**1 .Scenario:**

Techwing Solutions is a prominent software company in the city that prioritizes workplace safety for its employees. As part of their safety measures, the company collects vaccination status information from their employees, and even incentivizes those who have not yet been vaccinated to get vaccinated by offering rewards. To effectively manage this process, the company is in need of software that can efficiently handle and organize employee vaccination details.

As their software consultant, you can help them by developing a C# application.

**Functionalities:**

In class **Program,** implement the below-given method.

**public static Dictionary<string, string> vaccinationDetails**-In the code template, it is already provided.

implement the features listed below.

|  |  |
| --- | --- |
| **Method** | **Description** |
| public void AddVaccinationDetails(Dictionary<string, string> dic) | In this method pass the employee id and their vaccination status as dictionary.  Using this method add the details into the **vaccinationDetails**dictionary**.** |
| public List<String> GetEmployeeId(string status) | This method is used to get the employees id based on the vaccination status.  Then store the employees id into the **List**and return it |
| public string UpdateVaccinationStatus(string id, string status) | This method is used to update the vaccination status based on the employee id and return employee id with vaccination status in the given format.  Employee id : **Emp1**  Status : **Fully Vaccinated**  **Emp1\_Fully Vaccinated**  In between add**underscore(\_).** |

In **Program** class, **Main** method,

**1.**Get the values from the **user**as per the Sample Input.

**2.**Call the methods accordingly and display the result.

**3.**     In the Sample Input / Output provided, the highlighted text in bold corresponds to the input given by the user and the remaining text represents the output.

**Note:**

* Keep the method and class as **public.**
* Please read the method rules **clearly**.
* Do not use **Environment.Exit()** to terminate the program.
* Do not change the given code template.

**Sample Input / Output:**

1. Add the vaccination details

2. Get employee id by status

3. Update vaccination details

4. Exit

Enter your choice

**1**

Enter the employee id

**EMP1**

Enter the status

**Fully Vaccinated**

1. Add the vaccination details

2. Get employee id by status

3. Update vaccination details

4. Exit

Enter your choice

**1**

Enter the employee id

**EMP2**

Enter the status

**Partial Vaccinated**

1. Add the vaccination details

2. Get employee id by status

3. Update vaccination details

4. Exit

Enter your choice

**1**

Enter the employee id

**EMP3**

Enter the status

**Not Vaccinated**

1. Add the vaccination details

2. Get employee id by status

3. Update vaccination details

4. Exit

Enter your choice

**2**

Enter the status

**Fully Vaccinated**

EMP1

1. Add the vaccination details

2. Get employee id by status

3. Update vaccination details

4. Exit

Enter your choice

**3**

Enter the employee id

**EMP2**

Enter the status

**Fully Vaccinated**

EMP2\_Fully Vaccinated

1. Add the vaccination details

2. Get employee id by status

3. Update vaccination details

4. Exit

Enter your choice

**4**

Thank You

2.

using System;

using System.Collections.Generic;

public class Program

{

public static Dictionary<string, string> vaccinationDetails = new Dictionary<string, string>();

public static void AddVaccinationDetails(string id, string status)

{

vaccinationDetails[id] = status;

}

public static List<String> GetEmployeeId(string status)

{

List<string> employeeIds = new List<string>();

foreach (var detail in vaccinationDetails)

{

if (detail.Value == status)

{

employeeIds.Add(detail.Key);

}

}

return employeeIds;

}

public static string UpdateVaccinationStatus(string id, string status)

{

vaccinationDetails[id] = status;

return id + "\_" + status;

}

public static void Main(string[] args)

{

while (true)

{

Console.WriteLine("1. Add the vaccination details");

Console.WriteLine("2. Get employee id by status");

Console.WriteLine("3. Update vaccination details");

Console.WriteLine("4. Exit");

Console.WriteLine("Enter your choice");

int choice = Convert.ToInt32(Console.ReadLine());

switch (choice)

{

case 1:

Console.WriteLine("Enter the employee id");

string id = Console.ReadLine();

Console.WriteLine("Enter the status");

string status = Console.ReadLine();

AddVaccinationDetails(id, status);

break;

case 2:

Console.WriteLine("Enter the status");

string statusToSearch = Console.ReadLine();

List<string> ids = GetEmployeeId(statusToSearch);

foreach (string employeeId in ids)

{

Console.WriteLine(employeeId);

}

break;

case 3:

Console.WriteLine("Enter the employee id");

string idToUpdate = Console.ReadLine();

Console.WriteLine("Enter the status");

string newStatus = Console.ReadLine();

string result = UpdateVaccinationStatus(idToUpdate, newStatus);

Console.WriteLine(result);

break;

case 4:

Console.WriteLine("Thank You");

return;

}

}

}

}

**Scenario:**

ConTV is dedicated to being the epitome of user-centric television companies, continually surpassing expectations through groundbreaking advancements in software, hardware, and services. In their relentless pursuit of customer satisfaction, ConTV actively seeks feedback on the TV installation process. To further enhance their understanding of customer sentiments, they are in need of a sophisticated software solution capable of calculating ratings based on feedback comments.

 As their software consultant, you can help them by developing a C# application.

**Functionalities:**

In class **Installation,**implement the below-given properties.

|  |  |
| --- | --- |
| **Class** | **Properties** |
| Installation | String ExpectedDate     String InstalledDate     double Rating     String Feedback |

In class **InstallationDetails,**implement the below-given methods and also **Inherit** the class **Installation**.

|  |  |
| --- | --- |
| **Method** | **Description** |
| public void GetCustomerFeedback() | This method is used to get the feedback from the customer based on the expected date and installed date.  If the installed date is less than expected date, then assign "**VeryGood**" in the **Feedback**property.  If the expected and installed dates are the same, then assign "**Good**" in the **Feedback**property.  If the installed date is within three days from expected date, then assign "**Average**" in the **Feedback**property.  If the installed date is more than three days from expected date, then assign "**Poor**" in the **Feedback**property. |
| public double CalculateRating() | This method is used to calculate the updated rating based on the feedback and return it.  If the rating is not between **1**and **10**then return **0**.  The calculation conditions are given below. |

* If the feed back comment is **VeryGood,**then add 50% of the rating to the rating.
* If the feed back comment is **Good,**then add 20% of the rating to the rating.
* If the feed back comment is **Average,**then subtract 20% of the rating to the rating.
* If the feed back comment is **Poor,**then subtract 50% of the rating to the rating.

**Note:**

Feedbackis **Case-Sensitive**.

The date format is **(MM/dd/yyyy).**

In **Program** class - **Main** method,

**1.**Get the values from the **user**as per the Sample Input

**2.**Call the method **GetCustomerFeedback.**

**3.**Call the method**CalculateRating,**if it returns **0**then display**'Invalid rating'**in Main method. Else display the result as per the Sample Output

**Note:**

* Keep the properties, methods and classes as **public.**
* Please read the method rules **clearly**.
* Do not use **Environment.Exit()** to terminate the program.
* Do not change the given code template.

**Sample Input 1:**

Enter the expected date

**05/25/2023**

Enter the installed date

**05/20/2023**

Enter the rating

**5**

**Sample Output 1:**

Now your rating is 7.5

**Sample Input 2:**

Enter the expected date

**05/20/2023**

Enter the installed date

**05/27/2023**

Enter the rating

**7**

**Sample Output 2:**

Now your rating is 3.5

**Sample Input 3:**

Enter the expected date

**04/14/2023**

Enter the installed date

**04/17/2023**

Enter the rating

**15**

**Sample Output 3:**

Invalid rating

using System;

using System.Globalization;

public class Installation

{

public string ExpectedDate { get; set; }

public string InstalledDate { get; set; }

public double Rating { get; set; }

public string Feedback { get; set; }

}

public class InstallationDetails : Installation

{

public void GetCustomerFeedback()

{

DateTime expectedDate = DateTime.ParseExact(ExpectedDate, "MM/dd/yyyy", CultureInfo.InvariantCulture);

DateTime installedDate = DateTime.ParseExact(InstalledDate, "MM/dd/yyyy", CultureInfo.InvariantCulture);

if (installedDate < expectedDate)

Feedback = "VeryGood";

else if (installedDate == expectedDate)

Feedback = "Good";

else if (installedDate <= expectedDate.AddDays(3))

Feedback = "Average";

else

Feedback = "Poor";

}

public double CalculateRating()

{

if (Rating < 1 || Rating > 10)

return 0;

switch (Feedback)

{

case "VeryGood":

Rating += Rating \* 0.5;

break;

case "Good":

Rating += Rating \* 0.2;

break;

case "Average":

Rating -= Rating \* 0.2;

break;

case "Poor":

Rating -= Rating \* 0.5;

break;

}

return Rating;

}

}

public class Program

{

public static void Main(string[] args)

{

InstallationDetails installationDetails = new InstallationDetails();

Console.WriteLine("Enter the expected date");

installationDetails.ExpectedDate = Console.ReadLine();

Console.WriteLine("Enter the installed date");

installationDetails.InstalledDate = Console.ReadLine();

Console.WriteLine("Enter the rating");

installationDetails.Rating = double.Parse(Console.ReadLine());

installationDetails.GetCustomerFeedback();

double updatedRating = installationDetails.CalculateRating();

if (updatedRating == 0)

Console.WriteLine("Invalid rating");

else

Console.WriteLine($"Now your rating is {updatedRating}");

}

}

3.

**Scenario:**

The Public Cricket Academy is a facility where young or aspiring cricketers can go to receive training and coaching in the sport. These academies typically have experienced coaches on staff who provide instruction on various aspects of the game, such as batting, bowling, fielding, and strategy. The coach of the Public Cricket Academy (PCA)wants to store the players run details and calculate the average runs secured by individual players on his team.

 As their software consultant, you can help them by developing a C# application.

**Functionalities:**

In class **Program,** implement the below-given method.

