1. A retail banking system where customers perform various transactions, such as depositing money, withdrawing funds, or transferring funds between accounts. You are tasked with developing the C# application for the bank.

**Functional Requirement:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Req. #** | **Requirements Description** | **Class Name** | **Method Name** | **Parameters** | **Description** |
| **1** | Add transaction details to the queue in the program class. | **Transaction** | AddTransactionToQueue | Transaction transaction | This method **adds**the transactions to the **transactionQueue**property, which is implemented as Queue. (already given in Program class) |
| **2** | This method handles transactions sequentially, ensuring they are processed in the order they were received. | **Transaction** | ProcessTransactions | - | This method returns the string value.  **Constraints:**   * This method should retrieve the transactions in first in first out order from **transactionQueue**. * When the transaction amount is less than zero, return **"Enter the valid amount"** * When the amount is greater than zero return **"Transaction <Transaction Number> processed. Account Holder: <Account Holder Name>, Amount: <Transaction Amount>"** |

**Sample Input/Output 1:**

Enter the number of transactions

**3**

Enter the details (transaction number, name, amount)

**111**

**John**

**1000**

Transaction 111 processed. Account Holder: John, Amount: 1000

Enter the details (transaction number, name, amount)

**222**

**Lary**

**500**

Transaction 222 processed. Account Holder: Lary, Amount: 500

Enter the details (transaction number, name, amount)

**333**

**David**

**100**

Transaction 333 processed. Account Holder: David, Amount: 100

All transactions processed.

**Sample input/Output 2:**

Enter the number of transactions

**1**

Enter the details (transaction number, name, amount)

**111**

**John**

**-11000**

Enter the valid amount

**Answer:-**

using System;

using System.Collections.Generic;

public class Transaction

{

public int TransactionNumber { get; set; }

public string AccountHolderName { get; set; }

public decimal Amount { get; set; }

}

public class Program

{

private static Queue<Transaction> transactionQueue = new Queue<Transaction>();

public static void Main(string[] args)

{

Console.WriteLine("Enter the number of transactions");

int numberOfTransactions = int.Parse(Console.ReadLine());

for (int i = 0; i < numberOfTransactions; i++)

{

Console.WriteLine("Enter the details (transaction number, name, amount)");

Transaction transaction = new Transaction

{

TransactionNumber = int.Parse(Console.ReadLine()),

AccountHolderName = Console.ReadLine(),

Amount = decimal.Parse(Console.ReadLine())

};

AddTransactionToQueue(transaction);

}

Console.WriteLine(ProcessTransactions());

}

public static void AddTransactionToQueue(Transaction transaction)

{

transactionQueue.Enqueue(transaction);

}

public static string ProcessTransactions()

{

while (transactionQueue.Count > 0)

{

Transaction transaction = transactionQueue.Dequeue();

if (transaction.Amount < 0)

{

return "Enter the valid amount";

}

else

{

Console.WriteLine($"Transaction {transaction.TransactionNumber} processed. Account Holder: {transaction.AccountHolderName}, Amount: {transaction.Amount}");

}

}

return "All transactions processed.";

}

}

2. A professional writer is using a text editor to compose an article. They type a paragraph and decide to delete the last sentence. Later, they realize they want to revert the changes and undo the deletion.You are tasked with developing the C# application for them to make their work easy.

**Functional Requirement:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Req. #** | **Requirements Description** | **Class Name** | **Method Name** | **Parameters** | **Description** |
| **1** | Delete the last word if needed | **TextEditor** | DeleteLastCharacter | string text | This method Adds the text given by the user to **undoStack**which is implemented as a stack property in the program class(already provided) and deletes the last character of the sentence entered by the user when they need to delete.  **Constraints:**   * This method returns the result in string format as **<result>\_** |
| **2** | Undo the deletion if needed | **TextEditor** | UndoDeletion | - | This method undo the deletion of the last character when they need to undo and return the string.  **Constraints:**   * Retrieve the previously deleted text from the stack property in the Program class. * Set the retrieved value to the **text**variable. * Return the modified text. |

**Sample Input/Output 1:**

Enter the text

**Hello Everyone!!! I am Robot**

Enter yes, if you need to delete the last letter in your sentence

**yes**

Your sentence after deletion of last letter is :

