Assignment #2

Create C# Program for Completing Different Tasks

Yash Ketanbhai Shah

8990493

High-Quality Software Programming

PROG8051 - Winter 2025 - Section 1

Shankar Iyer

15th February, 2025

Task 1

Project: Customer Wiring Management System

Description: The **Customer Wiring Management System** is a C# code that will collect the customers details, classification of their building and determining the appropriate wiring tasks. This code is created by using OOP concepts which includes encapsulation, constructors, methods and enumerations.

Features and Functionalities

1. **Building Classification**: Uses an enumeration (enum) to define different building types.
2. **Customer Information Handling**: Stores and processes customer details which includes the credit card masking for security.
3. **Wiring Task Automation**: It will determine and execute wiring tasks based on the building type.
4. **User Input Validation**: It will ensure the valid data entry for numerical and textual inputs.

```
/*
 * Author: Shah Yash Ketanbhai
 * Date: 15th February, 2025
 * Project: Customer Wiring Management System
 * Description: The Customer Wiring Management System is a C# code that
will collect the customers details,
 * classification of their building  and determining the appropriate
wiring tasks. This code is created by using
 * OOP concepts which include encapsulation, constructors, methods and
enumerations.
 * <Fill>
*/

using System;
using System.Collections.Generic;

// We have used Enum to define the building type
enum BuildingType
{
    House,
    Barn,
    Garage
}
```

```
class Customer
{
    public string Name { get; set; }
    public BuildingType StructureType { get; set; }
    public double Size { get; set; }
    public double LightBulbs { get; set; }
    public double Outlets { get; set; }
    public string CreditCard { get; set; }

    // Here we have use the Constructor
    public Customer(String name, BuildingType type, double size,  double
lightBulbs, double outlets, string creditCard)
    {  Name = name; StructureType = type; Size = size; LightBulbs =
lightBulbs; Outlets = outlets; CreditCard = MaskCreditCard(creditCard);
}// Mask card before storing

    // Here we will be masking the credit card detials from the customer
    private string MaskCreditCard(string cardNumber)
    {
        return cardNumber.Substring(0,4) + " XXXX XXXX " +
cardNumber.Substring(12, 4);
    }

    // Here we have created a method to display the customer details
    public void DisplayCustomerInfo()
    {
        Console.WriteLine($"{Name} | {StructureType} | {Size} sq.ft |
{LightBulbs} bulbs | {Outlets} outlets | Card: {CreditCard}");
    }

    // Here we have created a method to perform specific wiring tasks
based on structure type
    public void PerformWiringTasks()
    {
        Console.WriteLine($"Creating wiring schema for
{StructureType}...");
        Console.WriteLine("Purchasing necessary parts...");

        if (StructureType == BuildingType.House)
        {
            Console.WriteLine("Installing fire alarms...");
        }
        else if (StructureType == BuildingType.Barn)
        {
            Console.WriteLine("Wiring milking equipment...");
        }
        else if (StructureType == BuildingType.Garage)
        {
            Console.WriteLine("Installing automatic doors...");
        }
    }
    class Program
    {
        static void Main()
        {
            List<Customer> customers = new List<Customer>();
            string continueInput;

            do
            {
                // Collect customer details
                Console.Write("Enter customer name: ");
                string name = Console.ReadLine();

                Console.Write("Enter building type (House, Barn, Garage):
");
                BuildingType type;
                while (!Enum.TryParse(Console.ReadLine(), true, out type))
                {
                    Console.Write("Invalid input. Enter building type
(House, Barn, Garage): ");
                }

                double size;
                do
                {
                    Console.Write("Enter building size (1000 - 50000
sq.ft): ");
```

```
                } while (!double.TryParse(Console.ReadLine(), out size) ||
size < 1000 || size > 50000);

                double bulbs;
                do
                {
                    Console.Write("Enter number of light bulbs (max 20):
");
                } while (!double.TryParse(Console.ReadLine(), out bulbs)
|| bulbs < 0 || bulbs > 20);

                double outlets;
                do
                {
                    Console.Write("Enter number of outlets (max 50): ");
                } while (!double.TryParse(Console.ReadLine(), out outlets)
|| outlets < 0 || outlets > 50);

                string creditCard;
                do
                {
                    Console.Write("Enter 16-digit credit card number: ");
                    creditCard = Console.ReadLine();
                } while (creditCard.Length != 16 ||
!long.TryParse(creditCard, out _));

                // Create a new customer and add to list
                Customer newCustomer = new Customer(name, type, size,
bulbs, outlets, creditCard);
                customers.Add(newCustomer);

                // Perform wiring tasks
                newCustomer.PerformWiringTasks();

                // Ask if another customer should be added
                Console.Write("Do you want to enter another customer?
(yes/no): ");
                continueInput = Console.ReadLine().ToLower();

            } while (continueInput == "yes");
            // Display all customers at the end
            Console.WriteLine("\nCustomer Summary:");

Console.WriteLine("-------------------------------------------------------
-----");
            foreach (var customer in customers)
            {
                customer.DisplayCustomerInfo();
            }
        }
    }
}
```

