****

**Universiti Tunku Abdul Rahman**

**Faculty Information and Communication Technology**

**UCCD2044 Object-Oriented Programming Practices**

**Group Assignment: Stock Management System**

**Group No 62**

|  |  |  |
| --- | --- | --- |
| **GROUP MEMBER NAMES** | **STUDENT ID** | **PROGRAM** |
| TAN KAI JUN (LEADER) | 22ACB06494 | CS |
| HO KAH MUN | 22ACB07230 | CS |
| LER JUN WEI | 22ACB07200 | CS |
| LOH WEI JIA | 22ACB07414 | CS |

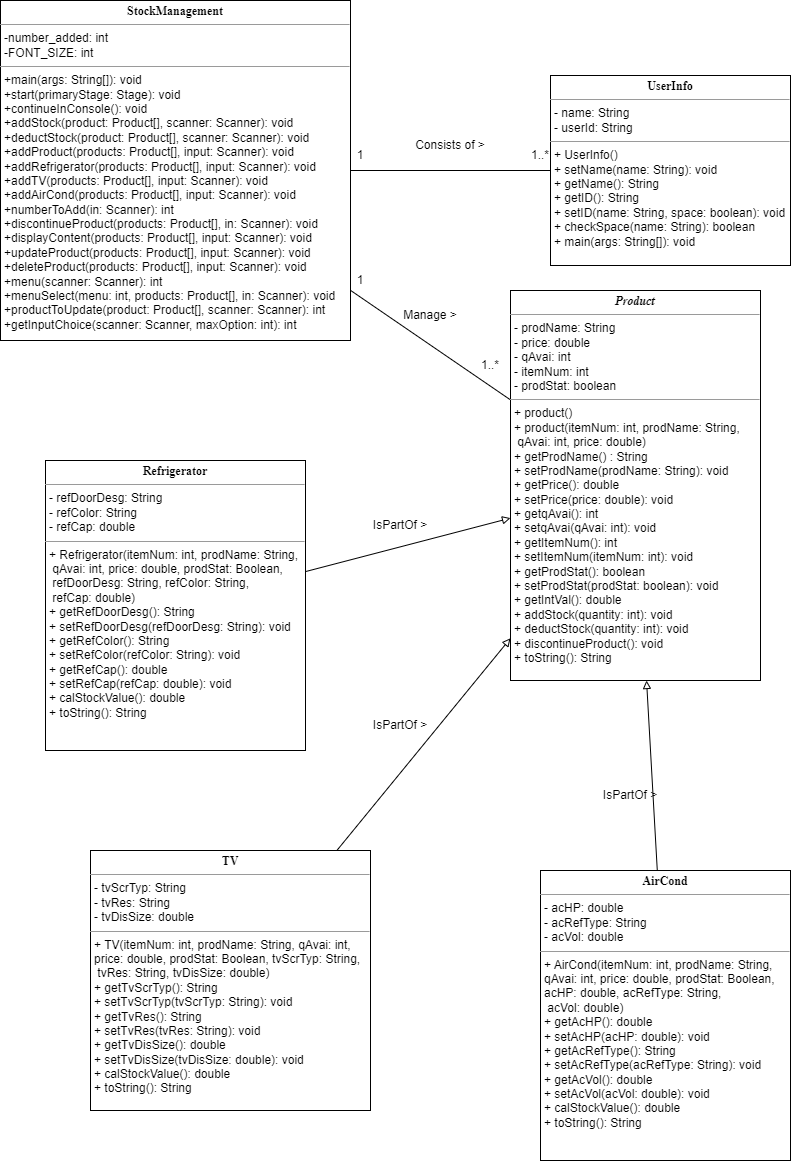
**Table of Contents:**

|  |  |
| --- | --- |
| **Pages** | **Contents** |
| 1 | Cover Page |
| 2 | Table of Contents |
| 3 | Contributions of each members |
| 3 | UML Diagram |
| 4-28 | Source Codes |
| 4-6 | Product.java |
| 7-8 | Refrigerator.java |
| 9-10 | TV.java |
| 11-12 | AirCond.java |
| 13-14 | UserInfo.java |
| 15-28 | StockManagement.java |

**CONTRIBUTION OF EACH MEMBERS:**

|  |  |  |  |
| --- | --- | --- | --- |
| TAN KAI JUN | HO KAH MUN | LER JUN WEI | LOH WEI JIA |
| Stock Management class, GUI, UML Diagram | Product class | Refrigerator, TV and AirCond class, Report | UserInfo class |

