

1. Write a program to demonstrate class, constructor, properties and method.

using System;

class MobilePhone

```
{  
    public string Brand { get; set; }  
    public string Model { get; set; }  
    public double Price { get; set; }  
    public MobilePhone(string brand, string model, double price)  
    {  
        Brand = brand;  
        Model = model;  
        Price = price;  
    }  
    public void DisplayPhoneInfo()  
    {  
        Console.WriteLine($"Phone Info: {Brand} {Model}, Price: ${Price}");  
    }  
    public void ApplyDiscount(double discountPercentage)  
    {  
        Price -= Price * (discountPercentage / 100);  
        Console.WriteLine($"New Price after {discountPercentage}% discount: ${Price}");  
    }  
}  
  
class Program  
{  
    static void Main(string[] args)
```

```
{  
    MobilePhone myPhone = new MobilePhone("Samsung", "Galaxy S23", 999.99);  
    myPhone.DisplayPhoneInfo();  
    Console.WriteLine($"Phone Brand: {myPhone.Brand}");  
    myPhone.Brand = "Apple";  
    Console.WriteLine($"Updated Phone Brand: {myPhone.Brand}");  
    myPhone.ApplyDiscount(10);  
}
```

Output:

```
Phone Info: Samsung Galaxy S23, Price: $999.99  
Phone Brand: Samsung  
Updated Phone Brand: Apple  
New Price after 10% discount: $899.991
```

2. Write a program to demonstrate method overloading.

using System;

class Calculator

```
{  
    public int Add(int a, int b)  
    {  
        return a + b;  
    }  
    public int Add(int a, int b, int c)  
    {  
        return a + b + c;  
    }  
    public double Add(double a, double b)  
    {  
        return a + b;  
    }  
}
```

class Program

```
{  
    static void Main(string[] args)  
    {  
        Calculator calc = new Calculator();  
        Console.WriteLine("Add two integers: " + calc.Add(5, 10));  
        Console.WriteLine("Add three integers: " + calc.Add(5, 10, 15));  
        Console.WriteLine("Add two doubles: " + calc.Add(5.5, 10.5));  
    }  
}
```

```
}  
}
```

Output:

```
Add two integers: 15  
Add three integers: 30  
Add two doubles: 16
```

3. Create a class Calculate which contains data member num1 and num2 both in integer and methods setData() to set the data, FindSum() that calculate the sum of num1 and num2 and display the result, FindMulti() that calculate the multiplication of num1 and num2 and returns the result, FindDifference( ) that calculate the difference between num1 and num2 and display the result. Now, create some instance of Calculate and invoke all the methods.

Code:

```
using System;
```

```
class Calculate
```

```
{
```

```
    private int num1;
```

```
    private int num2;
```

```
    public void SetData(int a, int b)
```

```
    {
```

```
        num1 = a;
```

```
        num2 = b;
```

```
    }
```

```
    public void FindSum()
```

```
    {
```

```
        int sum = num1 + num2;
```

```
        Console.WriteLine("Sum: " + sum);
```

```
    }
```

```
    public int FindMulti()
```

```
    {
```

```
        return num1 * num2;
```

```
    }
```

```
    public void FindDifference()
```

```
    {
```

```
        int diff = num1 - num2;

        Console.WriteLine("Difference: " + diff);
    }
}

class Program
{
    static void Main(string[] args)
    {
        Calculate calc1 = new Calculate();

        Console.Write("Enter first number: ");

        int n1 = int.Parse(Console.ReadLine());

        Console.Write("Enter second number: ");

        int n2 = int.Parse(Console.ReadLine());

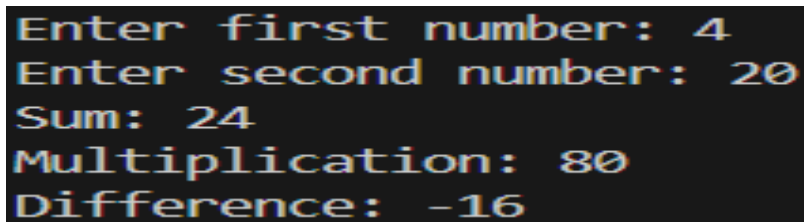
        calc1.SetData(n1, n2);

        calc1.FindSum();

        Console.WriteLine("Multiplication: " + calc1.FindMulti());

        calc1.FindDifference();
    }
}
```