Hello Everyone!!! I am Robo\_

Enter yes, if you need to undo the last deleted letter

**yes**

Your sentence after undoing the deletion is :

Hello Everyone!!! I am Robot

**Sample Input/output 2:**

Enter the text

**Hello Everyone!!! I am Robot**

Enter yes, if you need to delete the last letter in your sentence

**No**

Your editing completed

**Answer:-**

using System;

using System.Collections.Generic;

public class TextEditor

{

private static Stack<string> undoStack = new Stack<string>();

public static void Main(string[] args)

{

Console.WriteLine("Enter the text");

string text = Console.ReadLine();

Console.WriteLine("Enter yes, if you need to delete the last letter in your sentence");

string deleteResponse = Console.ReadLine();

if (deleteResponse.ToLower() == "yes")

{

text = DeleteLastCharacter(text);

Console.WriteLine("Your sentence after deletion of last letter is :");

Console.WriteLine(text);

}

Console.WriteLine("Enter yes, if you need to undo the last deleted letter");

string undoResponse = Console.ReadLine();

if (undoResponse.ToLower() == "yes")

{

text = UndoDeletion();

Console.WriteLine("Your sentence after undoing the deletion is :");

Console.WriteLine(text);

}

else

{

Console.WriteLine("Your editing completed");

}

}

public static string DeleteLastCharacter(string text)

{

undoStack.Push(text);

if (text.Length > 0)

{

text = text.Substring(0, text.Length - 1);

}

return text + "\_";

}

public static string UndoDeletion()

{

if (undoStack.Count > 0)

{

return undoStack.Pop();

}

return string.Empty;

}

}

3. DigiVerse, a reputable numerical exploration platform, is set to revolutionize data analysis with the development of a C# program. The Numeric Harmony Analyzer within DigiVerse enables users to input numeric sequences, applying criteria to identify even numbers for squaring and odd numbers for cube operations, unveiling hidden patterns in the digital landscape.

**Constraints:**

* The sequence should be numbers and space-separated; otherwise, print "**Invalid sequence**".

**Note:**

* Do not edit the existing code template.
* In the Sample Input / Output provided, the highlighted text in bold corresponds to the input given by the user, and the rest of the text represents the output.
* Implement the business requirements within the main method. Please do not change the class name.
* Do not use **Environment.Exit()** to terminate the program.

**Sample Input / Output 1**

Enter the numeric sequence (space-separated):

**2 5 8 12 17**

4 125 64 144 4913

**Sample Input / Output 2**

Enter the numeric sequence (space-separated):

**1 2 3 a**

Invalid sequence

**Answer:-**

using System;

using System.Linq;

public class NumericHarmonyAnalyzer

{

public static void Main(string[] args)

{

Console.WriteLine("Enter the numeric sequence (space-separated):");

string input = Console.ReadLine();

// Validate the input sequence

if (!IsValidSequence(input))

{

Console.WriteLine("Invalid sequence");

return;

}

// Process the sequence

string result = ProcessSequence(input);

Console.WriteLine(result);

}

private static bool IsValidSequence(string input)

{

// Check if all elements are numbers

return input.Split(' ').All(element => int.TryParse(element, out \_));

}

private static string ProcessSequence(string input)

{

var numbers = input.Split(' ').Select(int.Parse);

var results = numbers.Select(n => n % 2 == 0 ? Math.Pow(n, 2) : Math.Pow(n, 3));

return string.Join(" ", results);

}

}

-------------------------

4. In the heart of an industrial district, Modern Textiles Inc. stands as a pioneering manufacturer of high-quality synthetic fibers. As a software consultant specializing in process optimization, you've been tasked with developing a C# application to streamline the synthesis of nylon, ensuring efficiency and precision in the manufacturing process.