**public static List<int> playerList**-In the code template, it is already provided.

implement the features listed below.

|  |  |
| --- | --- |
| **Method** | **Description** |
| public void AddScoreDetails(int runScored) | This method is used to add the player's **runScored**passed as an argumentinto the **playerList.** |
| public double GetAverageRunsSecured() | This method is used to calculate the average run secured by a player and return it.  Average runs can be calculated based on the sum of all run scores available in the **playerList**and dividing it by the number of elements in the**playerList.**  If the **playerList** is empty this method should return**0.**  The average run should be in two decimal places.  **Hint :** Use **Math.Floor()** |

In **Program** class, **Main** method,

**1.**Get the values from the **user**.

**2.**Call the methods accordingly and display the result.

**3.**     In the Sample Input / Output provided, the highlighted text in bold corresponds to the input given by the user and the remaining text represents the output.

**Note:**

* Keep the method and class as **public.**
* Please read the method rules **clearly**.
* Do not use **Environment.Exit()** to terminate the program.
* Do not change the given code template.

**Sample Input / Output:**

1**.**Add Runs Scored

2. Average Score of the Player

3. Exit

Enter your choice

**1**

Enter the runs scored

**150**

1. Add Runs Scored

2. Average Score of the Player

3. Exit

Enter your choice

**1**

Enter the runs scored

**50**

1. Add Runs Scored

2. Average Score of the Player

3. Exit

Enter your choice

**1**

Enter the runs scored

**50**

1. Add player Details

2. Average Score for a Player

3. Exit

Enter your choice

**2**

Average runs secured: 83.33

1. Add player Details

2. Average Score for a Player

3. Exit

Enter your choice

**3**

Thank You

using System;

using System.Collections.Generic;

public class Program

{

public static List<int> playerList = new List<int>();

public static void AddScoreDetails(int runScored)

{

playerList.Add(runScored);

}

public static double GetAverageRunsSecured()

{

if (playerList.Count == 0)

return 0;

double sum = 0;

foreach (int score in playerList)

{

sum += score;

}

return Math.Floor((sum / playerList.Count) \* 100) / 100;

}

public static void Main(string[] args)

{

while (true)

{

Console.WriteLine("1. Add Runs Scored");

Console.WriteLine("2. Average Score of the Player");

Console.WriteLine("3. Exit");

Console.WriteLine("Enter your choice");

int choice = Convert.ToInt32(Console.ReadLine());

switch (choice)

{

case 1:

Console.WriteLine("Enter the runs scored");

int runScored = Convert.ToInt32(Console.ReadLine());

AddScoreDetails(runScored);

break;

case 2:

double averageScore = GetAverageRunsSecured();

Console.WriteLine($"Average runs secured: {averageScore}");

break;

case 3:

Console.WriteLine("Thank You");

return;

}

}

}

}

4.

**Scenario:**

Food Hub is the world's largest food service provider. They provide a fun and safe environment where our customers can enjoy good food made with quality ingredients at affordable prices. They serve happiness to our customers through delicious, quality meals and an extraordinary restaurant experience while working toward the greater good for our employees, community, and environment. They need software to manage their hotel bills.

As their software consultant, you can help them by developing a C# application.

**Functionalities:**

In class **HotelBill**, implement the below-given properties.

|  |  |
| --- | --- |
| **Data Type** | **Properties** |
| string | BillNo |
| double | BillAmount |
| string | Status |

In class **Program,** implement the below-given method.

**public static List<HotelBill> billList**-In the code template, it is already provided.

implement the features listed below.

|  |  |
| --- | --- |
| **Method** | **Description** |
| public List<HotelBill> GetBillDetails(string billNo) | This method is used to get the bill details based on the bill number.  If the **billNo**is available in the **billList ,**itshould return that item as **List**.  If the **billNo**is not available in  **billList ,** then return an empty **List.**  If this method returns an **empty List**, then print **"Invalid bill number**" in **Main()**method. |
| public List<HotelBill> UpdateBillStatus(string billNo,string status) | This method is used to update the bill **status**based on the **billNo**.  If the **billNo** is available in the **billList ,**itshould return that updated item as **List**.  If the **billNo** is not available in  **billList ,** then return an empty **List.**  If this method returns an **empty List**, then print **"Invalid bill number**" in **Main()**method. |
| public List<HotelBill> SortBillByStatus() | This method is used to display all the bill details available in the **billList**in descending order by bill status.  The result should be **List**. |

In **Program** class, **Main** method,

**1.**Get the values from the **user**.

**2.**Call the methods accordingly and display the result.

**3.**     In the Sample Input / Output provided, the highlighted text in bold corresponds to the input given by the user and the remaining text represents the output.

**Note:**

* Keep the method and class as **public.**
* Please read the method rules **clearly**.
* Do not use **Environment.Exit()** to terminate the program.
* Do not change the given code template.

**Sample Input / Output:**

1. Get bill details

2. Update Bill Status

3. Sort Bill By Status

4. Exit

Enter your choice

**1**

Enter bill number

**H01**

|  |  |  |
| --- | --- | --- |
| BillNo | BillAmount | Status |
| H01 | 450 | UnPaid |

1. Get bill details

2. Update Bill Status

3. Sort Bill By Status

4. Exit

Enter your choice

**2**

Enter bill number

**H01**

Enter bill status

**Paid**

|  |  |  |
| --- | --- | --- |
| BillNo | BillAmount | Status |
| H01 | 450 | Paid |

1. Get bill details

2. Update Bill Status

3. Sort Bill By Status

4. Exit

Enter your choice

**2**

Enter bill number

**H08**

Enter bill status

**Paid**

Invalid bill number

1. Get bill details

2. Update Bill Status

3. Sort Bill By Status

4. Exit

Enter your choice

**3**

|  |  |  |
| --- | --- | --- |
| BillNo | BillAmount | Status |
| H04 | 2000 | UnPaid |
| H01 | 450 | Paid |
| H02 | 50 | Paid |
| H03 | 500 | Paid |

1. Get bill details

2. Update Bill Status

3. Sort Bill By Status

4. Exit

Enter your choice

**4**Thank You

using System;

using System.Collections.Generic;

using System.Linq;

public class HotelBill

{

public string BillNo { get; set; }

public double BillAmount { get; set; }

public string Status { get; set; }

}

public class Program

{

public static List<HotelBill> billList = new List<HotelBill>();

public static List<HotelBill> GetBillDetails(string billNo)

{

return billList.Where(bill => bill.BillNo == billNo).ToList();

}

public static List<HotelBill> UpdateBillStatus(string billNo, string status)

{

var billToUpdate = billList.FirstOrDefault(bill => bill.BillNo == billNo);

if (billToUpdate != null)

{

billToUpdate.Status = status;

return new List<HotelBill> { billToUpdate };

}

return new List<HotelBill>();

}

public static List<HotelBill> SortBillByStatus()

{

return billList.OrderByDescending(bill => bill.Status).ToList();

}

public static void Main(string[] args)

{

while (true)

{

Console.WriteLine("1. Get bill details");

Console.WriteLine("2. Update Bill Status");

Console.WriteLine("3. Sort Bill By Status");

Console.WriteLine("4. Exit");

Console.WriteLine("Enter your choice");

int choice = Convert.ToInt32(Console.ReadLine());

switch (choice)

{

case 1:

Console.WriteLine("Enter bill number");

string billNo = Console.ReadLine();

var billDetails = GetBillDetails(billNo);

if (billDetails.Count == 0)

{

Console.WriteLine("Invalid bill number");

}

else

{

foreach (var bill in billDetails)

{

Console.WriteLine($"BillNo: {bill.BillNo}, BillAmount: {bill.BillAmount}, Status: {bill.Status}");

}

}

break;

case 2:

Console.WriteLine("Enter bill number");

string billNoToUpdate = Console.ReadLine();

Console.WriteLine("Enter bill status");

string newStatus = Console.ReadLine();

var updatedBillDetails = UpdateBillStatus(billNoToUpdate, newStatus);

if (updatedBillDetails.Count == 0)

{

Console.WriteLine("Invalid bill number");

}

else

{

foreach (var bill in updatedBillDetails)

{

Console.WriteLine($"BillNo: {bill.BillNo}, BillAmount: {bill.BillAmount}, Status: {bill.Status}");

}

}

break;

case 3:

var sortedBills = SortBillByStatus();

foreach (var bill in sortedBills)

{

Console.WriteLine($"BillNo: {bill.BillNo}, BillAmount: {bill.BillAmount}, Status: {bill.Status}");

}

break;

case 4:

Console.WriteLine("Thank You");

return;

}

}

}

}

5.

**Scenario:**

Water Wonders is a private water supplier company that specializes in providing drinking water to customers. They operate through a network of pipelines or delivery trucks, sourcing water from wells or other surface water sources. Prior to distribution, the company ensures that the water is treated to meet health and safety standards. However, they encountered challenges when it came to manually calculating water prices, prompting the need for software to streamline their billing processes.

 As their software consultant, you need to help them by developing a C# application.

**Functionalities:**

In class **WaterBill,**implement the below-given properties.

|  |  |
| --- | --- |
| **Datatype** | **Property Name** |
| string     double     double | UsageType     Volume     PricePerGallon |

In class **BillUtility,**implement the below-given methods and also **Inherit** the class **WaterBill**.

|  |  |
| --- | --- |
| **Method** | **Description** |
| public bool ValidateUsageType() | This method is used to validate the water usage type.  If the usage type is **Residential**or **Commercial**or **Industrial**, then return **true**. Otherwise return **false.** |
| public double WaterPriceCalculation() | This method is used to calculate the water price with tax amount and return it.  The calculation procedures are given below. |

**Formula :**

Water Price Calculation = Price + Tax amount

Price = Volume of water \* Price per gallon

|  |  |
| --- | --- |
| **Usage Type** | **Tax amount** |
| Residential | 10% of the price |
| Commercial | 20% of the price |
| Industrial | 30% of the price |

**Note: Usage type**is **Case-sensitive**.

In **Program** class - **Main** method,

**1.**Get the values from the **user**.

**2.**Call the **ValidateUsageType**method, If it returns true then get the **Volume**, and **Price Per Gallon**value from the **user**and move on to step 3, If it returns false then display  **Invalid usage type.**

**3.**Use the values in method**WaterPriceCalculation.**

**Note:**

* Keep the properties, methods and classes as **public.**
* Please read the method rules **clearly**.
* Do not use **Environment.Exit()** to terminate the program.
* Do not change the given code template.