Output:

A screenshot of a terminal window with a black background and white text. The text shows the program's execution: it prompts for the first number (4) and second number (20), then displays the calculated sum (24), multiplication (80), and difference (-16).

```
Enter first number: 4
Enter second number: 20
Sum: 24
Multiplication: 80
Difference: -16
```

4. Create a class Number having instance variable x and y both in integer, default constructor that set the value of x and y to 0, parameterized constructor that sets the value of x and y, method DecideOdd() that calculates the even no. occurring between x and y and display the result, DecideEven() that calculates the odd no. occurring between x and y and display the results. Now, create some instance of Number and invoke all the methods.

Code:

```
using System;
```

```
class Number
```

```
{
```

```
    private int x;
```

```
    private int y;
```

```
    public Number()
```

```
    {
```

```
        x = 0;
```

```
        y = 0;
```

```
    }
```

```
    public Number(int a, int b)
```

```
    {
```

```
        x = a;
```

```
        y = b;
```

```
    }
```

```
    public void DecideEven()
```

```
    {
```

```
        Console.WriteLine($"Even numbers between {x} and {y}:");
```

```
        for (int i = x; i <= y; i++)
```

```
        {
```

```
            if (i % 2 == 0)
```

```

        {
            Console.Write(i + " ");
        }
    }

    Console.WriteLine(); // For new line
}

public void DecideOdd()
{
    Console.WriteLine($"Odd numbers between {x} and {y}:");
    for (int i = x; i <= y; i++)
    {
        if (i % 2 != 0)
        {
            Console.Write(i + " ");
        }
    }

    Console.WriteLine(); // For new line
}
}

class Program
{
    static void Main(string[] args)
    {
        Console.Write("Enter starting number (x): ");

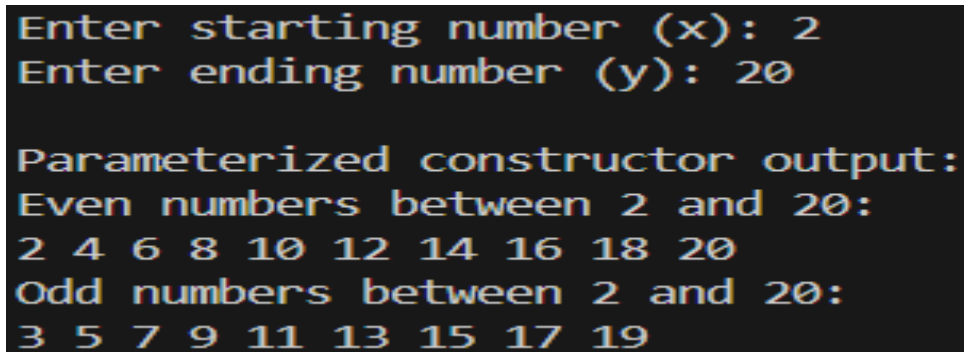
        int x = int.Parse(Console.ReadLine());
    }
}

```



```
Console.Write("Enter ending number (y): ");  
  
int y = int.Parse(Console.ReadLine());  
  
Number num2 = new Number(x, y);  
  
Console.WriteLine("\nParameterized constructor output:");  
  
num2.DecideEven();  
  
num2.DecideOdd();  
  
}  
  
}
```

Output:



```
Enter starting number (x): 2  
Enter ending number (y): 20  
  
Parameterized constructor output:  
Even numbers between 2 and 20:  
2 4 6 8 10 12 14 16 18 20  
Odd numbers between 2 and 20:  
3 5 7 9 11 13 15 17 19
```

5. Create a class Shape that contains instance variable length, breadth and height. Create a default constructor that sets the value of instance variable to zero, constructor with two parameter that will sets the value of length and breadth only and constructor with three parameter that will sets the value of length, breadth and height. After this create FindAreaRectangle() that calculates the area of rectangle, FindVolumeBox() that calculates volume of box and display the result. Now create first object of Shape which will have name rectangle and calls constructor with twoparameter and FindAreaRectangle() method, create second object of Shape that will have name Box which will call constructor with three parameter and FindVolumeBox() method.