**Functional Requirements:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Req. #** | **Requirements Description** | **Class Name** | **Method Name** | **Parameters** | **Description** |
| **1** | Validate the synthesis parameters entered by the user | **SynthesisUtility** | ValidateSynthesis | - | This method validates the details entered by the user.  The valid specifications include:   |  |  | | --- | --- | | **Temperature** | Between 1 and 300 (inclusive) | | **Pressure** | Between 1 and 10 (inclusive) | | **Reaction time** | Greater than 0 |   **Constraint:**   * This method returns **true**when the specifications are valid; otherwise, it returns **false** |
| 2 | Execute the synthesis process and provide feedback to the user | **SynthesisUtility** | PerformSynthesis | - | This method returns the int value.  **Constraints:**   * Synthesis calculation is performed by multiplying the temperature and pressure * Divide the above-resulting value by the reaction time. |

**SynthesisUtility**classneeds to refer the class.

**Sample Input 1:**

Enter the Primary Monomer:

**Adipoyl chloride**

Enter the Secondary Monomer:

**Hexamethylene diamine**

Enter the Reaction Temperature (in Celsius):

**250**

Enter the Reaction Pressure (in atm):

**5**

Enter the Catalyst:

**Sulfuric Acid**

Enter the Reaction Duration (in minutes):

**60**

**Sample Output 1:**

Performing synthesis of Nylon using Adipoyl chloride and Hexamethylene diamine...

Applying 250 degree Celsius and 5 atm pressure with Sulfuric Acid catalyst...

Reacting for 60 minutes...

The nylon synthesis results 20 as calculated value...

Nylon synthesis completed successfully!

**Sample Input 2:**

Enter the Primary Monomer:

**Adipoyl chloride**

Enter the Secondary Monomer:

**Hexamethylene diamine**

Enter the Reaction Temperature (in Celsius):

**350**

Enter the Reaction Pressure (in atm):

**12**

Enter the Catalyst:

**Sulfuric Acid**

Enter the Reaction Duration (in minutes):

**0**

**Sample Output 2:**

Enter the valid specifications

**Answer:-**

using System;

public class SynthesisUtility

{

public static void Main(string[] args)

{

Console.WriteLine("Enter the Primary Monomer:");

string primaryMonomer = Console.ReadLine();

Console.WriteLine("Enter the Secondary Monomer:");

string secondaryMonomer = Console.ReadLine();

Console.WriteLine("Enter the Reaction Temperature (in Celsius):");

int temperature = int.Parse(Console.ReadLine());

Console.WriteLine("Enter the Reaction Pressure (in atm):");

int pressure = int.Parse(Console.ReadLine());

Console.WriteLine("Enter the Catalyst:");

string catalyst = Console.ReadLine();

Console.WriteLine("Enter the Reaction Duration (in minutes):");

int reactionTime = int.Parse(Console.ReadLine());

if (ValidateSynthesis(temperature, pressure, reactionTime))

{

int result = PerformSynthesis(temperature, pressure, reactionTime);

Console.WriteLine($"Performing synthesis of Nylon using {primaryMonomer} and {secondaryMonomer}...");

Console.WriteLine($"Applying {temperature} degree Celsius and {pressure} atm pressure with {catalyst} catalyst...");

Console.WriteLine($"Reacting for {reactionTime} minutes...");

Console.WriteLine($"The nylon synthesis results {result} as calculated value...");

Console.WriteLine("Nylon synthesis completed successfully!");

}

else

{

Console.WriteLine("Enter the valid specifications");

}

}

public static bool ValidateSynthesis(int temperature, int pressure, int reactionTime)

{

return (temperature >= 1 && temperature <= 300) &&

(pressure >= 1 && pressure <= 10) &&

(reactionTime > 0);

}

public static int PerformSynthesis(int temperature, int pressure, int reactionTime)

{

return (temperature \* pressure) / reactionTime;

}}

5. A prestigious film studio is embarking on its next major project: a trilogy of films spanning various genres. To streamline the production process and accurately estimate costs, they've sought the assistance of a software developer. Your objective is to develop a C# application that validates movie specifications and computes the production cost for each movie in the trilogy.

**Functional Requirements:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Req. #** | **Requirements Description** | **Class Name** | **Method Name** | **Parameters** | **Description** |
| **1** | Validate the movie specifications entered by the user | **MovieUtility** | ValidateMovieSpecification | string genre, double mainActorSalary | This method is used to **validate**the movie specifications.  **Valid**Movie Specifications:   * Genre should be "**Action**", "**Drama**" or "". * The Main Actor's salary should be **greater**than **0**.   **Constraints:**   * When the specifications are valid, return **true;** Otherwise, return **false**. * The movie genre is **case-sensitive**. |
| **2** | Calculate the production cost of the movie using genre, main actor's salary. | **MovieUtility** | CalculateProductionCost | string genre, double mainActorSalary | This method calculates the production cost and returns double value.   |  |  |  | | --- | --- | --- | | **Genre** | **Cost (per genre)** | **Actor Salary** | | Action | 500000 | Three times of main actor's salary | | Drama | 300000 | N/A | | Comedy | 200000 | N/A |     **Constraints:**   * The production cost is calculated by adding the genre cost and the actor's salary(for available genre). |