**Sample Input 1:**

Enter the usage type

**Industrial**

Enter the volume of water

**1000**

Enter price per gallon

**30**

**Sample Output 1:**

The total price is 39000

**Sample Input 2:**

Enter the usage type

**Home**

**Sample Output 2:**

Invalid usage type

using System;

public class WaterBill

{

public string UsageType { get; set; }

public double Volume { get; set; }

public double PricePerGallon { get; set; }

}

public class BillUtility : WaterBill

{

public bool ValidateUsageType()

{

return UsageType == "Residential" || UsageType == "Commercial" || UsageType == "Industrial";

}

public double WaterPriceCalculation()

{

double price = Volume \* PricePerGallon;

double taxAmount = 0;

switch (UsageType)

{

case "Residential":

taxAmount = price \* 0.1;

break;

case "Commercial":

taxAmount = price \* 0.2;

break;

case "Industrial":

taxAmount = price \* 0.3;

break;

}

return price + taxAmount;

}

}

public class Program

{

public static void Main(string[] args)

{

BillUtility billUtility = new BillUtility();

Console.WriteLine("Enter the usage type");

billUtility.UsageType = Console.ReadLine();

if (!billUtility.ValidateUsageType())

{

Console.WriteLine("Invalid usage type");

return;

}

Console.WriteLine("Enter the volume of water");

billUtility.Volume = Convert.ToDouble(Console.ReadLine());

Console.WriteLine("Enter price per gallon");

billUtility.PricePerGallon = Convert.ToDouble(Console.ReadLine());

double totalPrice = billUtility.WaterPriceCalculation();

Console.WriteLine($"The total price is {totalPrice}");

}

}

**Scenario:**

Lisa, an aspiring entrepreneur, envisions establishing her own home appliance company. She recently acquired a small room located in the bustling city center to kickstart her venture. To initiate her business operations, Lisa decided to begin with selling washers and dryers. Unfortunately, due to the limited visibility of her newly opened shop, the sales of washers and dryers have been sluggish.

In an effort to enhance her shop's popularity and propel her business to new heights, Lisa devises an ingenious strategy. She decides to introduce a bulk offer for the washers and dryers, offering them at a discounted price. To facilitate this enticing deal, Lisa recognizes the need for software that can accurately calculate the discounted amount for each unit.

As their software consultant, you can help her by developing a C# application

**Functionalities:**

In class **Machine,**implement the below-given properties.

|  |  |
| --- | --- |
| **Class** | **Properties** |
| Machine | string Type     double CubicFootCapacity     double NoOfRevolution     double NoOfSeconds     double Price |

In class **MachineDetails,**implement the below-given methods and also **Inherit** class **Machine**.

|  |  |  |
| --- | --- | --- |
| **Class** | **Method** | **Description** |
| MachineDetails | public double FindTheSpinSpeed() | This method is used to calculate the spin speed of the washing machine and return it.  **Formula :**  Spin Speed (RPM) = (60 x (Number of revolutions)) / (Number of seconds) |
| MachineDetails | public double CalculatePrice() | This method is used to calculate the price of the washing machine after the discount and return it.  Refer to the below procedures to calculate price after the discount. |

**Formula :**

Price after discount = Price - Discount

|  |  |  |
| --- | --- | --- |
| **Type** | **Cubic Foot Capacity** | **Discount** |
| Front Load | Between 2.0 and 5.5      Above 5.5      Less than 2.0 | 20% of the price      40% of the price      No discount |
| Top Load | Between 2.0 and 5.5      Above 5.5      Less than 2.0 | 30% of the price      50% of the price      No discount |

**Note :** Type is **case sensitive.**

In **Program** class - **Main** method,

**1.**Get the values from the **user**as per the Sample Input.

**2.** Call the methods **FindTheSpinSpeed**and **CalculatePrice**

**3.**Display the result as per the Sample Output

**Note:**

* Keep the properties, methods and classes as **public.**
* Please read the method rules **clearly**.
* Do not use **Environment.Exit()** to terminate the program.
* Do not change the given code template.

**Sample Input 1:**

 Enter the load type

**Front Load**

Enter cubic foot capacity

**2.5**

Enter number of revolution

**400**

Enter number of seconds

**20**

Enter the price

**15000**

**Sample Output 1:**

Spin speed is 1200 RPM

The price after discount 12000

**Sample Input 2:**

Enter the load type

**Top Load**

Enter cubic foot capacity

**1.8**

Enter number of revolution

**300**

Enter number of seconds

**30**

Enter the price

**10000**

**Sample Output 2:**

Spin speed is 600 RPM

The price after discount 10000

using System;

public class Machine

{

public string Type { get; set; }

public double CubicFootCapacity { get; set; }

public double NoOfRevolution { get; set; }

public double NoOfSeconds { get; set; }

public double Price { get; set; }

}

public class MachineDetails : Machine

{

public double FindTheSpinSpeed()

{

return (60 \* NoOfRevolution) / NoOfSeconds;

}

public double CalculatePrice()

{

double discount = 0;

if (Type == "Front Load")

{

if (CubicFootCapacity < 2.0)

discount = 0;

else if (CubicFootCapacity <= 5.5)

discount = 0.2 \* Price;

else

discount = 0.4 \* Price;

}

else if (Type == "Top Load")

{

if (CubicFootCapacity < 2.0)

discount = 0;

else if (CubicFootCapacity <= 5.5)

discount = 0.3 \* Price;

else

discount = 0.5 \* Price;

}

return Price - discount;

}

}

class Program

{

static void Main(string[] args)

{

MachineDetails machine = new MachineDetails();

Console.WriteLine("Enter the load type");

machine.Type = Console.ReadLine();

Console.WriteLine("Enter cubic foot capacity");

machine.CubicFootCapacity = double.Parse(Console.ReadLine());

Console.WriteLine("Enter number of revolution");

machine.NoOfRevolution = double.Parse(Console.ReadLine());

Console.WriteLine("Enter number of seconds");

machine.NoOfSeconds = double.Parse(Console.ReadLine());

Console.WriteLine("Enter the price");

machine.Price = double.Parse(Console.ReadLine());

double spinSpeed = machine.FindTheSpinSpeed();

Console.WriteLine($"Spin speed is {spinSpeed} RPM");

double priceAfterDiscount = machine.CalculatePrice();

Console.WriteLine($"The price after discount {priceAfterDiscount}");

}

}

**Scenario:**

Food Hub is the world's largest food service provider. They provide a fun and safe environment where our customers can enjoy good food made with quality ingredients at affordable prices. They serve happiness to our customers through delicious, quality meals and an extraordinary restaurant experience while working toward the greater good for our employees, community, and environment. They need software to manage their hotel bills.

As their software consultant, you can help them by developing a C# application.

**Functionalities:**

In class **HotelBill**, implement the below-given properties.

|  |  |
| --- | --- |
| **Data Type** | **Properties** |
| string | BillNo |
| double | BillAmount |
| string | Status |

In class **Program,** implement the below-given method.

**public static List<HotelBill> billList**-In the code template, it is already provided.

implement the features listed below.

|  |  |
| --- | --- |
| **Method** | **Description** |
| public List<HotelBill> GetBillDetails(string billNo) | This method is used to get the bill details based on the bill number.  If the **billNo**is available in the **billList ,**itshould return that item as **List**.  If the **billNo**is not available in  **billList ,** then return an empty **List.**  If this method returns an **empty List**, then print **"Invalid bill number**" in **Main()**method. |
| public List<HotelBill> UpdateBillStatus(string billNo,string status) | This method is used to update the bill **status**based on the **billNo**.  If the **billNo** is available in the **billList ,**itshould return that updated item as **List**.  If the **billNo** is not available in  **billList ,** then return an empty **List.**  If this method returns an **empty List**, then print **"Invalid bill number**" in **Main()**method. |
| public List<HotelBill> SortBillByStatus() | This method is used to display all the bill details available in the **billList**in descending order by bill status.  The result should be **List**. |

In **Program** class, **Main** method,

**1.**Get the values from the **user**.

**2.**Call the methods accordingly and display the result.

**3.**     In the Sample Input / Output provided, the highlighted text in bold corresponds to the input given by the user and the remaining text represents the output.

**Note:**

* Keep the method and class as **public.**
* Please read the method rules **clearly**.
* Do not use **Environment.Exit()** to terminate the program.
* Do not change the given code template.

**Sample Input / Output:**

1. Get bill details

2. Update Bill Status

3. Sort Bill By Status

4. Exit

Enter your choice

**1**

Enter bill number

**H01**

|  |  |  |
| --- | --- | --- |
| BillNo | BillAmount | Status |
| H01 | 450 | UnPaid |

1. Get bill details

2. Update Bill Status

3. Sort Bill By Status

4. Exit

Enter your choice

**2**

Enter bill number

**H01**

Enter bill status

**Paid**

|  |  |  |
| --- | --- | --- |
| BillNo | BillAmount | Status |
| H01 | 450 | Paid |

1. Get bill details

2. Update Bill Status

3. Sort Bill By Status

4. Exit

Enter your choice

**2**

Enter bill number

**H08**

Enter bill status

**Paid**

Invalid bill number

1. Get bill details

2. Update Bill Status

3. Sort Bill By Status

4. Exit

Enter your choice

**3**

|  |  |  |
| --- | --- | --- |
| BillNo | BillAmount | Status |
| H04 | 2000 | UnPaid |
| H01 | 450 | Paid |
| H02 | 50 | Paid |
| H03 | 500 | Paid |