Code Breakdown

1. **Enumerations: Defining Building Types**

```
// We have used Enum to define the building type
enum BuildingType
{
    House,
    Barn,
    Garage
}
```

The `BuildingType` enumeration defines three possible building structures: House, Barn and Garage allowing easy classification and selection.

Reference:

- Enums in C#: Microsoft Docs. Retrieved from: https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/enum

2. **Class Definition: Customer**

```
class Customer
{
    public string Name { get; set; }
    public BuildingType StructureType { get; set; }
    public double Size { get; set; }
    public double LightBulbs { get; set; }
    public double Outlets { get; set; }
    public string CreditCard { get; set; }
```

The `Customer` class encapsulates customer details using auto-implemented properties.

3. **Constructor: Initializing Customer Data**

```
    // Here we have use the Constructor
    public Customer(String name, BuildingType type, double size,  double
lightBulbs, double outlets, string creditCard)
```

```
    {  Name = name; StructureType = type; Size = size; LightBulbs =
lightBulbs; Outlets = outlets; CreditCard = MaskCreditCard(creditCard);
}// Mask card before storing
```

The constructor initializes the customer attributes and applies credit card masking before storage.

4. **Credit Card Masking Function**

```
    // Here we will be masking the credit card detials from the customer
    private string MaskCreditCard(string cardNumber)
    {
        return cardNumber.Substring(0,4) + " XXXX XXXX " +
cardNumber.Substring(12, 4);
    }
```

Security Considerations:

- Credit card details are partially masked to ensure privacy and prevent unauthorized access.
- Uses Substring() to retain only the first and last four digits.

Reference:

- Best Practices for Credit Card Masking: PCI DSS Compliance Guide. Retrieved from: https://www.pcisecuritystandards.org/

5. **Displaying Customer Details**

```
    // Here we have created a method to display the customer details
    public void DisplayCustomerInfo()
    {
        Console.WriteLine($"{Name} | {StructureType} | {Size} sq.ft |
{LightBulbs} bulbs | {Outlets} outlets | Card: {CreditCard}");
    }
```

This method outputs the customer's details, ensuring privacy compliance for sensitive data.

6. **Wiring Task Execution**

```
    // Here we have created a method to perform specific wiring tasks
based on structure type
```

```
    public void PerformWiringTasks()
    {
        Console.WriteLine($"Creating wiring schema for 
{StructureType}...");
        Console.WriteLine("Purchasing necessary parts...");

        if (StructureType == BuildingType.House)
        {
            Console.WriteLine("Installing fire alarms...");
        }
        else if (StructureType == BuildingType.Barn)
        {
            Console.WriteLine("Wiring milking equipment...");
        }
        else if (StructureType == BuildingType.Garage)
        {
            Console.WriteLine("Installing automatic doors...");
        }
    }
```

This method executes specific tasks based on building type, ensuring customization per customer requirements.

7. Main Program Execution

```
        static void Main()
        {
            List<Customer> customers = new List<Customer>();
            string continueInput;
```

The Main() method manages customer input collection, validation, and processing.

8. **Input Validation Mechanisms**

```
                double size;
                do
                {
                    Console.Write("Enter building size (1000 - 50000 
sq.ft): ");
```

```
                } while (!double.TryParse(Console.ReadLine(), out size) ||
size < 1000 || size > 50000);
```

This ensures user input remains within a valid range.