**Sample Input 1:**

Enter Movie Genre (Action/Drama/Comedy):

**Action**

Enter Main Actor's Salary:

**100000**

**Sample Output 1:**

Production Cost: 800000

**Sample Input 2:**

Enter Movie Genre (Action/Drama/Comedy):

**drama**

Enter Main Actor's Salary:

**100000**

**Sample Output 2:**

Invalid movie specifications

**Answer:-**

using System;

public class MovieUtility

{

public static void Main(string[] args)

{

Console.WriteLine("Enter Movie Genre (Action/Drama/Comedy):");

string genre = Console.ReadLine();

Console.WriteLine("Enter Main Actor's Salary:");

double mainActorSalary = double.Parse(Console.ReadLine());

if (ValidateMovieSpecification(genre, mainActorSalary))

{

double productionCost = CalculateProductionCost(genre, mainActorSalary);

Console.WriteLine($"Production Cost: {productionCost}");

}

else

{

Console.WriteLine("Invalid movie specifications");

}

}

public static bool ValidateMovieSpecification(string genre, double mainActorSalary)

{

return (genre == "Action" || genre == "Drama" || genre == "Comedy") && mainActorSalary > 0;

}

public static double CalculateProductionCost(string genre, double mainActorSalary)

{

double genreCost = 0;

double actorCost = 0;

switch (genre)

{

case "Action":

genreCost = 500000;

actorCost = 3 \* mainActorSalary;

break;

case "Drama":

genreCost = 300000;

break;

case "Comedy":

genreCost = 200000;

break;

}

return genreCost + actorCost;

}

}

6. Jessica is a manager at a retail store and wants to create a program to track the sales performance of her employees. Write a C# program to assist Jessica in finding the first day when an employee meets or exceeds a certain sales target.

**For example**, consider a scenario where an employee's daily sales for a week are $100, $120, $90, $150, $110, $130, and $140 respectively. Jessica wants to determine the first day when the employee's sales reach or exceed $150. In this scenario, the first qualifying day is determined to be Day 4 because the employee's sales reach $150 on that day.

**Constraints:**

* When the employee meets or exceeds the sales target on any day, display "**Congratulations! The employee met or exceeded the sales target of $<targetAmount> on the <qualifyingDay>-day**".
* When no day meets or exceeds the sales target, display "**No qualifying day found".**

**Note:**

* Do not edit the existing code template.
* In the Sample Input / Output provided, the highlighted text in bold corresponds to the input given by the user, and the rest of the text represents the output.
* Implement the business requirements within the main method. Please do not change the class name.
* Do not use **Environment.Exit()** to terminate the program.

**Sample Input / Output 1**

Enter the number of days

**6**

Enter the sales amount for Day 1

**120**

Enter the sales amount for Day 2

**130**

Enter the sales amount for Day 3

**170**

Enter the sales amount for Day 4

**140**

Enter the sales amount for Day 5

**- 3**

Enter the sales amount for Day 6

**200**

Enter the sales target

**160**

Congratulations! The employee met or exceeded the sales target of $160 on the 3-day

**Sample Input / Output 2**

Enter the number of days

**4**

Enter the sales amount for Day 1

**50**

Enter the sales amount for Day 2

**80**

Enter the sales amount for Day 3

**90**

Enter the sales amount for Day 4

**40**

Enter the sales target

**100**

No qualifying day found

**Answer:-**

using System;

public class SalesTracker

{

public static void Main(string[] args)

{

Console.WriteLine("Enter the number of days");

int numberOfDays = int.Parse(Console.ReadLine());

int[] sales = new int[numberOfDays];

for (int i = 0; i < numberOfDays; i++)

{

Console.WriteLine($"Enter the sales amount for Day {i + 1}");

sales[i] = int.Parse(Console.ReadLine());

}

Console.WriteLine("Enter the sales target");

int salesTarget = int.Parse(Console.ReadLine());

bool targetMet = false;

for (int i = 0; i < numberOfDays; i++)

{

if (sales[i] >= salesTarget)

{

Console.WriteLine($"Congratulations! The employee met or exceeded the sales target of ${salesTarget} on the {i + 1}-day");

targetMet = true;

break;

}

}

if (!targetMet)

{

Console.WriteLine("No qualifying day found");

}

}

}

**ADO.Net**

1. 1. Insert the player details in the database.