**UML Diagram**

****

**SOURCE CODES**

**Product.java**

public abstract class Product {

private String prodName; //product name

private double price; //product price

private int qAvai; //quantity available in stock

private int itemNum; //item number of the product

private boolean prodStat; //product status : active (true)

// Default constructor

public Product() {

prodStat = true; // default status is active

}

// Parameterized constructor

public Product(int itemNum, String prodName, int qAvai, double price, boolean prodStat) {

this.itemNum = itemNum;

this.prodName = prodName;

this.qAvai = qAvai;

this.price = price;

this.prodStat = prodStat;

}

// Getter and setter methods

public String getProdName() {

return prodName;

}

public void setProdName(String prodName) {

this.prodName = prodName;

}

public double getPrice() {

return price;

}

public void setPrice(double price) {

this.price = price;

}

public int getqAvai() {

return qAvai;

}

public void setqAvai(int qAvai) {

this.qAvai = qAvai;

}

public int getItemNum() {

return itemNum;

}

public void setItemNum(int itemNum) {

this.itemNum = itemNum;

}

public boolean getProdStat() {

return prodStat;

}

public void setProdStat(boolean prodStat) {

this.prodStat = prodStat;

}

// Method to get total inventory value

public double getIntVal() {

return price \* qAvai;

}

// Method to allow user to add quantity to stock

public void addStock(int quantity) {

if (prodStat) {

qAvai += quantity;

}

else {

System.out.println("Sorry, cannot add stock to a discontinued product line.");

}

}

// Method to allow user to deduct quantity from stock

public void deductStock(int quantity) {

if (quantity <= qAvai) {

qAvai -= quantity;

}

else {

System.out.println("Not enough stock available to deduct.");

}

}

// Method to allow user to discontinue the product

public void discontinueProduct() {

prodStat = false;

}

// Override toString method

@Override

public String toString() {

return "Item number: " + itemNum +

"\nProduct name: " + prodName +

"\nQuantity available: " + qAvai +

"\nPrice (RM): " + price +

"\nInventory value (RM): " + getIntVal() +

"\nProduct status: " + (prodStat ? "Active" : "Discontinued");

}

}

**Refrigerator.java**

public class Refrigerator extends Product

{

//initialization of refrigerator door design, refrigerator color and refrigeraotr capacity

private String refDoorDesg;

private String refColor;

private double refCap;

//parameterized constructor of Refrigerator class

public Refrigerator(int itemNum, String prodName, int qAvai, double price, Boolean prodStat, String refDoorDesg, String refColor, double refCap)

{

super(itemNum, prodName, qAvai, price, prodStat);

this.refDoorDesg = refDoorDesg;

this.refColor = refColor;

this.refCap = refCap;

}

//getter and setter for refrigerator door design

public String getRefDoorDesg() {

return refDoorDesg;

}

public void setRefDoorDesg(String refDoorDesg) {

this.refDoorDesg = refDoorDesg;

}

//getter and setter for refrigerator color

public String getRefColor() {

return refColor;

}

public void setRefColor(String refColor) {

this.refColor = refColor;

}

//getter and setter for refrigerator capacity

public double getRefCap() {

return refCap;

}

public void setRefCap(double refCap) {

this.refCap = refCap;

}

//method to calculate the value of the stock of a refrigerator

public double calStockValue(){

return getqAvai() \* getPrice();

}

//override toString() method and return info of Refrigerator object

@Override

public String toString()

{

return "Item number : " + getItemNum()

+"\nProduct Name : " + getProdName()

+ "\nDoor design : " + refDoorDesg

+ "\nColor : " + refColor

+ "\nCapacity (in Litres) : " + refCap

+ "\nQuantity Available : " + getqAvai()

+ "\nPrice (RM) : " + getPrice()

+ "\nInventory value (RM) : " + calStockValue()

+ "\nProduct Status : " + getProdStat()

+ "\n";

}

}

**TV.java**

public class TV extends Product

{

//initialization of TV screen type, TV resolution and TV display size

private String tvScrTyp;

private String tvRes;

private double tvDisSize;

//parameterized constructor of TV class

public TV(int itemNum, String prodName, int qAvai, double price, Boolean prodStat, String tvScrTyp, String tvRes, double tvDisSize)