Reference:

- Handling User Input in C#: Microsoft Docs. Retrieved from: https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/types

9. Customer Storage & Processing

```
                // Create a new customer and add to list
                Customer newCustomer = new Customer(name, type, size,
bulbs, outlets, creditCard);
                customers.Add(newCustomer);
```

Each new customer instance is added to a list for later retrieval and display.

10. Displaying Summary

```
// Display all customers at the end
            Console.WriteLine("\nCustomer Summary:");

Console.WriteLine("-------------------------------------------------------
-----");
            foreach (var customer in customers)
            {
                customer.DisplayCustomerInfo();
            }
```

At the end of the program, a summary of all customers is displayed.

References:

1. Microsoft Docs: Enums in C#.

   https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/enum

2. PCI Security Standards Council: Best Practices for Credit Card Masking.

   https://www.pcisecuritystandards.org/

3. Microsoft Docs: Handling User Input in C#.

   https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/types

Task 2

Project: Customer Wiring Management System Updated

Description: This code provides an overview of the updated Customer Wiring Management System, which will help to handle wiring requirements for different types of buildings. The code takes customer details, processes wiring tasks based on structure types, and ensures secure handling of sensitive information such as credit card numbers using interface, Enumeration, Abstract Base Class, Concrete Class.

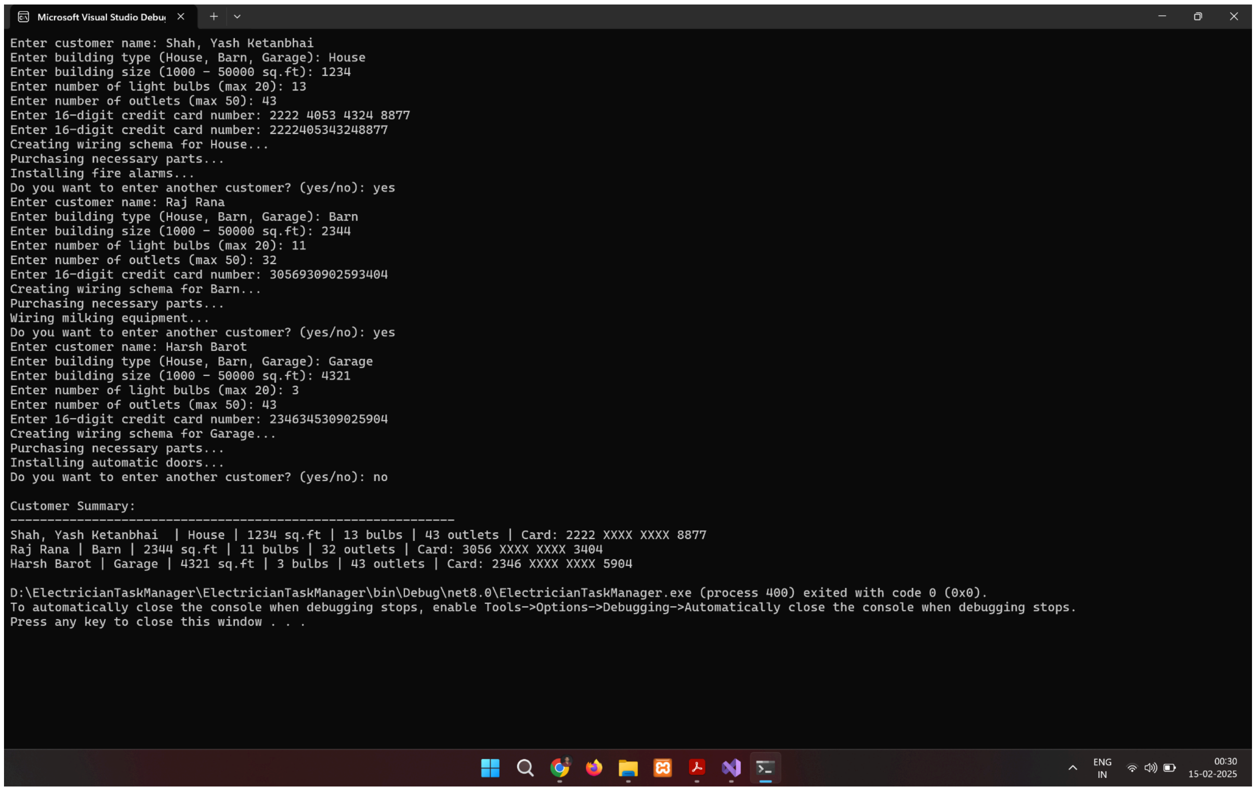
```
/*
 * Author: Shah Yash Ketanbhai
 * Date: 15th February, 2025
 * Project: Customer Wiring Management System Updated
 * Description: This code provides an overview of the updated Customer
Wiring Management System,
 * which will help to handle wiring requirements for different types of
buildings. The code takes customer details,
 * processes wiring tasks based on structure types, and ensures secure
handling of sensitive information such
 * as credit card numbers using interface, Enumeration, Abstract Base
Class, Concrete Class.
 * <Fill>
*/

using System;
using System.Collections.Generic;

// Here we have defined an interface for customer functionality
interface ICustomer
{
    string Name { get; set; }
    BuildingType StructureType { get; set; }
    double Size { get; set; }
    double LightBulbs { get; set; }
    double Outlets { get; set; }
    string CreditCard { get; set; }
    void DisplayCustomerInfo();
    void PerformWiringTasks();
}
```

```
// Enum for building type
enum BuildingType
{
    House,
    Barn,
    Garage
}

// Abstract base class implementing ICustomer
abstract class BaseCustomer : ICustomer
{
    public string Name { get; set; }
    public BuildingType StructureType { get; set; }
    public double Size { get; set; }
    public double LightBulbs { get; set; }
    public double Outlets { get; set; }
    public string CreditCard { get; set; }

    public BaseCustomer(string name, BuildingType type, double size,
double lightBulbs, double outlets, string creditCard)
    {
        Name = name;
        StructureType = type;
        Size = size;
        LightBulbs = lightBulbs;
        Outlets = outlets;
        CreditCard = MaskCreditCard(creditCard);
    }

    private string MaskCreditCard(string cardNumber)