1. Get bill details

2. Update Bill Status

3. Sort Bill By Status

4. Exit

Enter your choice

**4**Thank You

**using System;**

**using System.Collections.Generic;**

**using System.Linq;**

**public class HotelBill**

**{**

**public string BillNo { get; set; }**

**public double BillAmount { get; set; }**

**public string Status { get; set; }**

**}**

**class Program**

**{**

**public static List<HotelBill> billList = new List<HotelBill>();**

**public static List<HotelBill> GetBillDetails(string billNo)**

**{**

**return billList.Where(bill => bill.BillNo == billNo).ToList();**

**}**

**public static List<HotelBill> UpdateBillStatus(string billNo, string status)**

**{**

**var bill = billList.FirstOrDefault(bill => bill.BillNo == billNo);**

**if (bill != null)**

**{**

**bill.Status = status;**

**return new List<HotelBill> { bill };**

**}**

**return new List<HotelBill>();**

**}**

**public static List<HotelBill> SortBillByStatus()**

**{**

**return billList.OrderByDescending(bill => bill.Status).ToList();**

**}**

**static void Main(string[] args)**

**{**

**while (true)**

**{**

**Console.WriteLine("1. Get bill details");**

**Console.WriteLine("2. Update Bill Status");**

**Console.WriteLine("3. Sort Bill By Status");**

**Console.WriteLine("4. Exit");**

**Console.WriteLine("Enter your choice");**

**int choice = int.Parse(Console.ReadLine());**

**switch (choice)**

**{**

**case 1:**

**Console.WriteLine("Enter bill number");**

**string billNo = Console.ReadLine();**

**var billDetails = GetBillDetails(billNo);**

**if (billDetails.Count == 0)**

**{**

**Console.WriteLine("Invalid bill number");**

**}**

**else**

**{**

**foreach (var bill in billDetails)**

**{**

**Console.WriteLine($"BillNo: {bill.BillNo}, BillAmount: {bill.BillAmount}, Status: {bill.Status}");**

**}**

**}**

**break;**

**case 2:**

**Console.WriteLine("Enter bill number");**

**billNo = Console.ReadLine();**

**Console.WriteLine("Enter bill status");**

**string status = Console.ReadLine();**

**var updatedBill = UpdateBillStatus(billNo, status);**

**if (updatedBill.Count == 0)**

**{**

**Console.WriteLine("Invalid bill number");**

**}**

**else**

**{**

**foreach (var bill in updatedBill)**

**{**

**Console.WriteLine($"BillNo: {bill.BillNo}, BillAmount: {bill.BillAmount}, Status: {bill.Status}");**

**}**

**}**

**break;**

**case 3:**

**var sortedBills = SortBillByStatus();**

**foreach (var bill in sortedBills)**

**{**

**Console.WriteLine($"BillNo: {bill.BillNo}, BillAmount: {bill.BillAmount}, Status: {bill.Status}");**

**}**

**break;**

**case 4:**

**return;**

**default:**

**Console.WriteLine("Invalid choice");**

**break;**

**}**

**}**

**}**

**}**

**Scenario:**

Water Wonders is a private water supplier company that specializes in providing drinking water to customers. They operate through a network of pipelines or delivery trucks, sourcing water from wells or other surface water sources. Prior to distribution, the company ensures that the water is treated to meet health and safety standards. However, they encountered challenges when it came to manually calculating water prices, prompting the need for software to streamline their billing processes.

 As their software consultant, you need to help them by developing a C# application.

**Functionalities:**

In class **WaterBill,**implement the below-given properties.

|  |  |
| --- | --- |
| **Datatype** | **Property Name** |
| string     double     double | UsageType     Volume     PricePerGallon |

In class **BillUtility,**implement the below-given methods and also **Inherit** the class **WaterBill**.

|  |  |
| --- | --- |
| **Method** | **Description** |
| public bool ValidateUsageType() | This method is used to validate the water usage type.  If the usage type is **Residential**or **Commercial**or **Industrial**, then return **true**. Otherwise return **false.** |
| public double WaterPriceCalculation() | This method is used to calculate the water price with tax amount and return it.  The calculation procedures are given below. |

**Formula :**

Water Price Calculation = Price + Tax amount

Price = Volume of water \* Price per gallon

|  |  |
| --- | --- |
| **Usage Type** | **Tax amount** |
| Residential | 10% of the price |
| Commercial | 20% of the price |
| Industrial | 30% of the price |

**Note: Usage type**is **Case-sensitive**.

In **Program** class - **Main** method,

**1.**Get the values from the **user**.

**2.**Call the **ValidateUsageType**method, If it returns true then get the **Volume**, and **Price Per Gallon**value from the **user**and move on to step 3, If it returns false then display  **Invalid usage type.**

**3.**Use the values in method**WaterPriceCalculation.**

**Note:**

* Keep the properties, methods and classes as **public.**
* Please read the method rules **clearly**.
* Do not use **Environment.Exit()** to terminate the program.
* Do not change the given code template.

**Sample Input 1:**

Enter the usage type

**Industrial**

Enter the volume of water

**1000**

Enter price per gallon

**30**

**Sample Output 1:**

The total price is 39000

**Sample Input 2:**

Enter the usage type

**Home**

**Sample Output 2:**

Invalid usage type

using System;

using System.Collections.Generic;

public class WaterBill

{

public string UsageType { get; set; }

public double Volume { get; set; }

public double PricePerGallon { get; set; }

}

public class BillUtility : WaterBill

{

public bool ValidateUsageType()

{

return UsageType == "Residential" || UsageType == "Commercial" || UsageType == "Industrial";

}

public double WaterPriceCalculation()

{

double price = Volume \* PricePerGallon;

double taxAmount = 0;

switch (UsageType)

{

case "Residential":

taxAmount = 0.1 \* price;

break;

case "Commercial":

taxAmount = 0.2 \* price;

break;

case "Industrial":

taxAmount = 0.3 \* price;

break;

}

return price + taxAmount;

}

}

class Program

{

static void Main(string[] args)

{

BillUtility bill = new BillUtility();

Console.WriteLine("Enter the usage type");

bill.UsageType = Console.ReadLine();

if (!bill.ValidateUsageType())

{

Console.WriteLine("Invalid usage type");

return;

}

Console.WriteLine("Enter the volume of water");

bill.Volume = double.Parse(Console.ReadLine());

Console.WriteLine("Enter price per gallon");

bill.PricePerGallon = double.Parse(Console.ReadLine());

double totalPrice = bill.WaterPriceCalculation();

Console.WriteLine($"The total price is {totalPrice}");

}

}

**Scenario:**

Lisa, an aspiring entrepreneur, envisions establishing her own home appliance company. She recently acquired a small room located in the bustling city center to kickstart her venture. To initiate her business operations, Lisa decided to begin with selling washers and dryers. Unfortunately, due to the limited visibility of her newly opened shop, the sales of washers and dryers have been sluggish.

In an effort to enhance her shop's popularity and propel her business to new heights, Lisa devises an ingenious strategy. She decides to introduce a bulk offer for the washers and dryers, offering them at a discounted price. To facilitate this enticing deal, Lisa recognizes the need for software that can accurately calculate the discounted amount for each unit.

As their software consultant, you can help her by developing a C# application

**Functionalities:**

In class **Machine,**implement the below-given properties.

|  |  |
| --- | --- |
| **Class** | **Properties** |
| Machine | string Type     double CubicFootCapacity     double NoOfRevolution     double NoOfSeconds     double Price |

In class **MachineDetails,**implement the below-given methods and also **Inherit** class **Machine**.

|  |  |  |
| --- | --- | --- |
| **Class** | **Method** | **Description** |
| MachineDetails | public double FindTheSpinSpeed() | This method is used to calculate the spin speed of the washing machine and return it.  **Formula :**  Spin Speed (RPM) = (60 x (Number of revolutions)) / (Number of seconds) |
| MachineDetails | public double CalculatePrice() | This method is used to calculate the price of the washing machine after the discount and return it.  Refer to the below procedures to calculate price after the discount. |

**Formula :**

Price after discount = Price - Discount

|  |  |  |
| --- | --- | --- |
| **Type** | **Cubic Foot Capacity** | **Discount** |
| Front Load | Between 2.0 and 5.5      Above 5.5      Less than 2.0 | 20% of the price      40% of the price      No discount |
| Top Load | Between 2.0 and 5.5      Above 5.5      Less than 2.0 | 30% of the price      50% of the price      No discount |

**Note :** Type is **case sensitive.**

In **Program** class - **Main** method,

**1.**Get the values from the **user**as per the Sample Input.

**2.** Call the methods **FindTheSpinSpeed**and **CalculatePrice**

**3.**Display the result as per the Sample Output

**Note:**

* Keep the properties, methods and classes as **public.**
* Please read the method rules **clearly**.
* Do not use **Environment.Exit()** to terminate the program.
* Do not change the given code template.

**Sample Input 1:**

 Enter the load type

**Front Load**

Enter cubic foot capacity

**2.5**

Enter number of revolution

**400**

Enter number of seconds

**20**

Enter the price

**15000**

**Sample Output 1:**

Spin speed is 1200 RPM

The price after discount 12000

**Sample Input 2:**

Enter the load type

**Top Load**

Enter cubic foot capacity

**1.8**

Enter number of revolution

**300**

Enter number of seconds

**30**

Enter the price

using System;

public class Machine

{

public string Type { get; set; }

public double CubicFootCapacity { get; set; }

public double NoOfRevolution { get; set; }

public double NoOfSeconds { get; set; }

public double Price { get; set; }

}

public class MachineDetails : Machine

{

public double FindTheSpinSpeed()

{

return (60 \* NoOfRevolution) / NoOfSeconds;

}

public double CalculatePrice()

{

double discount = 0;

if (Type == "Front Load")

{

if (CubicFootCapacity < 2.0)

discount = 0;

else if (CubicFootCapacity <= 5.5)

discount = 0.2 \* Price;

else

discount = 0.4 \* Price;

}

else if (Type == "Top Load")

{

if (CubicFootCapacity < 2.0)

discount = 0;

else if (CubicFootCapacity <= 5.5)

discount = 0.3 \* Price;

else

discount = 0.5 \* Price;

}

return Price - discount;

}

}

class Program

{

static void Main(string[] args)

{

MachineDetails machine = new MachineDetails();

Console.WriteLine("Enter the load type");

machine.Type = Console.ReadLine();

Console.WriteLine("Enter cubic foot capacity");

machine.CubicFootCapacity = double.Parse(Console.ReadLine());

Console.WriteLine("Enter number of revolution");

machine.NoOfRevolution = double.Parse(Console.ReadLine());

Console.WriteLine("Enter number of seconds");

machine.NoOfSeconds = double.Parse(Console.ReadLine());

Console.WriteLine("Enter the price");

machine.Price = double.Parse(Console.ReadLine());

double spinSpeed = machine.FindTheSpinSpeed();

Console.WriteLine($"Spin speed is {spinSpeed} RPM");

double priceAfterDiscount = machine.CalculatePrice();

Console.WriteLine($"The price after discount {priceAfterDiscount}");

}

}

**Scenario:**

Crazy Frank Cakes is a small bakery that specializes in custom cakes for weddings, birthdays, and other special occasions. The owner, Maria, has been running the business for the past 5 years, and while it has been successful, she feels like it has reached a plateau. She wants to take the business to the next level, but she is not sure how to do it. One day, Maria decides to hire a consultant to develop a marketing plan and create a website to promote the new services.