{

super(itemNum, prodName, qAvai, price, prodStat);

this.tvScrTyp = tvScrTyp;

this.tvRes = tvRes;

this.tvDisSize = tvDisSize;

}

//getter and setter for TV screen type

public String getTvScrTyp() {

return tvScrTyp;

}

public void setTvScrTyp(String tvScrTyp) {

this.tvScrTyp = tvScrTyp;

}

//getter and setter for TV resolution

public String getTvRes() {

return tvRes;

}

public void setTvRes(String tvRes) {

this.tvRes = tvRes;

}

//getter and setter for TV display size

public double getTvDisSize() {

return tvDisSize;

}

public void setTvDisSize(double tvDisSize) {

this.tvDisSize = tvDisSize;

}

//method to calculate the value of the stock of a TV

public double calStockValue(){

return getqAvai() \* getPrice();

}

//override toString() method and return the info of TV object

@Override

public String toString()

{

return "Item number : " + getItemNum()

+"\nProduct Name : " + getProdName()

+ "\nScreen type : " + tvScrTyp

+ "\nResolution : " + tvRes

+ "\nDisplay size : " + tvDisSize

+ "\nQuantity Available : " + getqAvai()

+ "\nPrice (RM) : " + getPrice()

+ "\nInventory value (RM) : " + calStockValue()

+ "\nProduct Status : " + getProdStat()

+ "\n";

}

}

**AirCond.java**

public class AirCond extends Product{

//initialization of air conditioner horse power, air conditioner refrigerant type and air conditioner voltage

private double acHP;

private String acRefType;

private double acVol;

//parameterized constructor of AirCond class

public AirCond(int itemNum, String prodName, int qAvai, double price, Boolean prodStat, double acHP, String acRefType, double acVol)

{

super(itemNum, prodName, qAvai, price, prodStat);

this.acHP = acHP;

this.acRefType = acRefType;

this.acVol = acVol;

}

//getter and setter for air conditioner horse power

public double getAcHP() {

return acHP;

}

public void setAcHP(double acHP) {

this.acHP = acHP;

}

//getter and setter for air conditioner refrigerant type

public String getAcRefType() {

return acRefType;

}

public void setAcRefType(String acRefType) {

this.acRefType = acRefType;

}

//getter and setter for air conditioner voltage

public double getAcVol() {

return acVol;

}

public void setAcVol(double acVol) {

this.acVol = acVol;

}

//method to calculate the value of the stock of an air conditioner

public double calStockValue(){

return getqAvai() \* getPrice();

}

//override toString() method and return info of AirCong object

@Override

public String toString()

{

return "Item number : " + getItemNum()

+"\nProduct Name : " + getProdName()

+ "\nHorse Power : " + acHP

+ "\nRefrigerant Type : " + acRefType

+ "\nVoltage (V) : " + acVol

+ "\nQuantity Available : " + getqAvai()

+ "\nPrice (RM) : " + getPrice()

+ "\nInventory value (RM) : " + calStockValue()

+ "\nProduct Status : " + getProdStat()

+ "\n";

}

}

**UserInfo.java**

import java.util.Scanner;

public class UserInfo {

private String name;

private String userId;

public UserInfo() {

this.name = "";

this.userId = "guest";

}

public void setName(String name) {

this.name = name;

}

public String getName() {

return name;

}

public String getID() {

return userId;

}

public void setID(String name, boolean space){

if(space){

String[] nameParts = name.split(" ");

String firstName = nameParts[0];

String lastName = nameParts[nameParts.length - 1];

char firstInitial = firstName.charAt(0);

userId = Character.toUpperCase(firstInitial) + lastName;

}

}

public boolean checkSpace(String name) {

return name.contains(" ");

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

UserInfo userInfo = new UserInfo();

System.out.println("Enter your full name (first name and surname):");

String fullName = scanner.nextLine();

userInfo.setName(fullName);

userInfo.setID(userInfo.getName(), userInfo.checkSpace(userInfo.getName()));

if(userInfo.checkSpace(userInfo.getName())){

System.out.println("User ID generated: " + userInfo.getID());

}

else{

System.out.println("Invalid name format. User ID set to default: " + userInfo.getID());

}

scanner.close();

