "";

if (console != null) {

char tx[] = console.readPassword("Enter Password : ");

tempPass = String.*valueOf*(tx);

if (tempPass.length() != 0) {

for (char c : tx) {

System.*out*.print("\*");

}

}

*print*("");

}

else

{

Scanner scanner = new Scanner(System.*in*);

*print*("Enter Password : " );

tempPass = scanner.nextLine();

}

return tempPass;

}

return tempPass;

}

public static String userInputString(String t) {

Scanner scanner = new Scanner(System.*in*);

String data="";

*print*("Please Enter "+ t +" : ");

data = scanner.nextLine();

return data;

}

public static int userInputInt(String t)

{

Scanner scanner = new Scanner(System.*in*);

int no;

*print*(t);

no = Integer.*parseInt*(scanner.nextLine());

return no;

}

public static double userInputDouble(String t)

{

Scanner scanner = new Scanner(System.*in*);

double no;

*print*(t);

no = Double.*parseDouble*(scanner.nextLine());

return no;

}

public static void clearTerminal() {

System.*out*.print("\033[H\033[2J");

System.*out*.flush();

}

}

Description :

* The "Utils" package contains utility methods that are used throughout the application to perform common tasks like taking user input, printing messages to the console, clearing the terminal, and hashing passwords.
* The class contains several static methods including "print" which prints a message to the console.
* "userInputString" which takes a string input from the user.
* "userInputInt" which takes an integer input from the user.
* "userInputDouble" which takes a double input from the user.
* The "hashedPassword" method takes input from the console in a secure way and returns a password string.
* The "clearTerminal" method clears the terminal screen for better user experience.

src folder :

**Main.java**

import admin.AdminHelper;

import products.Availability;

import products.Product;

import products.ProductHelper;

import products.ProductSale;

import user.Membership;

import user.User;

import user.UserHelper;

import java.util.Date;

import java.util.HashMap;

import static java.lang.System.*exit*;

import static utils.Utils.\*;

public class Main {

public static void main(String[] args) {

boolean isRegistered = false, isLoggedIn = false;

HashMap<String, User> users = new HashMap<>();

UserHelper userHelper = new UserHelper();

AdminHelper adminHelper = new AdminHelper();

ProductHelper productHelper = new ProductHelper();

User loggedUser = null;

int choice;

while (true)

{

while (!isLoggedIn || !isRegistered) {

*print*("Enter 1 For Registering");

*print*("Enter 2 For Login");

*print*("Enter 3 For Admin Console");

*print*("Enter 4 For Forget Password");

*print*("Enter 5 For Clearing Screen");

choice = *userInputInt*("Enter Your Choice");

switch (choice) {

case 1 -> {

isRegistered = userHelper.registerUser(users);

if (!isRegistered)

{

*print*("Username Already Exists");

}

}

case 2 -> {

if (isRegistered) {

loggedUser = userHelper.loginUser(users);

if(loggedUser == null)

{

*print*("Wrong username or Password");

}

else

{

isLoggedIn=true;

}

} else {

*print*("Please Register Before Trying To Login");

}

}

case 3 -> adminHelper.showAdminMenu(productHelper);

case 4 -> {

if (isRegistered) {

boolean passChangeStatus = userHelper.forgetPassword(users);

if (passChangeStatus) {

*print*("Password Changed Successfully");

} else {

*print*("Oops Wrong Username");

}

} else {

*print*("You Must Register Before , If You Wanna Change Password");

}

}

case 5 -> *clearTerminal*();

default -> *print*("Wrong Choice...");

}

}

while (true)

{

productHelper.showProducts();

*print*("Enter 1 For User Profile");

*print*("Enter 2 For Buying The Product");

*print*("Enter 3 For Returning The Product");

*print*("Enter 4 For Logout");

*print*("Enter 5 For Closing The App");

*print*("Enter 6 For Clearing Screen");

choice = *userInputInt*("Enter Your Choice");

switch (choice)

{

case 1 -> userHelper.displayUserProfile(loggedUser);

case 2 ->

{

choice = *userInputInt*("Enter Product ID");

choice=choice-1; // 0 Base Indexing

if (choice >= 0 && choice < productHelper.getProductLength())

