1. **Create a class called Employee with properties for name, age, and salary. Implement a method to display employee details.**

```
using System;

public class Employee
{
   public string Name { get; set; }
   public int Age { get; set; }
   public double Salary { get; set; }

   public void DisplayDetails()
   {
      Console.WriteLine($"Name: {Name}, Age: {Age}, Salary: {Salary}");
   }
}

// Usage
class Program
{
   static void Main()
   {
      Employee emp = new Employee { Name = "Aman", Age = 25, Salary = 50000 };
      emp.DisplayDetails();
   }
}
```

2. **Create a class called Bank Account with properties for account number, account holder name, and balance. Implement methods for deposit, withdrawal, and displaying the account details. Static Class:**

```
using System;

public class BankAccount
{
   public int AccountNumber { get; set; }
   public string AccountHolder { get; set; }
   public double Balance { get; private set; }

   public void Deposit(double amount)
   {
      Balance += amount;
      Console.WriteLine($"Deposited {amount}. New Balance: {Balance}");
```

```
    }

    public void Withdraw(double amount)
    {
        if (amount <= Balance)
        {
            Balance -= amount;
            Console.WriteLine($"Withdrawn {amount}. Remaining Balance: {Balance}");
        }
        else
        {
            Console.WriteLine("Insufficient balance!");
        }
    }

    public void DisplayAccount()
    {
        Console.WriteLine($"Account Number: {AccountNumber}, Holder:
{AccountHolder}, Balance: {Balance}");
    }
}

// Usage
class Program
{
    static void Main()
    {
        BankAccount acc = new BankAccount { AccountNumber = 12345, AccountHolder
= "Aman" };
        acc.Deposit(5000);
        acc.Withdraw(2000);
        acc.DisplayAccount();
    }
}
```

3. **Create a static utility class named Math Helper with a static method Calculate Average that takes an array of integers as input and returns their average.**

```
public static class MathHelper
{
   public static double CalculateAverage(int[] numbers)
   {
      double sum = 0;
      foreach (int num in numbers)
         sum += num;

      return (numbers.Length > 0) ? sum / numbers.Length : 0;
   }
}

// Usage
class Program
{
   static void Main()
   {
      int[] nums = { 10, 20, 30, 40, 50 };
      Console.WriteLine("Average: " + MathHelper.CalculateAverage(nums));
   }
}
```

4. **Implement a static logger class called Logger that has a method Log Message for writing messages on console. Demonstrate its usage in a simple console application. Partial Class:**

```
using System;

public static class Logger
{
   public static void LogMessage(string message)
   {
      Console.WriteLine($"[LOG] {DateTime.Now}: {message}");
   }
}

// Usage
class Program
{
   static void Main()
   {
```

```
            Logger.LogMessage("Application started");
            Logger.LogMessage("Performing some tasks...");
            Logger.LogMessage("Application ended");
        }
    }
```

5. **Define a partial class Person with one part containing properties like First Name and Last Name, and another part with methods like Print FullName to display the full name. Implement these parts in separate files.**

Person1.cs

```
public partial class Person
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
}
```

Person2.cs

```
using System;

public partial class Person
{
    public void PrintFullName()
    {
        Console.WriteLine($"{FirstName} {LastName}");
    }
}
```

Usage

```
class Program
{
    static void Main()
    {
        Person p = new Person { FirstName = "Aman", LastName = "Kushwah" };
        p.PrintFullName();
    }
}
```

6. **Create a partial class Employee with properties representing employee details. In another part, implement methods for calculating salary based on different factors. Abstract Class:**

Employee1.cs

```
public partial class Employee
{
   public string Name { get; set; }
   public double BaseSalary { get; set; }
}
```

Employee2.cs

```
using System;

public partial class Employee
{
   public double CalculateSalary(double bonus, double deductions)
   {
      return BaseSalary + bonus - deductions;
   }
}
```

Usage

```
class Program
{
   static void Main()
   {
      Employee emp = new Employee { Name = "Aman", BaseSalary = 40000 };
      Console.WriteLine($"Final Salary: {emp.CalculateSalary(5000, 2000)}");
   }
}
```

7. **Define an abstract base class Shape with an abstract method Calculate Area. Derive classes like Circle and Rectangle from Shape and implement the area calculation methods for each.**

```
using System;

public abstract class Shape
{
   public abstract double CalculateArea();
}

public class Circle : Shape
{
   public double Radius { get; set; }
   public override double CalculateArea() => Math.PI * Radius * Radius;
}

public class Rectangle : Shape
{
   public double Length { get; set; }
   public double Width { get; set; }
   public override double CalculateArea() => Length * Width;
}

// Usage
class Program
{
   static void Main()
   {
      Shape c = new Circle { Radius = 5 };
      Shape r = new Rectangle { Length = 10, Width = 4 };
      Console.WriteLine($"Circle Area: {c.CalculateArea()}");
      Console.WriteLine($"Rectangle Area: {r.CalculateArea()}");
   }
}
```

8. **Design an abstract class Animal with properties like Name and Age. Derive classes like Dog and Cat from Animal with their unique methods. Sealed Class:**

```
using System;

public abstract class Animal
{
   public string Name { get; set; }
   public int Age { get; set; }

   public abstract void Speak();
}

public class Dog : Animal
{
   public override void Speak() => Console.WriteLine($"{Name} barks!");
}

public class Cat : Animal
{
   public override void Speak() => Console.WriteLine($"{Name} meows!");
}

// Usage
class Program
{
   static void Main()
   {
      Animal d = new Dog { Name = "Tommy", Age = 3 };
      Animal c = new Cat { Name = "Kitty", Age = 2 };

      d.Speak();
      c.Speak();
   }
}
```

9. **Create a base class Vehicle with methods like Start Engine and Stop Engine. Derive a class Car from Vehicle and seal it. Try to create a class that inherits from Car and observe the behavior.**

```
using System;


public class Vehicle

{

   public void StartEngine() => Console.WriteLine("Engine started");

   public void StopEngine() => Console.WriteLine("Engine stopped");

}


public sealed class Car : Vehicle

{

   public void Drive() => Console.WriteLine("Car is driving...");

}


// □ This will cause an error because Car is sealed

/*

public class SportsCar : Car { }

*/


class Program

{

   static void Main()

   {

      Car c = new Car();

      c.StartEngine();
```

```
        c.Drive();
        c.StopEngine();
    }
}using System;

public class Vehicle
{
    public void StartEngine() => Console.WriteLine("Engine started");
    public void StopEngine() => Console.WriteLine("Engine stopped");
}

public sealed class Car : Vehicle
{
    public void Drive() => Console.WriteLine("Car is driving...");
}
class Program
{
    static void Main()
    {
        Car c = new Car();
        c.StartEngine();
        c.Drive();
        c.StopEngine();
    }
}
```

10. Design a class BankAccount with properties like Account Number and Balance. Implement a sealed class SavingsAccount that extends BankAccount with methods for interest calculation.

```
using System;


public class BankAccount
{
    public int AccountNumber { get; set; }
    public double Balance { get; set; }
}


public sealed class SavingsAccount : BankAccount
{
    public double InterestRate { get; set; }


    public double CalculateInterest()
    {
        return Balance * InterestRate / 100;
    }
}


// Usage
class Program
{
    static void Main()
    {
```

```
        SavingsAccount acc = new SavingsAccount { AccountNumber = 101, Balance = 10000,
InterestRate = 5 };

        Console.WriteLine($"Interest: {acc.CalculateInterest()}");

    }

}
```