}

}

**StockManagement.java**

import java.text.SimpleDateFormat;

import java.util.Date;

import java.util.InputMismatchException;

import java.util.Scanner;

import javafx.application.Application;

import javafx.application.Platform;

import javafx.geometry.Pos;

import javafx.scene.Scene;

import javafx.scene.control.Button;

import javafx.scene.layout.VBox;

import javafx.scene.text.Font;

import javafx.scene.text.Text;

import javafx.stage.Stage;

public class StockManagement extends Application {

static public int number\_added = 0; //global variable storing number of products added into product array

private static final int FONT\_SIZE = 20;

public static int productToUpdate(Product[] product, Scanner scanner) { //method to list products so users can select one to update

System.out.println("\nProducts List:");

for (int i = 0; i < number\_added; i++) {

System.out.println(i+1 + ": " + product[i].getProdName());

}

System.out.print("Please select the product to update (enter index number): ");

int choice;

while (true) //loops infinitely until a valid input is entered

{

try //Throws InputMismatchException if incorrect data type is entered

{

choice = scanner.nextInt();

while (choice < 0 || choice >= number\_added) { //prompts user to enter number again if the number entered is not within the valid indexes of the array

System.out.print("Invalid index number. Please select a valid index number: ");

choice = scanner.nextInt();

}

break; //exits the infinite while loop when a valid input is entered

}

catch (InputMismatchException ex)

{

scanner.next(); //consumes invalid input

System.out.print("Invalid input. Please select a valid index number: ");

}

}

scanner.nextLine(); // consume the newline character

return choice; //returns the index of the product to update

}

public static void addStock(Product[] product, Scanner scanner) //method to add stocks to a particular product

{

int to\_update = productToUpdate(product, scanner); //grab product index to update from the productToUpdate method

System.out.print("Please enter the quantity to add: ");

int quantity;

try //Throws InputMismatchException if incorrect data type is entered

{

quantity = scanner.nextInt();

}

catch (InputMismatchException ex)// If an invalid data type is entered, an error message will be displayed before the method is forcefully exited

{

scanner.next(); //consumes invalid input

System.out.println("Invalid data entered! Only numbers are accepted!\n");

return;

}

while (quantity < 0) //Prompts user to enter quantity again if they enter a negative value

{

System.out.print("Invalid quantity. Please enter a positive value: ");

quantity = scanner.nextInt();

}

product[to\_update].addStock(quantity);

}

public static void deductStock(Product[] product, Scanner scanner) //method to deduct stocks from a particular product

{

int to\_update = productToUpdate(product, scanner); //grab product index to update from the productToUpdate method

System.out.print("Please enter quantity to deduct: ");

int quantity;

try //Throws InputMismatchException if incorrect data type is entered

{

quantity = scanner.nextInt();

}

catch (InputMismatchException ex) // If an invalid data type is entered, an error message will be displayed before the method is forcefully exited

{

scanner.next(); //consumes invalid input

quantity = 0;

System.out.println("Invalid data entered! Only numbers are accepted!/n");

return;

}

while (quantity < 0 || quantity > product[to\_update].getqAvai()) // Prompts user to enter quantity again if they enter a negative value or a quantity value > current stocks

{

if (quantity < 0)

{

System.out.print("Invalid quantity.\nPlease enter a positive value: ");

quantity = scanner.nextInt();

}

else

{

System.out.print("Failed! Cannot deduct more than stock availability.\nPlease enter a valid quantity: ");

quantity = scanner.nextInt();

}

}

product[to\_update].deductStock(quantity);

}

public static void addProduct(Product[] products, Scanner input) {

System.out.println("\nChoose a product to add\n1. Refrigerator\n2. TV\n3. Air Conditioner");

System.out.print("Enter selection: ");

int choice = getInputChoice(input, 3);

switch (choice) {

case 1:

addRefrigerator(products, input);

break;

case 2:

addTV(products, input);

break;

case 3:

addAirCond(products, input);

break;

}

}

// Utility method to handle choice input safely

public static int getInputChoice(Scanner scanner, int maxOption) {

int choice;

while (true) {

try {

choice = scanner.nextInt();

if (choice >= 1 && choice <= maxOption) {

break;

}

System.out.println("Invalid choice. Please enter a number between 1 and " + maxOption + ".");

} catch (InputMismatchException ex) {

System.out.println("Invalid input. Please enter a numerical value.");

scanner.next(); // Consume the erroneous input

}

}

return choice;

}

public static void addRefrigerator(Product[] products, Scanner input) //method to add Refrigerator object into product array

{

input.nextLine(); // clear input buffer

System.out.print("\nEnter product name: "); //create local variables to be inserted into Refrigerator object