2. 2. Get the player details by player id from the database.

3. 3. Calculate the body fat percentage of the player.

Sample Data is Already inserted in the Player\_Info table.

Id Name Age Height Weight Gender

1 Sam 23 175.00 65.50 Male

2 David 24 168.00 58.00 Male

3 Peter 23 180.00 75.50 Male

4 Rosy 23 155.00 50.00 Female

We need to go with these 3 methods:

----------------------------------------------------

1. Insert the player details in the database

Id Name Age Height Weight Gender

2. Get the player details by player id

3. Calculating the body fat percentage Body Fat Percentage =

For Male (1.20 \* BMI) + (0.23 \* age) - 16.2;

For Female (1.20 \* BMI) + (0.23 \* age) - 5.4;

BMI = Weight / ((Height\*Height)/10000);

Class Name : Player

Create a Class Player with the following public Properties.

Data Type Variable Name

int Id

string Name

int Age

float Height

float Weight

string Gender

Class Name : PlayerUtility

Create a Class PlayerUtility to use the below methods and functions:

1 AddPlayerDetails Method

Type(Class) Method

PlayerUtility public bool AddPlayerDetails(Player playerObj)

This method is used to insert the player details into the database. If the player details are added in the database, then print“Player details added successfully”

2. public List GetPlayerDetailsByPlayerId(int playerId)

This method is used to get the player details from the database based on the player id. Add the player details to the List and return it.

3. public double FindBodyFatPercentage(int age,float height, float weight,string gender)

This method is used to calculate the bodyfat percentage based on the age,height, weight and gender.The percentage should have two decimal places. The age must bebetween20and 28 (both inclusive),else return 0. The gender must be Male or Female, else return0.

**Answer:-**

public class Player

{

public int Id { get; set; }

public string Name { get; set; }

public int Age { get; set; }

public float Height { get; set; }

public float Weight { get; set; }

public string Gender { get; set; }

}

using System;

using System.Collections.Generic;

public class PlayerUtility

{

private static List<Player> playerDatabase = new List<Player>();

// Method to add player details

public bool AddPlayerDetails(Player playerObj)

{

playerDatabase.Add(playerObj);

Console.WriteLine("Player details added successfully");

return true;

}

// Method to get player details by player ID

public List<Player> GetPlayerDetailsByPlayerId(int playerId)

{

List<Player> result = new List<Player>();

foreach (var player in playerDatabase)

{

if (player.Id == playerId)

{

result.Add(player);

}

}

return result;

}

// Method to calculate body fat percentage

public double FindBodyFatPercentage(int age, float height, float weight, string gender)

{

if (age < 20 || age > 28 || (gender != "Male" && gender != "Female"))

{

return 0;

}

double bmi = weight / ((height \* height) / 10000);

double bodyFat = gender == "Male"

? (1.20 \* bmi) + (0.23 \* age) - 16.2

: (1.20 \* bmi) + (0.23 \* age) - 5.4;

return Math.Round(bodyFat, 2);

}

}

using System;

public class Program

{

public static void Main(string[] args)

{

PlayerUtility utility = new PlayerUtility();

// Adding sample data

utility.AddPlayerDetails(new Player { Id = 1, Name = "Sam", Age = 23, Height = 175, Weight = 65.5f, Gender = "Male" });

utility.AddPlayerDetails(new Player { Id = 2, Name = "David", Age = 24, Height = 168, Weight = 58, Gender = "Male" });

// Fetching details by ID

Console.WriteLine("Enter Player ID to fetch details:");

int playerId = int.Parse(Console.ReadLine());

var playerDetails = utility.GetPlayerDetailsByPlayerId(playerId);

foreach (var player in playerDetails)

{

Console.WriteLine($"Player: {player.Name}, Age: {player.Age}, Height: {player.Height}, Weight: {player.Weight}, Gender: {player.Gender}");

}

// Calculating Body Fat

Console.WriteLine("Calculating Body Fat...");

double bodyFat = utility.FindBodyFatPercentage(23, 175, 65.5f, "Male");

Console.WriteLine($"Body Fat Percentage: {bodyFat}%");

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

2. Get data and insert into the database. In the ‘Main’ get the following values,

MedicineId

MedicineName

PharmacyName

Quantity

TotalPrice

DeliveryStatus

2 Get medicineId and update Details from database.

The User enters the medicineId then it should update delivery status of the medicine that matches the medicineId.