    {
        return cardNumber.Substring(0, 4) + " XXXX XXXX " +
cardNumber.Substring(12, 4);
    }

    public virtual void DisplayCustomerInfo()
    {
        Console.WriteLine($"{Name} | {StructureType} | {Size} sq.ft |
{LightBulbs} bulbs | {Outlets} outlets | Card: {CreditCard}");
```

```
    }

    public abstract void PerformWiringTasks();
}

// Derived class for different buildings
class Customer : BaseCustomer
{
    public Customer(string name, BuildingType type, double size, double
lightBulbs, double outlets, string creditCard)
        : base(name, type, size, lightBulbs, outlets, creditCard) { }

    public override void PerformWiringTasks()
    {
        Console.WriteLine($"Creating wiring schema for
{StructureType}...");
        Console.WriteLine("Purchasing necessary parts...");

        switch (StructureType)
        {
            case BuildingType.House:
                Console.WriteLine("Installing fire alarms...");
                break;
            case BuildingType.Barn:
                Console.WriteLine("Wiring milking equipment...");
                break;
            case BuildingType.Garage:
                Console.WriteLine("Installing automatic doors...");
                break;
        }
    }
}
class Program
{
    static void Main()
    {
        List<ICustomer> customers = new List<ICustomer>();
        string continueInput;

        do
        {
            Console.Write("Enter customer name: ");
            string name = Console.ReadLine();

            Console.Write("Enter building type (House, Barn, Garage): ");
            BuildingType type;
            while (!Enum.TryParse(Console.ReadLine(), true, out type))
            {
                Console.Write("Invalid input. Enter building type (House,
Barn, Garage): ");
            }

            double size;
            do
            {
                Console.Write("Enter building size (1000 - 50000 sq.ft):
");
            } while (!double.TryParse(Console.ReadLine(), out size) ||
size < 1000 || size > 50000);

            double bulbs;
            do
            {
                Console.Write("Enter number of light bulbs (max 20): ");
            } while (!double.TryParse(Console.ReadLine(), out bulbs) ||
bulbs < 0 || bulbs > 20);

            double outlets;
            do
            {
                Console.Write("Enter number of outlets (max 50): ");
            } while (!double.TryParse(Console.ReadLine(), out outlets) ||
outlets < 0 || outlets > 50);

            string creditCard;
            do
            {
                Console.Write("Enter 16-digit credit card number: ");
                creditCard = Console.ReadLine();
            } while (creditCard.Length != 16 || !long.TryParse(creditCard,
out _));

            // Create a new customer
            ICustomer newCustomer = new Customer(name, type, size, bulbs,
outlets, creditCard);
            customers.Add(newCustomer);

            // Perform wiring tasks
            newCustomer.PerformWiringTasks();

            Console.Write("Do you want to enter another customer?
(yes/no): ");
            continueInput = Console.ReadLine().ToLower();

        } while (continueInput == "yes");

        // Display all customers
        Console.WriteLine("\nCustomer Summary:");

Console.WriteLine("-------------------------------------------------------
-----");
        foreach (var customer in customers)
        {
            customer.DisplayCustomerInfo();
        }
    }
}
```