As their software consultant, you need to help her by developing a C# application

**Functionalities:**

In class **Cake,**implement the below-given properties.

|  |  |
| --- | --- |
| **Class** | **Properties** |
| Cake | string CakeType    string Flavour    int Quantity    int PricePerKg    double TotalPrice |

In class **CakeUtility,**implement the below-given methods and also **Inherit** class **Cake**.

|  |  |
| --- | --- |
| **Method** | **Description** |
| public bool ValidateCakeType() | This method is used to validate the cake type.  If the cake type  is **Butter**or **Sponge**or **Chiffon**, then return **true**.  Otherwise return **false**, and Also display **Invalid cake type** in Main method. |
| public string[ ] CalculatePrice() | This method is used to calculate cost of the cake after the discount based on the below given formula.  Then store the details such as **CakeType**, **Flavour**, **Quantity**, **PricePerKg**and **TotalPrice**in **string array**and return it. |

**Formula :**

Total Price = (Quantity \* Price Per Kg ) - Discount

|  |  |
| --- | --- |
| **Flavour** | **Discount Percentage** |
| Vanilla | 5% of Total Price |
| Chocolate | 10% of Total Price |
| Others | 0%   of Total Price |

**Note:**

**CakeType** and **Flavour**is **Case-sensitive**.

In **Program** class - **Main** method,

**1.**Get the **CakeType**, **Flavour**, **Quantity**, and **PricePerKg**value from the **user**.

**2.**    Call the **ValidateCakeType**method and if it returns true then move on to step 3, otherwise display **Invalid cake type**in the Main method

**3.**Use the values in method **CalculatePrice** and display the result as per the Sample Output.

**Note:**

* Keep the properties, methods and classes as **public.**
* Please read the method rules **clearly**.
* Do not use **Environment.Exit()** to terminate the program.
* Do not change the given code template.

**Sample Input 1:**

Enter the cake type

**Butter**

Enter the flavour

**Chocolate**

Enter the quantity

**5**

Enter the price per kg

**200**

**Sample Output 1:**

Cake Type : Butter

Cake Flavour : Chocolate

Quantity : 5

Price Per Kg : 200

Total Price : 900

**Sample Input 2:**

Enter the cake type

**Chiffon**

Enter the flavour

**Strawberry**

Enter the quantity

**10**

Enter the price per kg

**150**

**Sample Output 2:**

Cake Type : Chiffon

Cake Flavour : Strawberry

Quantity : 10

Price Per Kg : 150

Total Price : 1500

**Sample Input 3:**

Enter the cake type

**Lemon**

Enter the flavour

**Chocolate**

Enter the quantity

**1**

Enter the price per kg

**180**

**Sample Output 3:**

Invalid cake type

public class Cake

{

public string CakeType { get; set; }

public string Flavour { get; set; }

public int Quantity { get; set; }

public int PricePerKg { get; set; }

public double TotalPrice { get; set; }

}

public class CakeUtility : Cake

{

public bool ValidateCakeType()

{

return CakeType == "Butter" || CakeType == "Sponge" || CakeType == "Chiffon";

}

public string[] CalculatePrice()

{

double discount = 0;

if (Flavour == "Vanilla")

discount = 0.05;

else if (Flavour == "Chocolate")

discount = 0.10;

TotalPrice = (Quantity \* PricePerKg) \* (1 - discount);

return new string[] { CakeType, Flavour, Quantity.ToString(), PricePerKg.ToString(), TotalPrice.ToString() };

}

}

class Program

{

static void Main(string[] args)

{

CakeUtility cake = new CakeUtility();

Console.WriteLine("Enter the cake type");

cake.CakeType = Console.ReadLine();

if (!cake.ValidateCakeType())

{

Console.WriteLine("Invalid cake type");

return;

}

Console.WriteLine("Enter the flavour");

cake.Flavour = Console.ReadLine();

Console.WriteLine("Enter the quantity");

cake.Quantity = int.Parse(Console.ReadLine());

Console.WriteLine("Enter the price per kg");

cake.PricePerKg = int.Parse(Console.ReadLine());

string[] result = cake.CalculatePrice();

Console.WriteLine($"Cake Type : {result[0]}");

Console.WriteLine($"Cake Flavour : {result[1]}");

Console.WriteLine($"Quantity : {result[2]}");

Console.WriteLine($"Price Per Kg : {result[3]}");

Console.WriteLine($"Total Price : {result[4]}");

}

}

**Scenario:**

Retire Hub is a leading entertainment club in the city. In their last board meeting, board members decided to collect the membership fee for improving and maintaining the club in top-class condition. As an initiative, the club outlined the development of software that would calculate the membership fee.

As their software consultant, you help them by developing a C# application.

**Functionalities:**

In class **Club,**implement the below-given properties.

|  |  |
| --- | --- |
| **Data Type** | **Property Name** |
| string     string     string | MemberId       MemberName       MemberType |

In class **Service,**implement the below-given methods and also **Inherit** the class **Club**.

|  |  |
| --- | --- |
| **Method** | **Description** |
| public bool ValidateMemberId() | This method is used to validate the member id.  The member id should be the member type, followed by **4 digits**.  The member type is "**Gold**" or "**Premium**"  Example : **Premium3456**  If the above conditions are satisfied, then return **true**. Otherwise return **false.** |
| public double CalculateMembershipFees() | This method is used to calculate the membership fees and return the **fees**.  If the member type is "**Gold**" , then return the membership fees of **50000**.  If the member type is "**Premium**" , then return the membership fees of **75000**. |

In **Program** class - **Main** method,

**1.**Get the **MemberId**value from the **user**.

**2.**Call the **ValidateMemberId**method, If it returns true, then get the **MemberName**and **MemberType**values from the user and move on to step 3, If it returns false then display **Invalid member id.**

**3.**Use the values in method **CalculateMembershipFees**and display the result as per the sample output.

**Note:**

* Keep the properties, methods and classes as **public.**
* Please read the method rules **clearly**.
* Do not use **Environment.Exit()** to terminate the program.
* Do not change the given code template.

**Sample Input 1:**

Enter member id

**Gold1123**

Enter member name

**Nisha**

Enter member type

**Gold**

**Sample Output 1:**

Membership Fees is 50000

**Sample Input 2:**

Enter member id

**Silver1123**

**Scenario:**

Hamels is an experienced traveler and an avid mountaineer. He has decided that his next adventure will be climbing a mountain and he wants to choose one that is particularly tall. He has researched several mountains and narrowed down his options to a few that he is interested in climbing.

However, he wants to make sure that he chooses the tallest one out of the options he has. He wants software that provides information on the height of different mountains around the world.

You being his software consultant help him by developing a C# application.

**Functionalities:**

In class **Program,** implement the below-given method.

**public static Dictionary<string, int> MountainDetails**-In the code template, it is already provided.

implement the features listed below.

|  |  |
| --- | --- |
| **Method** | **Description** |
| public void AddMountainDetails (string[ ] mountain) | This method is used to add the mountain details into the **MountainDetails**dictionary.  This method should separate the values in each string by a **colon(:)** from the input array and store them in a **Dictionary**. |
| public int FindMountainHeight (String  mountainName) | This method is used to find the height of the mountain based on the **mountainName**passed as an argument.  If the mountain name is found in the Dictionary, then return that mountain **height**. else, return **-1** and print "**No  mountains  are available**"  in the Main method. |
| public List <String>  FindTheHighestMountains() | This method is used to find the highest height mountain from the  **MountainDetails**dictionary, then store the mountain name as a **list**and return it.  **Note:** If the highest height has more than one mountain, then consider that all are the highest height mountains, and they need to be added to the list. |

In **Program** class, **Main** method,

**1.**Get the values from the **user**.

**2.**Call the methods accordingly and display the result.

**3.** In the Sample Input / Output provided, the highlighted text in bold corresponds to the input given by the user and the remaining text represents the output.

**Note:**

* Keep the method and class as **public.**
* Please read the method rules **clearly**.
* Do not use **Environment.Exit()** to terminate the program.
* Do not change the given code template.

**Sample Input and Output :**

1. Add Mountain Details

2. View Mountain Height

3. View Mountains With Highest Height

4. Exit

Enter the choice

**1**

Enter the number of entries

**5**

**Mount Elbrus:7970**

**Aconcagua:7827**

**Manaslu:4510**

**Mount Vinson:7970**

**K2:6120**

1. Add Mountain Details

2. View Mountain Height

3. View Mountains With Highest Height

4. Exit

Enter the choice

**2**

Enter the mountain name needs to be searched

**K34**

No mountains are available

1. Add Mountain Details

2. View Mountain Height

3. View Mountains With Highest Height

4. Exit

Enter the choice

**2**

Enter the mountain name needs to be searched

**Aconcagua**

Height is : 7827

1. Add Mountain Details

2. View Mountain Height

3. View Mountains With Highest Height

4. Exit

Enter the choice

**3**

Mountain names with the highest height are:

Mount Elbrus

Mount Vinson

1. Add Mountain Details

2. View Mountain Height

3. View Mountains With Highest Height

4. Exit

Enter the choice

**4**

Thank you.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text.RegularExpressions;

public class Club

{

public string MemberId { get; set; }

public string MemberName { get; set; }

public string MemberType { get; set; }

}

public class Service : Club

{

public bool ValidateMemberId()

{

return Regex.IsMatch(MemberId, $"^{MemberType}\\d{{4}}$");

}

public double CalculateMembershipFees()

{

return MemberType == "Gold" ? 50000 : MemberType == "Premium" ? 75000 : 0;

}

}

public class Program

{

static void Main(string[] args)

{

Service service = new Service();

Console.WriteLine("Enter member id");

service.MemberId = Console.ReadLine();

if (!service.ValidateMemberId())

{

Console.WriteLine("Invalid member id.");

return;

}

Console.WriteLine("Enter member name");

service.MemberName = Console.ReadLine();

Console.WriteLine("Enter member type");

service.MemberType = Console.ReadLine();

double fees = service.CalculateMembershipFees();

Console.WriteLine($"Membership Fees is {fees}");

}

}

**Scenario:**

Treasure Mine, a jewellery store, is planning to develop a software system to keep track of their stock. The store specialises in unique and high-end jewellery pieces, and they want to ensure that they can easily keep track of their inventory levels and sales. The store wants to be able to quickly and easily see how many of each item they have in stock, so they can reorder items that are running low.