Code:

```
using System;
```

```
class Shape
```

```
{
```

```
    private int length;
```

```
    private int breadth;
```

```
    private int height;
```

```
    public Shape()
```

```
    {
```

```
        length = 0;
```

```
        breadth = 0;
```

```
        height = 0;
```

```
    }
```

```
    public Shape(int l, int b)
```

```
    {
```

```
        length = l;
```

```
        breadth = b;
```

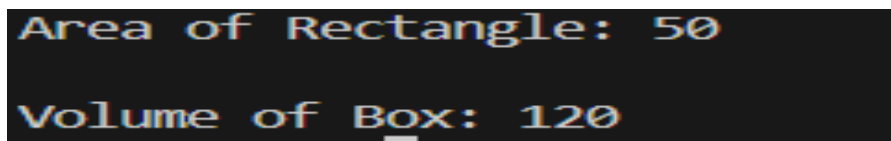
```
        height = 0;
```

```
    }
```

```
    public Shape(int l, int b, int h)
    {
        length = l;
        breadth = b;
        height = h;
    }
    public int FindAreaRectangle()
    {
        int area = length * breadth;
        Console.WriteLine("Area of Rectangle: " + area);
        return area;
    }
    public int FindVolumeBox()
    {
        int volume = length * breadth * height;
        Console.WriteLine("Volume of Box: " + volume);
        return volume;
    }
}
class Program
{
    static void Main(string[] args)
    {
        Shape rectangle = new Shape(10, 5);
        rectangle.FindAreaRectangle();
        Console.WriteLine();
    }
}
```

```
Shape box = new Shape(4, 5, 6);  
box.FindVolumeBox();  
}  
}
```

Output:



```
Area of Rectangle: 50  
Volume of Box: 120
```

6. Create a class EmployeeDetails having data member empId, empName, empGender, empAddress, and empPosition, constructor to set the details and display method to display the details. Create a subclass SalaryInfo that will inherit EmployeeDetails having own data member salary which will record salary per year, constructor to set the value of salary and method calcTax() that deduct the tax of salary and display the final salary. Tax rate is given as (if salary <= 400000 tax is 1%, salary between 400001 to 800000 tax is 10% and salary > 800000 tax 20%). Now create the object of Salary info and demonstrate the scenario.

Code:

```
using System;
```

```
class EmployeeDetails
```

```
{
```

```
    protected int empId;
```

```
    protected string empName;
```

```
    protected string empGender;
```

```
    protected string empAddress;
```

```
    protected string empPosition;
```

```
    public EmployeeDetails(int id, string name, string gender, string address, string position)
```

```
    {
```

```
        empId = id;
```

```
        empName = name;
```

```
        empGender = gender;
```

```
        empAddress = address;
```

```
        empPosition = position;
```

```
    }
```

```
    public void Display()
```

```
    {
```

```
        Console.WriteLine("Employee ID: " + empId);
```

```
        Console.WriteLine("Employee Name: " + empName);
```

```

        Console.WriteLine("Gender: " + empGender);
        Console.WriteLine("Address: " + empAddress);
        Console.WriteLine("Position: " + empPosition);
    }
}

class SalaryInfo : EmployeeDetails
{
    private double salary;

    public SalaryInfo(int id, string name, string gender, string address, string position, double sal)
        : base(id, name, gender, address, position)
    {
        salary = sal;
    }

    public void CalcTax()
    {
        double taxRate = 0;
        if (salary <= 400000)
        {
            taxRate = 0.01;
        }
        else if (salary > 400000 && salary <= 800000)
        {
            taxRate = 0.10;
        }
        else if (salary > 800000)