Output:

With 3 data entered



```
Enter customer name: Shah, Yash Ketanbhai
Enter building type (House, Barn, Garage): House
Enter building size (1000 - 50000 sq.ft): 1234
Enter number of light bulbs (max 20): 13
Enter number of outlets (max 50): 43
Enter 16-digit credit card number: 2222 4053 4324 8877
Enter 16-digit credit card number: 2222405343248877
Creating wiring schema for House...
Purchasing necessary parts...
Installing fire alarms...
Do you want to enter another customer? (yes/no): yes
Enter customer name: Raj Rana
Enter building type (House, Barn, Garage): Barn
Enter building size (1000 - 50000 sq.ft): 2344
Enter number of light bulbs (max 20): 11
Enter number of outlets (max 50): 32
Enter 16-digit credit card number: 3056930902593404
Creating wiring schema for Barn...
Purchasing necessary parts...
Wiring milking equipment...
Do you want to enter another customer? (yes/no): yes
Enter customer name: Harsh Barot
Enter building type (House, Barn, Garage): Garage
Enter building size (1000 - 50000 sq.ft): 4321
Enter number of light bulbs (max 20): 3
Enter number of outlets (max 50): 43
Enter 16-digit credit card number: 2346345309025904
Creating wiring schema for Garage...
Purchasing necessary parts...
Installing automatic doors...
Do you want to enter another customer? (yes/no): no

Customer Summary:
------------------------------------------------------------
Shah, Yash Ketanbhai  | House | 1234 sq.ft | 13 bulbs | 43 outlets | Card: 2222 XXXX XXXX 8877
Raj Rana | Barn | 2344 sq.ft | 11 bulbs | 32 outlets | Card: 3056 XXXX XXXX 3404
Harsh Barot | Garage | 4321 sq.ft | 3 bulbs | 43 outlets | Card: 2346 XXXX XXXX 5904

D:\ElectricianTaskManager\ElectricianTaskManager\bin\Debug\net8.0\ElectricianTaskManager.exe (process 400) exited with code 0 (0x0).
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

1. **Interface**: ICustomer

```
// Here we have defined an interface for customer functionality
interface ICustomer
{
    string Name { get; set; }
    BuildingType StructureType { get; set; }
    double Size { get; set; }
    double LightBulbs { get; set; }
    double Outlets { get; set; }
    string CreditCard { get; set; }
    void DisplayCustomerInfo();
    void PerformWiringTasks();
}
```

An interface defining the blueprint for customer-related functionalities.

2. **Enumeration**: BuildingType

```
// Enum for building type
enum BuildingType
{
    House,
    Barn,
    Garage
}
```

3. **Abstract Base Class**: BaseCustomer

Implements ICustomer and provides base functionality for customer objects.