String name = input.nextLine();

System.out.print("Enter door design: ");

String design = input.nextLine();

System.out.print("Enter color: ");

String color = input.nextLine();

System.out.print("Enter capacity: ");

int capacity = input.nextInt();

System.out.print("Enter quantity available in stock: ");

int stock = input.nextInt();

System.out.print("Enter price: ");

double price = input.nextDouble();

System.out.print("Enter item number: ");

int id = input.nextInt();

// create Refrigerator object using user input and add to products array

products[number\_added] = new Refrigerator(id, name, stock, price, true, design, color, capacity);

number\_added++; //increment global variable by 1

System.out.println("Refrigerator " + name + " with item number " + id + " has been added.");

}

public static void addTV(Product[] products, Scanner input)

{

input.nextLine(); //clear input buffer

System.out.print("\nEnter product name: "); //create local variables to be inserted into TV object

String name = input.nextLine();

System.out.print("Enter screen type: ");

String screenType = input.nextLine();

System.out.print("Enter resolution: ");

String resolution = input.nextLine();

System.out.print("Enter display size: ");

double displaySize = input.nextDouble();

System.out.print("Enter quantity available in stock: ");

int stock = input.nextInt();

System.out.print("Enter price: ");

double price = input.nextDouble();

System.out.print("Enter item number: ");

int id = input.nextInt();

products[number\_added] = new TV(id, name, stock, price, true, screenType, resolution, displaySize);

number\_added++; //increment global variable by 1

System.out.println("TV " + name + " with item number " + id + " has been added.");

}

public static void addAirCond(Product[] products, Scanner input)

{

input.nextLine(); // Clear the buffer

System.out.print("\nEnter product name: ");

String name = input.nextLine();

System.out.print("Enter horse power: ");

double hp = input.nextDouble();

System.out.print("Enter refrigerant type: ");

input.nextLine(); // Clear the buffer

String refType = input.nextLine();

System.out.print("Enter voltage: ");

double voltage = input.nextDouble();

System.out.print("Enter quantity available in stock: ");

int stock = input.nextInt();

System.out.print("Enter price: ");

double price = input.nextDouble();

System.out.print("Enter item number: ");

int id = input.nextInt();

products[number\_added] = new AirCond(id, name, stock, price, true, hp, refType, voltage);

number\_added++;

System.out.println("Air Conditioner " + name + " with item number " + id + " has been added.");

}

public static int numberToAdd(Scanner in) //method to prompt user the maximum number of the products they want to add

{

int num;

System.out.print("\nWhat is the maximum number of products you want to add?: ");

do

{

try //Throws InputMismatchException if incorrect data type is entered

{

num = in.nextInt();

}

catch (InputMismatchException ex) //num = -1 if an invalid data type is entered

{

in.next(); //consumes invalid input

num = -1;

}

if (num < 0) //prompts user to enter number again if it is not 0 or above

System.out.print("Invalid input! Please enter a number of 0 and above: ");

}while (num <0);

return(num);

}

public static void discontinueProduct(Product[] products, Scanner in) //method to discontinue a stock in the product array

{

int to\_update;

to\_update = productToUpdate(products, in); //grab product index to update from the productToUpdate method

products[to\_update].setProdStat(false); //set status of the selected product to false

System.out.println("Successfully discontinued " + products[to\_update].getProdName() + "\n");

}

public static void main(String[] args) {

launch(args);

}

@Override

public void start(Stage primaryStage) {

VBox root = new VBox(10);

root.setAlignment(Pos.CENTER);

root.setStyle("-fx-background-color: yellow;");// Set the background color to yellow

// Welcome text at the top with larger font

Text welcomeText = new Text("Welcome to Stock Management System");

welcomeText.setFont(new Font(FONT\_SIZE));

// Current date and time with larger font

SimpleDateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");

Date date = new Date();

Text dateTimeText = new Text(dateFormat.format(date));

dateTimeText.setFont(new Font(FONT\_SIZE));

// Group members in the middle with larger font

Text groupMembersText = new Text(

"Group Members:\nHo Kah Mun\nLer Jun Wei\nLoh Wei Jia\nTan Kai Jun"

);

groupMembersText.setFont(new Font(FONT\_SIZE));

// Button at the bottom with larger font

Button continueButton = new Button("Continue");

continueButton.setFont(new Font(FONT\_SIZE));

continueButton.setOnAction(e -> primaryStage.close());