{

Product pd = productHelper.getSpecificProduct(choice);

double userPoints = loggedUser.getPoints();

double productPrice = loggedUser.getMembership() == Membership.*IS\_MEMBER* ? pd.getDiscountedPrice() : pd.getPrice();

if (pd.getAvailability() == Availability.*IN\_STORE* && userPoints >= productPrice) {

//Getting CurrDate

Date currentDate = new Date();

//Product Sold Data

ProductSale productBought = new ProductSale(pd,currentDate);

//Status Is Completed

boolean isCompleted = loggedUser.buyProduct(productBought);

//Changing Points

loggedUser.setPoints(userPoints - productPrice);

//Decrease Quantity

productHelper.decreaseProductQuantity(choice);

if (isCompleted) {

*print*("--------------------------- Transaction Status -------------------------------- ");

*print*("Product Bought");

*print*("--------------------------- Transaction Status -------------------------------- ");

userHelper.displayUserProfile(loggedUser);

}

} else {

*print*("--------------------------- Transaction Status -------------------------------- ");

*print*("The Product Is Not Available To Buy Or You Have Less Points Kindly Check");

*print*("--------------------------- Transaction Status -------------------------------- ");

}

}

else

{

*print*("Enter Valid Product ID");

}

}

case 3 ->

{

userHelper.displayUserProfile(loggedUser);

if(loggedUser.getProductBoughtSize() <= 0)

{

*print*("You Must Buy Something , Inorder To Sell !!");

}

else

{

choice=*userInputInt*("Enter ID");

choice=choice-1; // 0 Based Indexing

Date date = loggedUser.getProductsBought().get(choice).getDateBought();

*clearTerminal*();

*print*("--------------------------- IMP Note -------------------------------- ");

*print*("Product Name : " + loggedUser.getProductsBought().get(choice).getProduct().getProductName());

*print*("You Have Bought The Product On : " + date );

*print*("Please Enter After How Many Days You Are Returning The Product (Eg 1,2)");

*print*("Note : If You Return Within 5 Days Of Purchase You Will Not Be Charged Any Fine");

*print*("Adding Negative Value Means We'll Consider It After 5 Days And Start Charging Fine");

*print*("--------------------------- IMP Note Ends -------------------------------- ");

*print*("");

double userPoints = loggedUser.getPoints();

if (choice >= 0 && choice < loggedUser.getProductBoughtSize())

{

int askUserReturn = *userInputInt*("Enter Extra Days : ");

//Getting Specific Product

Product pd = productHelper.getSpecificProduct(choice);

//Increasing Product As It Being Returned

productHelper.increaseProductQuantity(choice);

//Removing Product From User's Account

loggedUser.removeProduct(choice);

// Return Price

double returnPrice = loggedUser.getMembership() == Membership.*IS\_MEMBER* ? pd.getDiscountedPrice() : pd.getPrice();

//Deducting Fine

int fineDeduct = productHelper.calculateFine(askUserReturn);

returnPrice = returnPrice - fineDeduct;

//Setting User Points

loggedUser.setPoints(userPoints+returnPrice);

*print*("--------------------------- Transaction Status -------------------------------- ");

*print*("Fine Charged $ " +fineDeduct);

*print*("Product Sold At $ "+returnPrice);

*print*("--------------------------- Transaction Status -------------------------------- ");

}

else

{

*print*("--------------------------- Transaction Status -------------------------------- ");

*print*("Check The Index While Entering");

*print*("--------------------------- Transaction Status -------------------------------- ");

}

}

}

case 5 ->

{

*print*("Bye Bye");

*exit*(1);

}

case 6 -> *clearTerminal*();

default -> *print*("Enter Valid Choice");

}

if(choice == 4)

{

*print*("Good Bye , Hope You Had A Good Time");

loggedUser=null;

isLoggedIn=false;

break;

}

}

}

}

}

Description :

* This is a Java program that simulates a retail system.
* The system has two types of users, regular and admin.
* Users can register, log in, view and update their profile, buy a product, and return a product.
* The admin can manage the product details.

**How To Run The Project ?**

* Here are the steps to run an IntelliJ project on another system
* Copy the project files: Transfer all the necessary files from the source system to the target system. This includes the source code, libraries, and configuration files.
* Install IntelliJ IDEA: Download and install the latest version of IntelliJ IDEA on the target system.
* Import the project: Launch IntelliJ IDEA and select "File" > "Open" to import the project files into the IDE.
* Run the project: Select the main class or run configuration and click the "Run" button to run the project.

**Screenshot :**