```
// Abstract base class implementing ICustomer
abstract class BaseCustomer : ICustomer
{
    public string Name { get; set; }
    public BuildingType StructureType { get; set; }
    public double Size { get; set; }
    public double LightBulbs { get; set; }
    public double Outlets { get; set; }
    public string CreditCard { get; set; }

    public BaseCustomer(string name, BuildingType type, double size,
double lightBulbs, double outlets, string creditCard)
    {
        Name = name;
        StructureType = type;
        Size = size;
        LightBulbs = lightBulbs;
        Outlets = outlets;
        CreditCard = MaskCreditCard(creditCard);
    }

    private string MaskCreditCard(string cardNumber)
    {
        return cardNumber.Substring(0, 4) + " XXXX XXXX " +
cardNumber.Substring(12, 4);
    }

    public virtual void DisplayCustomerInfo()
    {
        Console.WriteLine($"{Name} | {StructureType} | {Size} sq.ft |
{LightBulbs} bulbs | {Outlets} outlets | Card: {CreditCard}");
    }

    public abstract void PerformWiringTasks();
}
```

4. **Concrete Class**: Customer

Inherits from BaseCustomer and provides implementation for PerformWiringTasks.

```
// Derived class for different buildings
class Customer : BaseCustomer
{
    public Customer(string name, BuildingType type, double size, double
lightBulbs, double outlets, string creditCard)
        : base(name, type, size, lightBulbs, outlets, creditCard) { }

    public override void PerformWiringTasks()
    {
        Console.WriteLine($"Creating wiring schema for
{StructureType}...");
        Console.WriteLine("Purchasing necessary parts...");

        switch (StructureType)
        {
            case BuildingType.House:
                Console.WriteLine("Installing fire alarms...");
                break;
            case BuildingType.Barn:
                Console.WriteLine("Wiring milking equipment...");
                break;
            case BuildingType.Garage:
                Console.WriteLine("Installing automatic doors...");
                break;
        }
    }
}
```

5. **Program Execution**: *Main* Method

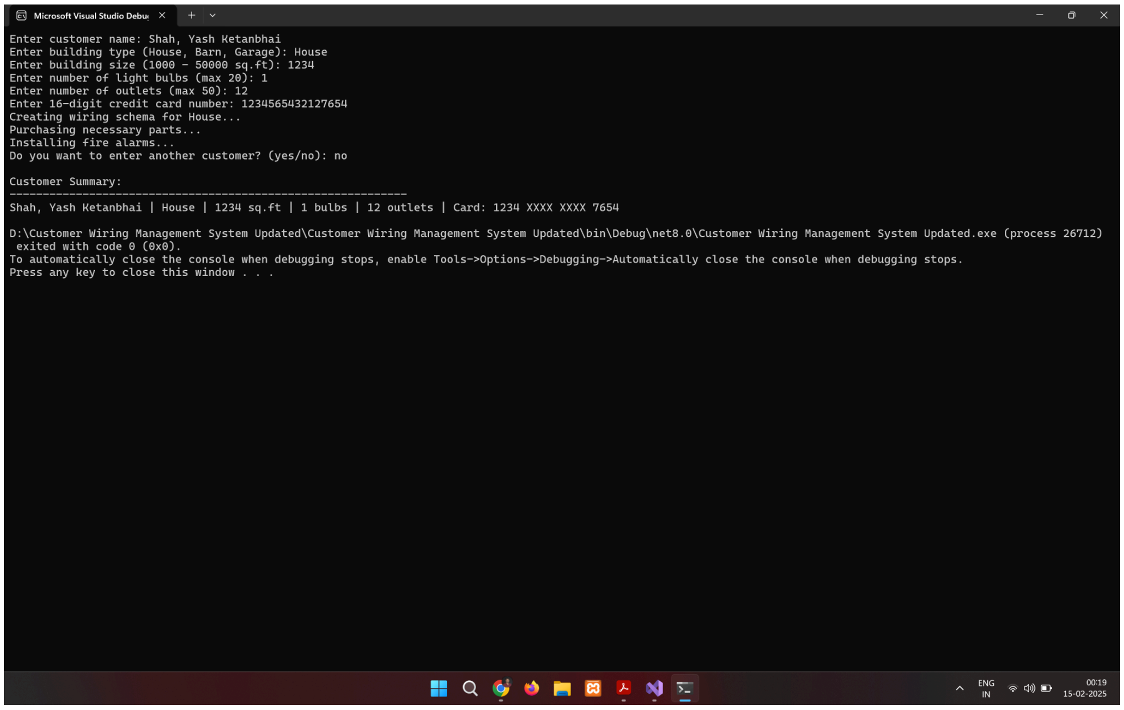
Handles customer data collection and task execution.