```

```

{
    taxRate = 0.20;
}

double taxAmount = salary * taxRate;

double finalSalary = salary - taxAmount;

Console.WriteLine("Original Salary: " + salary);

Console.WriteLine("Tax Deducted: " + taxAmount);

Console.WriteLine("Final Salary after Tax: " + finalSalary);

} }

```

class Program

```

{
    static void Main(string[] args)
    {
        SalaryInfo emp1 = new SalaryInfo(101, "Alice Johnson", "Female", "New York", "Manager",
850000);

        Console.WriteLine("--- Employee Details ---");

        emp1.Display();

        Console.WriteLine("\n--- Salary Details ---");

        emp1.CalcTax();

    }
}

```

Output:

```

--- Employee Details ---
Employee ID: 101
Employee Name: Alice Johnson
Gender: Female
Address: New York
Position: Manager

--- Salary Details ---
Original Salary: 850000
Tax Deducted: 170000
Final Salary after Tax: 680000

```

7. Write a program to demonstrate single level, multilevel inheritance.

```
using System;
```

```
class Animal
```

```
{
```

```
    public void Eat()
```

```
    {
```

```
        Console.WriteLine("Animal eats food.");
```

```
    }
```

```
}
```

```
class Dog : Animal
```

```
{
```

```
    public void Bark()
```

```
    {
```

```
        Console.WriteLine("Dog barks.");
```

```
    }
```

```
}
```

```
class Puppy : Dog
```

```
{
```

```
    public void Weep()
```

```
    {
```

```
        Console.WriteLine("Puppy weeps.");
```

```
    }
```

```
}
```

```
class Program
```

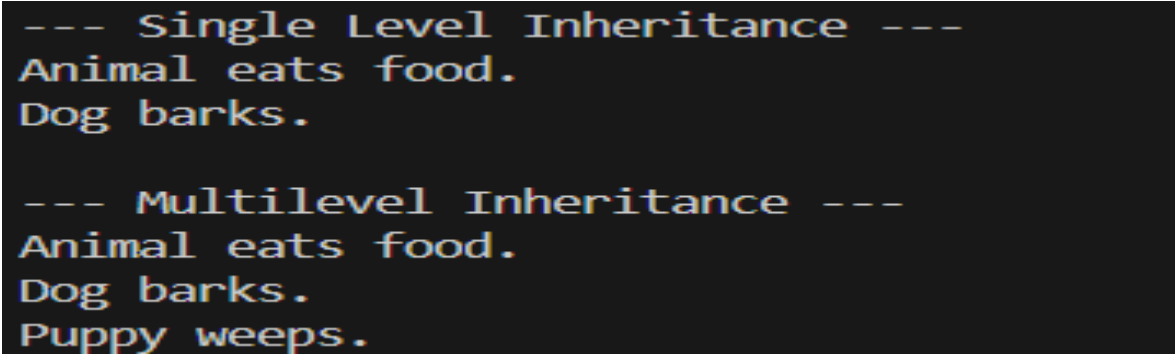
```
{
```

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



```
{  
    Console.WriteLine("--- Single Level Inheritance ---");  
    Dog dog = new Dog();  
    dog.Eat();  
    dog.Bark();  
    Console.WriteLine("\n--- Multilevel Inheritance ---");  
    Puppy puppy = new Puppy();  
    puppy.Eat();  
    puppy.Bark();  
    puppy.Weep();  
}  
}
```

Output:

A screenshot of a terminal window showing the output of the C# program. The text is displayed in a monospaced font with syntax highlighting: keywords like 'Animal', 'Dog', and 'Puppy' are in blue, and strings are in yellow. The output is organized into two sections separated by blank lines. The first section is titled '--- Single Level Inheritance ---' and shows 'Animal eats food.' and 'Dog barks.'. The second section is titled '--- Multilevel Inheritance ---' and shows 'Animal eats food.', 'Dog barks.', and 'Puppy weeps.'.

```
--- Single Level Inheritance ---  
Animal eats food.  
Dog barks.  
  
--- Multilevel Inheritance ---  
Animal eats food.  
Dog barks.  
Puppy weeps.
```

8. Write a program to demonstrate use of base keyword.

Code:

```
using System;

class Person
{
    public string name;

    public Person(string name)
    {
        this.name = name;
    }

    public void Display()
    {
        Console.WriteLine("Name: " + name);
    }
}

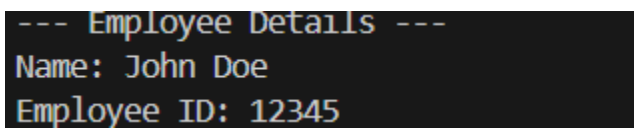
class Employee : Person
{
    public int empld;

    public Employee(string name, int id) : base(name)
    {
        empld = id;
    }

    public void ShowDetails()
    {
        base.Display(); // Calling base class method
    }
}
```

```
        Console.WriteLine("Employee ID: " + empld);
    }
}
class Program
{
    static void Main(string[] args)
    {
        Employee emp = new Employee("John Doe", 12345);
        Console.WriteLine("--- Employee Details ---");
        emp.ShowDetails();
    }
}
```

Output:

A screenshot of a terminal window showing the output of the program. The text is displayed on a black background with a light blue/teal color. The output consists of three lines: a separator line, the employee's name, and the employee's ID.

```
--- Employee Details ---
Name: John Doe
Employee ID: 12345
```

9. Write a program to demonstrate method overriding (dynamic polymorphism).

Code:

```
using System;
```

```
class Animal
```

```
{
```

```
    public virtual void MakeSound()
```

```
    {
```

```
        Console.WriteLine("Animal makes a sound.");
```

```
    }
```

```
}
```

```
class Dog : Animal
```

```
{
```

```
    public override void MakeSound()
```

```
    {
```

```
        Console.WriteLine("Dog barks.");
```

```
    }
```

```
}
```

```
class Cat : Animal
```

```
{
```

```
    public override void MakeSound()
```

```
    {
```

```
        Console.WriteLine("Cat meows.");
```

```
    }
```

```
}
```

```
class Program
```

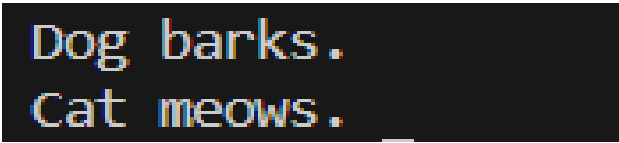
```
{
```

```
static void Main(string[] args)
{
    Animal animal;

    animal = new Dog();
    animal.MakeSound();

    animal = new Cat();
    animal.MakeSound();
}
}
```

Output:

A screenshot of a terminal window with a black background. The text 'Dog barks.' is displayed on the first line, and 'Cat meows.' is displayed on the second line. The text is in a light blue, monospaced font.

```
Dog barks.
Cat meows.
```

10. Write a program to demonstrate multiple inheritance using interface.

Code:

```
using System;
```

```
interface IPrintable
```

```
{
```

```
    void Print();
```

```
}
```

```
interface IScannable
```

```
{
```

```
    void Scan();
```

```
}
```

```
class MultiFunctionPrinter : IPrintable, IScannable
```

```
{
```

```
    public void Print()
```

```
    {
```

```
        Console.WriteLine("Printing document...");
```

```
    }
```

```
    public void Scan()
```

```
    {
```

```
        Console.WriteLine("Scanning document...");
```

```
    }
```

```
}
```

```
class Program
```

```
{
```

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

```
    {
```

```
MultiFunctionPrinter mfp = new MultiFunctionPrinter();  
  
Console.WriteLine("--- Multiple Inheritance using Interfaces ---");  
  
mfp.Print();  
  
mfp.Scan();  
  
}  
  
}
```