// Add all components to the VBox

root.getChildren().addAll(welcomeText, dateTimeText, groupMembersText, continueButton);

Scene scene = new Scene(root, 900, 600); // Set the scene size

primaryStage.setTitle("Stock Management System");

primaryStage.setScene(scene);

primaryStage.setOnHidden(e -> Platform.runLater(()->continueInConsole()));

primaryStage.show();

}

public static void continueInConsole() {

System.out.println("Welcome to Stock Management System");

SimpleDateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss"); //create an object to indicate date format

Date date = new Date(); //create a date object

System.out.println(dateFormat.format(date) + "\n"); //display date as indicated by date format

System.out.println("Group Members:");

System.out.println("Ho Kah Mun");

System.out.println("Ler Jun Wei");

System.out.println("Loh Wei Jia");

System.out.println("Tan Kai Jun\n");

UserInfo user = new UserInfo();

Scanner input = new Scanner(System.in);

System.out.println("Enter your full name (first name and surname): ");

String fullName = input.nextLine();

user.setName(fullName);// Pass the full name to setName() method

user.setID(user.getName(), user.checkSpace(user.getName()));

Product[] products = new Product[numberToAdd(input)]; //specify array size

for(int i = 0; i < products.length; i++) //loop to add products into product array

{

int check;

addProduct(products, input);

if (i == (products.length - 1))

System.out.println("Maximum number of products added.\n");

else

{

System.out.print("Continue to add product? (1 - yes, 0 - no): "); //exit loop early if user doesn't want to add more product objects

do

{

try //Throws InputMismatchException if incorrect data type is entered

{

check = input.nextInt();

}

catch (InputMismatchException ex) //num = -1 if an invalid data type is entered

{

input.next(); //consumes invalid input

check = -1;

}

if (check < 0 || check > 1) //Prompts user to enter again if check is not 1 or 0

{

System.out.print("Invalid input! Please enter 1 or 0 (1 - yes, 0 - no): ");

check = input.nextInt();

}

}while(check < 0 || check > 1);

if (check == 0)

{

System.out.println();

break;

}

}

}

menuSelect(menu(input), products, input); //displays menu and prompts user to make selection

System.out.println("\nExited program.");

System.out.println("User ID: " + user.getID());

System.out.println("Username: " + user.getName());

input.close();

}

public static void displayContent(Product[] products, Scanner input) //method to display information about all products in product array

{

int option;

for (int i = 0; i < number\_added; i++)

System.out.println(products[i].toString());

do

{

System.out.println("\n1. Add Product");

System.out.println("2. Update Product");

System.out.println("3. Delete Product");

System.out.println("0. Back to Main Menu");

try

{

option = input.nextInt();

}

catch(InputMismatchException ex)

{

input.next();

option = -1;

}

switch(option)

{

case 0:

menuSelect(menu(input), products, input);

case 1:

addProduct(products, input);

case 2:

updateProduct(products, input);

case 3:

deleteProduct(products, input);

default:

System.out.println("Invalid input! Please enter a number from 0 to 3!"); //prompts user to enter number again if the number entered is not between 0 and 3

}

}while(option < 0 || option > 3);

}

public static void updateProduct(Product[] products, Scanner input)

{

input.nextLine();

int to\_update = productToUpdate(products, input);

Product updateProd = products[to\_update];

System.out.println("Press enter if nothing to edit in that field.");

System.out.println("Enter new item number: ");

String newItemNum = input.nextLine();

if (!newItemNum.isEmpty()) {

updateProd.setItemNum(Integer.parseInt(newItemNum));

}

// Update product name

System.out.println("Enter new product name: ");

String newProdName = input.nextLine();

if (!newProdName.isEmpty()) {

updateProd.setProdName(newProdName);

}

// Update quantity available

System.out.println("Enter new quantity available: ");

String newQAvai = input.nextLine();

if (!newQAvai.isEmpty()) {

updateProd.setqAvai(Integer.parseInt(newQAvai));

}

// Update price

System.out.println("Enter new price: ");

String newPrice = input.nextLine();

if (!newPrice.isEmpty()) {

updateProd.setPrice(Double.parseDouble(newPrice));

}

// Update product status

System.out.println("Enter new product status (true/false): ");

String newProdStat = input.nextLine();

if (!newProdStat.isEmpty()) {

updateProd.setProdStat(Boolean.parseBoolean(newProdStat));