Example: If medicineId is 1 and then it should update the information of DeliveryStatus.

3 Calculate Total Price

The User enters quantity then, the method should calculate the total charges and return it. If quantity is between 50 to 99(inclusive) then the discount is 5% and charge of the medicine is INR 52. If quantity is between 100 to 199(inclusive) then the discount is 7% and charge of the medicine is INR 52. If quantity is between 200 to 499(inclusive) then the discount is 10% and charge of the medicine is INR 52.

If quantity is greater than or equal to 500 then the discount is 10% and charge of the medicine is INR 52. If quantity is less than 50 then the no discount Example: quantity = 100 price = 100 \* 52 Discount is 7% off from the price Total Charge = 4836

UpdateDeliveryStatus Method

public MedicineDetails UpdateDeliveryStatus (int medicineId, string deliveryStatus)

This method should accept an medicine id and deliverystatus then execute an sqlquery to update the details into the database then return the updatedetails and store the details in Medicine Details object and return that object

RetrieveAllDetails Method

Public List RetriveAllDetails ()

This method should retrieve all the medicine details of the medicine that is in the database. Retrieve all the details and store it in a List and return it

CalculateTotalPrice Method

public double CalulateTotalPrice (int quantity)

This method is used to calculate the totalprice based on the quantity. calculate the totalprice and return it as datatype double.

**Answer:-**

public class MedicineDetails

{

public int MedicineId { get; set; }

public string MedicineName { get; set; }

public string PharmacyName { get; set; }

public int Quantity { get; set; }

public double TotalPrice { get; set; }

public string DeliveryStatus { get; set; }

}

using System;

using System.Collections.Generic;

public class MedicineUtility

{

private static List<MedicineDetails> medicineDatabase = new List<MedicineDetails>();

// Method to update delivery status

public MedicineDetails UpdateDeliveryStatus(int medicineId, string deliveryStatus)

{

foreach (var medicine in medicineDatabase)

{

if (medicine.MedicineId == medicineId)

{

medicine.DeliveryStatus = deliveryStatus;

return medicine;

}

}

return null;

}

// Method to retrieve all details

public List<MedicineDetails> RetrieveAllDetails()

{

return medicineDatabase;

}

// Method to calculate total price

public double CalculateTotalPrice(int quantity)

{

double pricePerUnit = 52;

double discount = 0;

if (quantity >= 50 && quantity <= 99) discount = 0.05;

else if (quantity >= 100 && quantity <= 199) discount = 0.07;

else if (quantity >= 200) discount = 0.10;

double totalPrice = quantity \* pricePerUnit;

totalPrice -= totalPrice \* discount;

return Math.Round(totalPrice, 2);

}

}

using System;

public class Program

{

public static void Main(string[] args)

{

MedicineUtility utility = new MedicineUtility();

// Adding sample data

utility.RetrieveAllDetails().Add(new MedicineDetails { MedicineId = 1, MedicineName = "Paracetamol", PharmacyName = "MediCare", Quantity = 100, TotalPrice = 5200, DeliveryStatus = "Pending" });

// Updating Delivery Status

Console.WriteLine("Enter Medicine ID to update delivery status:");

int medicineId = int.Parse(Console.ReadLine());

Console.WriteLine("Enter New Delivery Status:");

string status = Console.ReadLine();

var updatedMedicine = utility.UpdateDeliveryStatus(medicineId, status);

if (updatedMedicine != null)

{

Console.WriteLine($"Updated Medicine: {updatedMedicine.MedicineName}, Status: {updatedMedicine.DeliveryStatus}");

}

else

{

Console.WriteLine("Medicine not found!");

}

// Calculating Total Price

Console.WriteLine("Enter Quantity to calculate total price:");

int quantity = int.Parse(Console.ReadLine());

double totalPrice = utility.CalculateTotalPrice(quantity);

Console.WriteLine($"Total Price: {totalPrice}");

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*