You being their software consultant help them with the process by developing a C# application.

**Functionalities:**

In class **Program,** implement the below-given method.

**public static Dictionary<String, int> ProductDetails**-In the code template, it is already provided.

implement the features listed below.

|  |  |
| --- | --- |
| **Method** | **Description** |
| public void AddProductDetails (string itemName, int quantity) | This method is used to add the product details such as **itemName**and **quantity**passed as an argument into the  **ProductDetails**dictionary. |
| public List<string> CheckReorderLevel(int reorderLevel) | This method is used to find the **items name**that have less stock than the **reorderLevel**passed as an argument.  If the stock is available less than the reorder level, then store the items name as **list**and return it. else, return an **empty list** and print "**No need for reorder**" in Main method. |

In **Program** class, **Main** method,

**1.**Get the values from the **user**.

**2.**Call the methods accordingly and display the result.

**3.**     In the Sample Input / Output provided, the highlighted text in bold corresponds to the input given by the user and the remaining text represents the output.

**Note:**

* Keep the method and class as **public.**
* Please read the method rules **clearly**.
* Do not use **Environment.Exit()** to terminate the program.
* Do not change the given code template.

**Sample Input and Output 1:**

Enter the number of products

**3**

Enter the item name

**Diamond set**

Enter the stock quantity

**100**

Enter the item name

**Gold chain**

Enter the stock quantity

**2000**

Enter the item name

**Platinum set**

Enter the stock quantity

**197**

Enter re-order level

**500**

List of Products - reorder level below 500

Diamond set

Platinum set

**Sample Input and Output 2:**

Enter the number of products

**2**

Enter the item name

**Diamond set**

Enter the stock quantity

**1500**

Enter the item name

**Gold chain**

Enter the stock quantity

**2000**

Enter re-order level

**500**

No need for reorder

using System;

using System.Collections.Generic;

public class Program

{

public static Dictionary<String, int> ProductDetails = new Dictionary<String, int>();

public static void AddProductDetails(string itemName, int quantity)

{

ProductDetails[itemName] = quantity;

}

public static List<string> CheckReorderLevel(int reorderLevel)

{

List<string> reorderItems = new List<string>();

foreach (var item in ProductDetails)

{

if (item.Value < reorderLevel)

{

reorderItems.Add(item.Key);

}

}

return reorderItems;

}

public static void Main(string[] args)

{

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

int numProducts = Convert.ToInt32(Console.ReadLine());

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

{

Console.WriteLine("Enter the item name");

string itemName = Console.ReadLine();

Console.WriteLine("Enter the stock quantity");

int quantity = Convert.ToInt32(Console.ReadLine());

AddProductDetails(itemName, quantity);

}

Console.WriteLine("Enter re-order level");

int reorderLevel = Convert.ToInt32(Console.ReadLine());

List<string> reorderItems = CheckReorderLevel(reorderLevel);

if (reorderItems.Count > 0)

{

Console.WriteLine("List of Products - reorder level below " + reorderLevel);

foreach (string item in reorderItems)

{

Console.WriteLine(item);

}

}

else

{

Console.WriteLine("No need for reorder");

}

}

}

**Scenario:**

George owns one of the leading detergent stores in the city with two outlets. He plans to automate the process in his store as the sales increase. He wants the system to store detergent details and help him generate bills.

Detergent name, packet capacity (in grams), and cost per pack are the expected details to be stored in the system. And whenever a customer buys a detergent packet, the bill needs to be calculated for the given quantity (in kg).

You being their software consultant help them with the process by developing a C# application.

**Functionalities:**

In class **Detergent,**implement the below-given properties.

|  |  |
| --- | --- |
| **Data Type** | **Property Name** |
| string      string       int      double | BillId       Name       GramsInPack       CostPerPack |

In class **Service,**implement the below-given methods and also **Inherit** the class **Detergent**.

|  |  |
| --- | --- |
| **Method** | **Description** |
| public bool ValidateBillId( ) | This method is used to validate the **bill id**.  The bill id should consists of **5** characters, the first **two**characters should be **digits**, followed by **colon (:)** and the last **two**characters should be **alphabets**in upper case.  If the above conditions are satisfied, then return  **true**. Otherwise, return **false.** |
| public double CalculateTotalCost(float quantity) | This method is used to calculate the total cost and return the **cost**.  Refer to the below formula to calculate the total cost. |

**Formula :**

Total Cost = (CostPerPack \* (quantity \* 1000) / GramsInPack)

**1.**Get the **BillId** value from the **user**.

**2.**Call the **ValidateBillId**method, If it returns true, then get the **Name**, **GramsInPack**, **CostPerPack**and **quantity**values from the user and move on to step 3, If it returns false then display **Invalid bill id.**

**3.**Use the values in method**CalculateTotalCost**and display the result as per the sample output.

**Note:**

* Keep the properties, methods and classes as **public.**
* Please read the method rules **clearly**.
* Do not use **Environment.Exit()** to terminate the program.
* Do not change the given code template.

**Sample Input 1:**

Enter bill id

**12:AB**

Enter detergent name

**Arial**

Enter pack capacity in grams

**250**

Enter cost per pack

**80**

Enter Quantity to purchase in kgs

**1**

**Sample Output 1:**

Detergent cost is 320

**Sample Input 2:**

Enter bill id

**123-ABC**

**Sample Output 2:**

Invalid bill id

using System;

using System.Text.RegularExpressions;

public class Detergent

{

public string BillId { get; set; }

public string Name { get; set; }

public int GramsInPack { get; set; }

public double CostPerPack { get; set; }

}

public class Service : Detergent

{

public bool ValidateBillId()

{

Regex regex = new Regex(@"^\d{2}:[A-Z]{2}$");

return regex.IsMatch(BillId);

}

public double CalculateTotalCost(float quantity)

{

return (CostPerPack \* (quantity \* 1000) / GramsInPack);

}

}

class Program

{

static void Main(string[] args)

{

Service service = new Service();

Console.WriteLine("Enter bill id");

service.BillId = Console.ReadLine();

if (!service.ValidateBillId())

{

Console.WriteLine("Invalid bill id");

return;

}

Console.WriteLine("Enter detergent name");

service.Name = Console.ReadLine();

Console.WriteLine("Enter pack capacity in grams");

service.GramsInPack = int.Parse(Console.ReadLine());

Console.WriteLine("Enter cost per pack");

service.CostPerPack = double.Parse(Console.ReadLine());

Console.WriteLine("Enter Quantity to purchase in kgs");

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

double totalCost = service.CalculateTotalCost(quantity);

Console.WriteLine($"Detergent cost is {totalCost}");

}

}

**Scenario:**

Alia Cabs has emerged as a rapidly expanding startup, riding the wave of technological advancements in the Internet and consumer space. This has presented them with an exciting market opportunity, enabling them to incorporate cutting-edge technology into their operations. With a diverse fleet comprising hatchbacks, sedans, and SUVs, Alia Cabs aims to cater to the varying needs of their customers. To streamline their services, they are seeking innovative software that can accurately calculate fares based on the specific vehicle type selected by passengers.

As their software consultant, you can help them by developing a C# application

**Functionalities:**

In class **Cab,**implement the below-given properties.

|  |  |
| --- | --- |
| **Class** | **Properties** |
| Cab | string BookingID       string CabType       double Distance       double Fare |

In class **CabDetails,**implement the below-given methods and also **Inherit** class **Cab**.

|  |  |  |
| --- | --- | --- |
| **Class** | **Method** | **Description** |
| CabDetails | public bool ValidateBookingID() | This method is used to validate the booking id.  In this method extract the booking id by character **(@).**The id should have **AC** before the character **@**and three numbers after the character **@.**  **For Example :  AC@123**  The above conditions are passed then return **true**, Otherwise return **false** |
| CabDetails | public Cab CalculateFareAmount() | This method is used to calculate the fare of the cab bookings based on the below formula.  Store the result in the property **Fare**  Then store the details such as **BookingID**, **CabType**, **Distance**and **Fare**in **Cab**object and return it.  Refer to the table below to calculate fare. |

**Formula :**

Fare = Distance \* Price per km

|  |  |
| --- | --- |
| **Cab Type** | **Price per km** |
| Hatchback | 10 |
| Sedan | 20 |
| SUV | 30 |

**Note :** Cab type is **case sensitive.**

In **Program** class - **Main** method,

**1.**Get the values from the **user**as per the Sample Input

**2.**Call the method **ValidateBookingID**, if it returns **true**then move on to next step, else display **"Invalid booking id"**in Main method

**3.**Call the method **CalculateFareAmount**and display the result as per the Sample Output.

**Note:**

* Keep the properties, methods and classes as **public.**
* Please read the method rules **clearly**.
* Do not use **Environment.Exit()** to terminate the program.
* Do not change the given code template.