```
class Program
{
    static void Main()
    {
        List<ICustomer> customers = new List<ICustomer>();
        string continueInput;

        do
        {
            Console.Write("Enter customer name: ");
            string name = Console.ReadLine();

            Console.Write("Enter building type (House, Barn, Garage): ");
            BuildingType type;
            while (!Enum.TryParse(Console.ReadLine(), true, out type))
            {
                Console.Write("Invalid input. Enter building type (House,
Barn, Garage): ");
```

```
            }

            double size;
            do
            {
                Console.Write("Enter building size (1000 - 50000 sq.ft):
");
            } while (!double.TryParse(Console.ReadLine(), out size) ||
size < 1000 || size > 50000);

            double bulbs;
            do
            {
                Console.Write("Enter number of light bulbs (max 20): ");
            } while (!double.TryParse(Console.ReadLine(), out bulbs) ||
bulbs < 0 || bulbs > 20);

            double outlets;
            do
            {
                Console.Write("Enter number of outlets (max 50): ");
            } while (!double.TryParse(Console.ReadLine(), out outlets) ||
outlets < 0 || outlets > 50);

            string creditCard;
            do
            {
                Console.Write("Enter 16-digit credit card number: ");
                creditCard = Console.ReadLine();
            } while (creditCard.Length != 16 || !long.TryParse(creditCard,
out _));

            // Create a new customer
            ICustomer newCustomer = new Customer(name, type, size, bulbs,
outlets, creditCard);
            customers.Add(newCustomer);

            // Perform wiring tasks
            newCustomer.PerformWiringTasks();
            Console.Write("Do you want to enter another customer?
(yes/no): ");
            continueInput = Console.ReadLine().ToLower();

        } while (continueInput == "yes");

        // Display all customers
        Console.WriteLine("\nCustomer Summary:");

Console.WriteLine("-------------------------------------------------------
-----");
        foreach (var customer in customers)
        {
            customer.DisplayCustomerInfo();
        }
    }
}
```

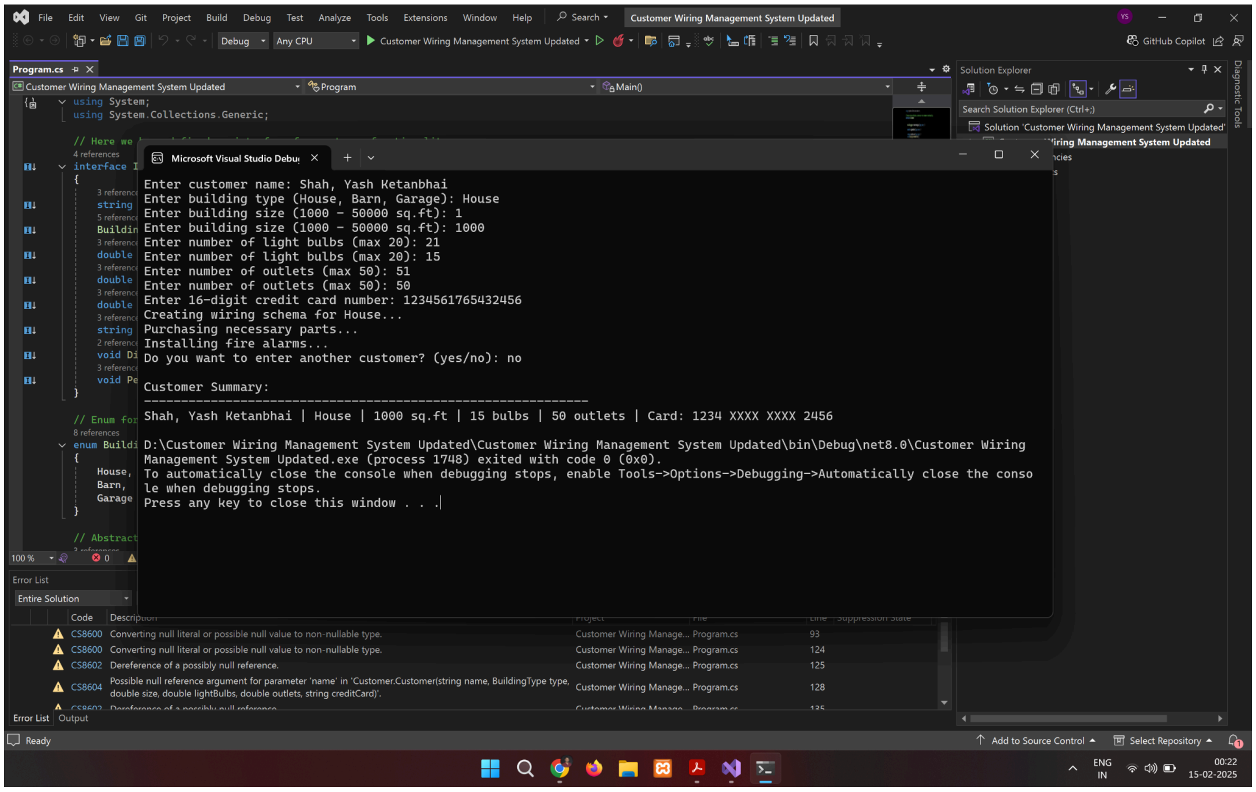
Output:

If the user enters correct information:



```
Enter customer name: Shah, Yash Ketanbhai
Enter building type (House, Barn, Garage): House
Enter building size (1000 - 50000 sq.ft): 1234
Enter number of light bulbs (max 20): 1
Enter number of outlets (max 50): 12
Enter 16-digit credit card number: 1234565432127654
Creating wiring schema for House...
Purchasing necessary parts...
Installing fire alarms...
Do you want to enter another customer? (yes/no): no

Customer Summary:
------------------------------------------------------------
Shah, Yash Ketanbhai | House | 1234 sq.ft | 1 bulbs | 12 outlets | Card: 1234 XXXX XXXX 7654

D:\Customer Wiring Management System Updated\Customer Wiring Management System Updated\bin\Debug\net8.0\Customer Wiring Management System Updated.exe (process 26712)
 exited with code 0 (0x0).
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

If the user does not enter correct information



```
Enter customer name: Shah, Yash Ketanbhai
Enter building type (House, Barn, Garage): House
Enter building size (1000 - 50000 sq.ft): 1
Enter building size (1000 - 50000 sq.ft): 1000
Enter number of light bulbs (max 20): 21
Enter number of light bulbs (max 20): 15
Enter number of outlets (max 50): 51
Enter number of outlets (max 50): 50
Enter 16-digit credit card number: 1234561765432456
Creating wiring schema for House...
Purchasing necessary parts...
Installing fire alarms...
Do you want to enter another customer? (yes/no): no

Customer Summary:
------------------------------------------------------------
Shah, Yash Ketanbhai | House | 1000 sq.ft | 15 bulbs | 50 outlets | Card: 1234 XXXX XXXX 2456

D:\Customer Wiring Management System Updated\Customer Wiring Management System Updated\bin\Debug\net8.0\Customer Wiring
Management System Updated.exe (process 1748) exited with code 0 (0x0).
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

Code	Description	Project	File	Line
CS8600	Converting null literal or possible null value to non-nullable type.	Customer Wiring Manage...	Program.cs	93
CS8600	Converting null literal or possible null value to non-nullable type.	Customer Wiring Manage...	Program.cs	124
CS8602	Dereference of a possibly null reference.	Customer Wiring Manage...	Program.cs	125
CS8604	Possible null reference argument for parameter 'name' in 'Customer.Customer(string name, BuildingType type, double size, double lightBulbs, double outlets, string creditCard)'.	Customer Wiring Manage...	Program.cs	128
CS8602	Dereference of a possibly null reference.	Customer Wiring Manage...	Program.cs	135

References:

1. Microsoft Docs - C# Interfaces:
   https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/interfaces/
2. Microsoft Docs - Abstract and Sealed Classes:
   https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/abstract-and-sealed-classes-and-class-members
3. Microsoft Docs - Enums in C#:
   https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/enum