Output:

```
--- Multiple Inheritance using Interfaces ---  
Printing document...  
Scanning document...
```

11. Write a program to demonstrate abstract class.

Code:

using System;

abstract class Animal

{

public string Name { get; set; }

public abstract void MakeSound();

public void Eat()

{

Console.WriteLine(Name + " is eating.");

}

}

class Dog : Animal

{

public Dog(string name)

{

Name = name;

}

public override void MakeSound()

{

Console.WriteLine(Name + " barks.");

}

}

class Program

{

static void Main(string[] args)



```
{  
    Dog dog = new Dog("Rex");  
    dog.MakeSound();  
    dog.Eat();  
}  
}
```

Output:

```
Rex barks.  
Rex is eating.
```

12. Write a program to demonstrate exception handling (try, catch, throw).

Code:

```
using System;
```

```
class Program
```

```
{
```

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

```
    {
```

```
        try
```

```
        {
```

```
            try
```

```
            {
```

```
                int result = Divide(10, 0); // Will throw an exception
```

```
                Console.WriteLine("Result: " + result);
```

```
            }
```

```
        catch (DivideByZeroException ex)
```

```
        {
```

```
            Console.WriteLine("Error: " + ex.Message);
```

```
            throw new InvalidOperationException("Cannot divide by zero", ex); // Re-throwing
```

```
        }
```

```
    }
```

```
    catch (InvalidOperationException ex)
```

```
    {
```

```
        Console.WriteLine("Handled re-thrown exception: " + ex.Message);
```

```
    }
```

```
    catch (Exception ex)
```

```
{  
    Console.WriteLine("General error: " + ex.Message);  
}  
}  
  
static int Divide(int a, int b)  
{  
    if (b == 0)  
    {  
        throw new DivideByZeroException("Attempt to divide by zero");  
    }  
    return a / b;  
}  
}
```

Output:

```
Error: Attempt to divide by zero  
Handled re-thrown exception: Cannot divide by zero
```

13. Write a program to demonstrate interface.

Code:

```
using System;

interface IPlayable

{
    void Play();
}

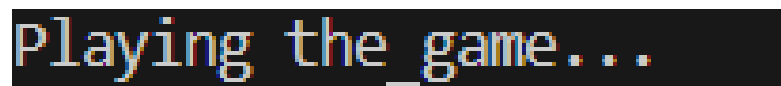
class Game : IPlayable

{
    public void Play()
    {
        Console.WriteLine("Playing the game...");
    }
}

class Program

{
    static void Main(string[] args)
    {
        Game game = new Game();
        game.Play();
    }
}
```

Output:



```
Playing the game...
```

14. Write a program to demonstrate lamda expression.

Code:

using System;

class Program

{

static void Main(string[] args)

{

Func<int, int, int> add = (a, b) => a + b;

int result = add(5, 10);

Console.WriteLine("Sum: " + result);

var numbers = new int[] { 1, 2, 3, 4, 5, 6 };

var evenNumbers = Array.FindAll(numbers, n => n % 2 == 0);

Console.WriteLine("Even numbers:");

foreach (var num in evenNumbers)

{

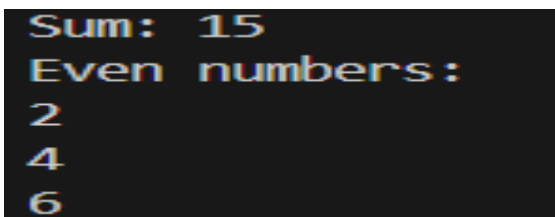
Console.WriteLine(num);

}

}

}

Output:

A screenshot of a terminal window with a black background and yellow text. The output shows the sum of 5 and 10 as 15, followed by the label 'Even numbers:' and a list of even numbers from the array: 2, 4, and 6, each on a new line.

```
Sum: 15
Even numbers:
2
4
6
```