**Sample Input 1:**

 Enter the booking id

**AC@234**

Enter the cab type

**SUV**

Enter the distance in km

**10**

**Sample Output 1:**

Booking Id : AC@234

Cab Type : SUV

Distance : 10

Fare : 300

**Sample Input 2:**

Enter the booking id

**DC@123**

**Sample Output 2:**

Invalid booking id

**Sample Input 3:**

Enter the booking id

**AC@123456**

**Sample Output 3:**

Invalid booking id

using System;

using System.Text.RegularExpressions;

public class Cab

{

public string BookingID { get; set; }

public string CabType { get; set; }

public double Distance { get; set; }

public double Fare { get; set; }

}

public class CabDetails : Cab

{

public bool ValidateBookingID()

{

Regex regex = new Regex(@"^AC@\d{3}$");

return regex.IsMatch(BookingID);

}

public Cab CalculateFareAmount()

{

double pricePerKm = 0;

switch (CabType)

{

case "Hatchback":

pricePerKm = 10;

break;

case "Sedan":

pricePerKm = 20;

break;

case "SUV":

pricePerKm = 30;

break;

}

Fare = Distance \* pricePerKm;

return this;

}

}

class Program

{

static void Main(string[] args)

{

CabDetails cabDetails = new CabDetails();

Console.WriteLine("Enter the booking id");

cabDetails.BookingID = Console.ReadLine();

if (!cabDetails.ValidateBookingID())

{

Console.WriteLine("Invalid booking id");

return;

}

Console.WriteLine("Enter the cab type");

cabDetails.CabType = Console.ReadLine();

Console.WriteLine("Enter the distance in km");

cabDetails.Distance = double.Parse(Console.ReadLine());

Cab cab = cabDetails.CalculateFareAmount();

Console.WriteLine($"Booking Id : {cab.BookingID}");

Console.WriteLine($"Cab Type : {cab.CabType}");

Console.WriteLine($"Distance : {cab.Distance}");

Console.WriteLine($"Fare : {cab.Fare}");

}

}

**Scenario:**

As the sun shines brightly on this picturesque spring day, Maria, the proprietor of The Plant Haven, a thriving plant store in the heart of the town, busies herself with curating a diverse array of botanical offerings for her valued customers. From colorful annuals and hardy perennials to fragrant herbs and nutritious vegetables, she has something for everyone. To sweeten the deal, she even offers generous discounts to her patrons. As the day progresses, Maria is delighted by the steady stream of customers who come through her doors. Recognizing the need to streamline her operations, she decides to invest in software that can accurately calculate the cost of her plants after discounts have been applied.

You being her software consultant, help her by developing a C# application

**Functionalities:**

In class **Plant,**implement the below-given properties.

|  |  |
| --- | --- |
| **Datatype** | **Property Name** |
| string     int     string     int | PlantName     NoOfSapling     Category     PricePerSapling |

In class **PlantUtility,**implement the below-given methods and also **Inherit** class **Plant**.

|  |  |
| --- | --- |
| **Method** | **Description** |
| public Plant ExtractDetails(string plantDetails) | This method is used to extract plant details from a **colon(:) - separated**input **string**and assign the values to **Plant**class **object**and return that**.**  In this method, pass the plant details as an argument based on the below-given string format.  **<PlantName:NoOfSapling:Category:PricePerSapling>** |
| public double CalculateCost() | This method is used to calculate the cost of the plant after the discount and return it as a **double** datatype.  **Total Amount** = **NoOfSapling \* PricePerSapling**  Refer to the below procedure to calculate cost after the discount. |

**Formula :**

Cost after the discount = Total Amount - Discount

|  |  |
| --- | --- |
| **Total Amount** | **Discount** |
| > 500 to <=1000 | 10 % of Total Amount |
| Above 1000 | 20 % of Total Amount |
| <=500 | No discount |

In **Program** class - **Main** method,

**1.**Get the input details from the **user**.

**2.**Call the **ExtractDetails** method by using the input values.

**3.**Call the **CalculateCost** method and finally display all the details as per the output given.

**Note:**

* Keep the properties, methods and classes as **public.**
* Please read the method rules **clearly**.
* Do not use **Environment.Exit()** to terminate the program.
* Do not change the given code template.

**Sample Input 1:**

Enter the plant details

**Sunflower:10:flowering:70**

**Sample Output 1:**

Plant name is : Sunflower

No of sapling is : 10

Category : flowering

Price per sapling : 70

Total cost is : 630

**Sample Input 2:**

Enter the plant details

**Lavender:5:herb:30**

**Sample Output 2:**

Plant name is : Lavender

No of sapling is : 5

Category : herb

Price per sapling : 30

Total cost is : 150

using System;

public class Plant

{

public string PlantName { get; set; }

public int NoOfSapling { get; set; }

public string Category { get; set; }

public int PricePerSapling { get; set; }

}

public class PlantUtility : Plant

{

public Plant ExtractDetails(string plantDetails)

{

string[] details = plantDetails.Split(':');

PlantName = details[0];

NoOfSapling = int.Parse(details[1]);

Category = details[2];

PricePerSapling = int.Parse(details[3]);

return this;

}

public double CalculateCost()

{

double totalAmount = NoOfSapling \* PricePerSapling;

double discount = 0;

if (totalAmount > 500 && totalAmount <= 1000)

{

discount = 0.1 \* totalAmount;

}

else if (totalAmount > 1000)

{

discount = 0.2 \* totalAmount;

}

return totalAmount - discount;

}

}

class Program

{

static void Main(string[] args)

{

PlantUtility plantUtility = new PlantUtility();

Console.WriteLine("Enter the plant details");

string plantDetails = Console.ReadLine();

Plant plant = plantUtility.ExtractDetails(plantDetails);

double totalCost = plantUtility.CalculateCost();

Console.WriteLine($"Plant name is : {plant.PlantName}");

Console.WriteLine($"No of sapling is : {plant.NoOfSapling}");

Console.WriteLine($"Category : {plant.Category}");

Console.WriteLine($"Price per sapling : {plant.PricePerSapling}");

Console.WriteLine($"Total cost is : {totalCost}");

}

}

**Scenario:**

Win Cinemas is an inclusive theater that caters to everyone in our community. They take pride in enriching all their community through exceptional and daring theatrical performances, as well as engaging educational experiences. With their commitment to excellence, they guarantee a world-class theatrical journey enhanced by the mesmerizing 4K Dolby Atmos technology.  They need a software to maintain movie screening details.  
  
As their software consultant, you can help them by developing a C# application.

**Functionalities:**

In class **Movie**, implement the below-given properties.

|  |  |
| --- | --- |
| **Data Type** | **Property Name** |
| string | MovieName |
| string | ScreenedDate |
| string | RemovedDate |
| double | Price |

In class **Program,**

**public static Dictionary<int,Movie> screeningDetails**-In the code template, it is already provided. 

Implement the below-given method.

|  |  |  |
| --- | --- | --- |
| **Class** | **Method** | **Description** |
| Program | public Dictionary<string, double> MovieScreenedMoreNumberOfDays() | This method is used to find the movie which is screened more number of days using the screened and removed date from **screeningDetails**dictionary.  Then store the **movie name**and **price**in **Dictionary**and return it. |
| Program | public Dictionary<string, double> MovieWithScreenedDays() | This method is used to find all the movie as well as total number of screened days.  Then store the **movie name** and that movie's **total number of screened days**  as  **Dictionary**and return it. |

**Note**: The date format is **(MM/dd/yyyy).**

In **Program** class, **Main** method,

**1.**Get the values from the **user**.

**2.**Call the methods accordingly and display the result.

**3.**     In the Sample Input / Output provided, the highlighted text in bold corresponds to the input given by the user and the remaining text represents the output.

**Note:**

* Keep the properties, methods and classes as **public.**
* Please read the method rules **clearly**.
* Do not use **Environment.Exit()** to terminate the program.
* Do not change the given code template.

Assume these below-given data as example,

If you need you can try this in your code also.

**Sample Input / Output:**

1. Movie screening more number of days

2. Movie with their screening days

3. Exit

Enter your choice

**1**

Avatar 150

1. Movie screening more number of days

2. Movie with their screening days

3. Exit

Enter your choice

**2**

Eternals  35

Iron Man  31

Avatar     103

LightYear  20

Black Panther  24

1. Movie screening more number of days

2. Movie with their screening days

3. Exit

Enter your choice

**3**

Thank You

using System;

using System.Collections.Generic;

using System.Globalization;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Dict{

public class Movie{

public string MovieName{get; set;}

public string ScreenedDate{get; set;}

public string RemovedDate{get; set;}

public double Price{ get; set;}

public Movie(string m,string s, string r, double p){

MovieName = m;

ScreenedDate = s;

RemovedDate = r;

Price = p;

}

}

public class Program{

public static Dictionary<int, Movie> screeningDetails = new Dictionary<int, Movie>();

public static Dictionary<string, double> MovieScreenedMoreNumberOfDays(){

Dictionary<string, double> MSmNoD = new Dictionary<string, double>();

string maxScreenedMovie = null;

double Maxdays= 0;

foreach(KeyValuePair<int,Movie> KVP in screeningDetails){

Movie movie = KVP.Value;

TimeSpan NoOfDdays = (DateTime.Parse(movie.ScreenedDate)-DateTime.Parse(movie.RemovedDate));

if(maxScreenedMovie == null || Maxdays<Math.Abs(NoOfDdays.Days)){

maxScreenedMovie = movie.MovieName;

Maxdays =Math.Abs(NoOfDdays.Days);

}

}

MSmNoD.Add(maxScreenedMovie,Maxdays);

return MSmNoD;

}

public static Dictionary<string, double> MovieWithScreenedDays(){

Dictionary<string, double> mwsd = new Dictionary<string, double>();

foreach(KeyValuePair<int,Movie> KVP in screeningDetails){

Movie movie = KVP.Value;

TimeSpan NoOfDdays = (DateTime.Parse(movie.ScreenedDate)-DateTime.Parse(movie.RemovedDate));

mwsd.Add(movie.MovieName,Math.Abs(NoOfDdays.Days));

}

return mwsd;

}

static void Main(string[] args){

screeningDetails.Add(1,new Movie("Eternals","04/25/2020","05/30/2020",350));

screeningDetails.Add(2,new Movie("Iron Man","07/15/2008","08/15/2008",100));

screeningDetails.Add(3,new Movie("Avatar","10/25/2003","02/05/2004",150));

screeningDetails.Add(4,new Movie("LightYear","07/10/2020","07/30/2020",1750));

screeningDetails.Add(5,new Movie("Black Panther","05/17/2020","06/10/2020",200));

Console.WriteLine("1. Movie screening more number of days\n2. Movie with their screening days\n3. Exit\nEnter your choice");

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

while(true){

switch(choice){

case 1: Dictionary<string, double> output = MovieScreenedMoreNumberOfDays();

foreach(KeyValuePair<string, double> KVP in output){

Console.WriteLine($"{KVP.Key} {KVP.Value}");

return;

}

break;

case 2: Dictionary<string, double> output2 = MovieWithScreenedDays();

foreach(KeyValuePair<string, double> KVP2 in output2){

Console.WriteLine($"{KVP2.Key} {KVP2.Value}");

}

break;

case 3: Console.WriteLine("Thank you");

return;

}

}

}

}

}

**Scenario:**

There are various types of mobile apps and web apps that greatly assist customers of a well-known software company in the United States, which provides products to its customers. Recently, the company received an order from a client for a tax amount calculator. The employee provides their ID and salary, and the company wishes to display their tax amount based on the salary.