}

// Update specific fields for subclass (Refrigerator, TV, AirCond) if necessary

if (updateProd instanceof Refrigerator)

{

Refrigerator updateRefrigerator = (Refrigerator) updateProd;

System.out.println("Enter new refrigerator door design: ");

String newRefDoorDesg = input.nextLine();

if (!newRefDoorDesg.isEmpty())

{

updateRefrigerator.setRefDoorDesg(newRefDoorDesg);

}

System.out.println("Enter new refrigerator color: ");

String newRefColor = input.nextLine();

if (!newRefColor.isEmpty())

{

updateRefrigerator.setRefColor(newRefColor);

}

System.out.println("Enter new refrigerator capacity: ");

String newRefCap = input.nextLine();

if (!newRefCap.isEmpty())

{

updateRefrigerator.setRefCap(Double.parseDouble(newRefCap));

}

}

else if (updateProd instanceof TV)

{

TV updateTV = (TV) updateProd;

System.out.println("Enter new TV screen type: ");

String newTvScrTyp = input.nextLine();

if (!newTvScrTyp.isEmpty())

{

updateTV.setTvScrTyp(newTvScrTyp);

}

System.out.println("Enter new TV resolution: ");

String newTvRes = input.nextLine();

if (!newTvRes.isEmpty())

{

updateTV.setTvRes(newTvRes);

}

System.out.println("Enter new TV display size: ");

String newTvDisSize = input.nextLine();

if (!newTvDisSize.isEmpty())

{

updateTV.setTvDisSize(Double.parseDouble(newTvDisSize));

}

}

else if (updateProd instanceof AirCond)

{

AirCond updateAirCond = (AirCond) updateProd;

System.out.println("Enter new Air Conditioner horsepower: ");

String newAcHP = input.nextLine();

if (!newAcHP.isEmpty())

{

updateAirCond.setAcHP(Double.parseDouble(newAcHP));

}

System.out.println("Enter new Air Conditioner horsepower: ");

String newAcRefType = input.nextLine();

if (!newAcRefType.isEmpty())

{

updateAirCond.setAcHP(Double.parseDouble(newAcRefType));

}

System.out.println("Enter new Air Conditioner horsepower: ");

String newAcVol = input.nextLine();

if (!newAcVol.isEmpty())

{

updateAirCond.setAcHP(Double.parseDouble(newAcVol));

}

}

}

public static void deleteProduct(Product[] products, Scanner input)

{

int to\_delete = productToUpdate(products, input);

String deletedName = products[to\_delete].getProdName();

for (int i = to\_delete; i < products.length - 1; i++)

{

products[i] = products[i + 1];

}

System.out.println("Product with the name " + deletedName + " has been deleted successfullly.");

}

public static int menu(Scanner scanner) { //Displays a menu for users to execute other methods from

int choice;

do {

System.out.println("1. View products");

System.out.println("2. Add stock");

System.out.println("3. Deduct stock");

System.out.println("4. Discontinue product");

System.out.println("0. Exit");

System.out.print("Please enter a menu option: ");

try //Throws InputMismatchException if incorrect data type is entered

{

choice = scanner.nextInt();

}

catch (InputMismatchException ex) //choice = -1 if an invalid data type is entered

{

scanner.next(); //consumes invalid input

choice = -1;

}

if (choice < 0 || choice > 4)

System.out.println("Invalid input! Please enter a number from 0 to 4!"); //prompts user to enter number again if the number entered is not between 0 and 4

} while (choice < 0 || choice > 4);

return choice; //returns selection

}

public static void menuSelect(int menu, Product[] products, Scanner in) //method to execute method based on input from menu method

{

do {

switch (menu) { //switch...case statement to execute methods based on input in menu method

case 1:

if (products.length == 0)

System.out.println("There are no products to display!\n");

else

displayContent(products, in);

menu = menu(in);

break;

case 2:

if (products.length == 0)

System.out.println("There are no products to update!\n");

else

addStock(products, in);

menu = menu(in);

break;

case 3:

if (products.length == 0)

System.out.println("There are no products to update!\n");

else

deductStock(products, in);

menu = menu(in);

break;

case 4:

if (products.length == 0)

System.out.println("There are no products to update!\n");

else

discontinueProduct(products, in);

menu = menu(in);

break;

case 0:

System.out.println("Exiting...");

break;

}

} while (menu != 0); //Loops the method until 0 is input

}

}