 As their software consultant, you need to help them by developing a C# application.

**Functionalities:**

In class **Employee,**implement the below-given properties.

|  |  |
| --- | --- |
| **Datatype** | **Property Name** |
| string | EmployeeId |
| double | Salary |

In class **EmployeeUtility,**implement the below-given methods and also **Inherit** the class **Employee**.

|  |  |
| --- | --- |
| **Method** | **Description** |
| public bool ValidateEmployeeId() | This method is used to validate the employee id.  Condition :  1. The employee id length should be 4.  2. It should be in the format of a capital letter followed by 3 digits  **For Example :**E123  If the above-given conditions are passed, then return **true**. Otherwise return **false.** |
| public double CalculateTaxAmount() | This method is used to calculate the tax amount of the employee based on their salary and return it.  If the employee's salary is up to 20,000, then the tax amount is **0**.  If the employee's salary is between 20,001 and 50,000, the first 20,000 is tax-free, after that, the tax rate is **10%**.  If an employee's salary is between 50,001 and 100000, the first 20,000 is tax-free, after that, the tax rate is **10%**, and after 50,000, the tax rate is **20%**.  If an employee's salary is above 100000, the first 20,000 is tax-free, after that, the tax rate is **10%,**and after 50,000, the tax rate is **20%,** and after 100000, the tax rate is **30%**. |

**For Example :**

1. **Employee Salary =  45000.**

     The first 20000 is tax free, and the remaining amount of 25000 is taxed at 10%.

     So, **Tax Amount**= (Salary - 20000) \* 0.10

2. **Employee Salary =  65000.**

     The first 20,000 is tax-free, the next 30,000 is taxed at 10%, and the remaining 15,000 is taxed at 20%.

     So, **Tax Amount** = (20000 \* 0) + (30000 \* 0.10) + ((Salary - 50000) \* 0.20)

3. **Employee Salary =  155000.**

     The first 20,000 is tax-free, the next 30,000 is taxed at 10%, the next 50,000 is taxed at 20% and the remaining 55,000 is taxed at 30%.

     So, **Tax Amount** = (20000 \* 0) + (30000 \* 0.10) + (50000 \* 0.20) + ((Salary - 100000) \* 0.30)

In **Program** class - **Main** method,

**1.**Get the values from the **user**as per the Sample Input.

**2.**Call the **ValidateEmployeeId**method, If it returns true then get the **Salary**value from the **user**and move on to step 3, If it returns false then display **Invalid employee id.**

**3.**Use the values in method**CalculateTaxAmount**and display the result as per the Sample Output.

**Note:**

* Keep the properties, methods and classes as **public.**
* Please read the method rules **clearly**.
* Do not use **Environment.Exit()** to terminate the program.
* Do not change the given code template.

**Sample Input 1:**

Enter the employee id

**H456**

Enter the salary

**15000**

**Sample Output 1:**

Tax amount to pay is 0

**Sample Input 2:**

Enter the employee id

**A234**

Enter the salary

**200000**

**Sample Output 2:**

Tax amount to pay is 43000

**Sample Input 3:**

Enter the employee id

**A123456**

**Sample Output 3:**

Invalid employee id

**Sample Input 4:**

Enter the employee id

**e123**

**Sample Output 4:**

Invalid employee id

using System;

using System.Collections.Generic;

using System.Globalization;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Employee{

public class Employee{

public string EmployeeId{get; set;}

public double Salary{get; set;}

public Employee(){}

public Employee(string id,double salary){

EmployeeId = id;

Salary = salary;

}

}

public class EmployeeUtility:Employee{

public EmployeeUtility():base(){}

public EmployeeUtility(string id,double salary):base(id,salary){

}

public bool ValidateEmployeeId(){

Regex reg = new Regex("^[A-Z]{1}[0-9]{3}$");

if(EmployeeId.Length==4 && reg.IsMatch(EmployeeId))

{ return true;

}

else{

return false;

}

}

public double CalculateTaxAmount(){

double tax=0;

if(Salary<=20000)

tax=0;

else if(Salary>20000 && Salary<=50000)

tax=(Salary-20000)\*0.1;

else if(Salary>50000 && Salary<=100000)

tax=3000+(Salary-50000)\*0.2;

else if(Salary>100000)

tax=3000+10000+(Salary-100000)\*0.3;

return tax;

}

}

public class Program(){

public static void Main(string[] args){

Console.WriteLine("Enter the Employee id :");

string employeeId =Console.ReadLine();

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

EmployeeUtility e = new EmployeeUtility(employeeId,salary);

if(e.ValidateEmployeeId()){

double tax = e.CalculateTaxAmount();

Console.WriteLine("Tax amount to pay is "+tax);

}

else{

Console.WriteLine("Invalid employee id");

}

}

}

}

**Scenario:**

Get Holiday is a well-known tourist agency and they usually book hotel rooms for their customers based on the hotel rating. Now, manually locating the best hotel is extremely difficult. So, they need an application that can use web scraping techniques to gather information on various hotels and their ratings, and then allow the user to filter and sort through the results to find the best options for their customers.

As their software consultant, you can help them by developing a C# application.

**Functionalities:**

In class **Program,** implement the below-given method.

**public static Dictionary<String, float> hotelDetails**-In the code template, it is already provided.

implement the features listed below.

|  |  |
| --- | --- |
| **Method** | **Description** |
| public Dictionary<String, float> SearchHotel(String hotelName) | This method is used to find the hotel details by hotel name.  If the hotel name is available in the **hotelDetails,**this method should return the hotel name and rating as a **Dictionary**.  If the hotel name is not available in the **hotelDetails ,** then it should return an empty **Dictionary**.  If this method returns an empty **Dictionary,**then print **"Hotel Not Found"** in Main method. |
| public Dictionary<String, float> UpdateHotelRating(string hotelName, float rating) | This method is used to update the rating of the hotel by hotel name.  If the hotel name is available in the **hotelDetails ,**it should  return the hotel name and updated rating as a  **Dictionary**.  If the hotel name is not available in the**hotelDetails ,** then it should return an empty **Dictionary**.  If this method returns an empty **Dictionary,**then print **"Hotel Not Found"** in Main method. |
| public Dictionary<String, float> SortByHotelName() | This method is used to display all the hotels available in the **hotelDetails**in ascending order by hotel name.  The result should be **Dictionary** |

In **Program** class, **Main** method,

**1.**Get the values from the **user**.

**2.**Call the methods accordingly and display the result.

**3.**     In the Sample Input / Output provided, the highlighted text in bold corresponds to the input given by the user and the remaining text represents the output.

**Note:**

* Keep the method and class as **public.**
* Please read the method rules **clearly**.
* Do not use **Environment.Exit()** to terminate the program.
* Do not change the given code template.

**Sample Input / Output:**

1. Search by hotel name

2. Update hotel rating

3. Sort hotels by name

4. Exit

Enter your choice

**1**

Enter the hotel name

**The Hay Adams**

The Hay Adams 3

1. Search by hotel name

2. Update hotel rating

3. Sort hotels by name

4. Exit

Enter your choice

**2**

Enter the hotel name

**Montage Kapalua Bay**

Enter the rating

**4**

Montage Kapalua Bay  4

1. Search by hotel name

2. Update hotel rating

3. Sort hotels by name

4. Exit

Enter your choice

**2**

Enter the hotel name

**Line Bay**

Enter the rating

**5**

Hotel Not Found

1. Search by hotel name

2. Update hotel rating

3. Sort hotels by name

4. Exit

Enter your choice

**3**

Jungle Resort   4.5

Mandarin Oriental   5

Montage Kapalua Bay   4

The Greenwich Hotel    5

The Hay Adams    3

1. Search by hotel name

2. Update hotel rating

3. Sort hotels by name

4. Exit

Enter your choice

**4**Thank You

using System;

using System.Collections.Generic;

using System.Linq;

public class Program

{

public static Dictionary<String, float> hotelDetails = new Dictionary<String, float>();

public static Dictionary<String, float> SearchHotel(String hotelName)

{

if (hotelDetails.ContainsKey(hotelName))

{

return new Dictionary<String, float> { { hotelName, hotelDetails[hotelName] } };

}

else

{

return new Dictionary<String, float>();

}

}

public static Dictionary<String, float> UpdateHotelRating(string hotelName, float rating)

{

if (hotelDetails.ContainsKey(hotelName))

{

hotelDetails[hotelName] = rating;

return new Dictionary<String, float> { { hotelName, rating } };

}

else

{

return new Dictionary<String, float>();

}

}

public static Dictionary<String, float> SortByHotelName()

{

return hotelDetails.OrderBy(hotel => hotel.Key).ToDictionary(hotel => hotel.Key, hotel => hotel.Value);

}

public static void Main(string[] args)

{

while (true)

{

Console.WriteLine("1. Search by hotel name\n2. Update hotel rating\n3. Sort hotels by name\n4. Exit");

Console.WriteLine("Enter your choice");

int choice = Convert.ToInt32(Console.ReadLine());

if (choice == 4)

{

break;

}

switch (choice)

{

case 1:

Console.WriteLine("Enter the hotel name");

string hotelName = Console.ReadLine();

var result = SearchHotel(hotelName);

if (result.Count == 0)

{

Console.WriteLine("Hotel Not Found");

}

else

{

foreach (var item in result)

{

Console.WriteLine($"{item.Key} {item.Value}");

}

}

break;

case 2:

Console.WriteLine("Enter the hotel name");

hotelName = Console.ReadLine();

Console.WriteLine("Enter the rating");

float rating = float.Parse(Console.ReadLine());

result = UpdateHotelRating(hotelName, rating);

if (result.Count == 0)

{

Console.WriteLine("Hotel Not Found");

}

else

{

foreach (var item in result)

{

Console.WriteLine($"{item.Key} {item.Value}");

}

}

break;

case 3:

result = SortByHotelName();

foreach (var item in result)

{

Console.WriteLine($"{item.Key} {item.Value}");

}

break;

}

}

}

}