2100030408

A.Sambasivarao

HOMEASSIGNMENT-2

1. Access Modifiers(public,private,protected,internal) using

```
System;

// Class with various access
modifiers public class MyClass {
public int myPublicField;    private
int myPrivateField;    protected int
myProtectedField;    internal int
myInternalField;

  // Public method    public void
MyPublicMethod() {
      Console.WriteLine("This is a public method.");
  }

  // Private method    private void
MyPrivateMethod() {
      Console.WriteLine("This is a private method.");
  }

  // Protected method    protected void
MyProtectedMethod() {
      Console.WriteLine("This is a protected method.");
  }

  // Internal method    internal void
MyInternalMethod() {
      Console.WriteLine("This is an internal method.");
```

```
    }
}

public class Program {     public static void Main(string[] args) {
MyClass obj = new MyClass();        obj.myPublicField = 10;        //
Accessible from anywhere        obj.myInternalField = 40;      // Accessible
within the same assembly        obj.MyPublicMethod();         // Accessible
from anywhere        obj.MyInternalMethod();        // Accessible within
the same assembly
    }
}
```

OUTPUT:

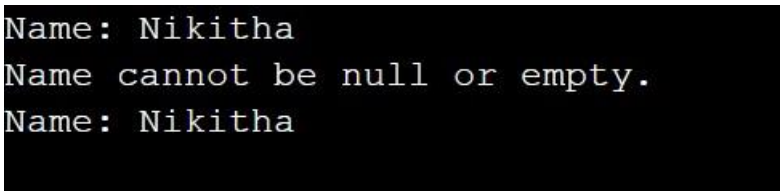
```
main.cs(6,17): warning CS0169: The private field `MyClass.myPrivateField' is never used
Compilation succeeded - 1 warning(s)
This is a public method.
This is an internal method.
```

2.

```
using System;

public class Person {
// Private field
private string name;

  // Property with set and get accessors
public string Name {
    get {
        return name; // Retrieve the value of the private field
    }
set {
        if (!string.IsNullOrEmpty(value)) {              name =
value; // Set the value of the private field
        } else {
            Console.WriteLine("Name cannot be null or empty.");
        }
```

```
    }
  }
}

public class Program {    public static
void Main(string[] args) {        Person
person = new Person();
person.Name = "Nikitha";
      Console.WriteLine("Name: " + person.Name);
person.Name = null;
      Console.WriteLine("Name: " + person.Name);
  }
}
```

Output:



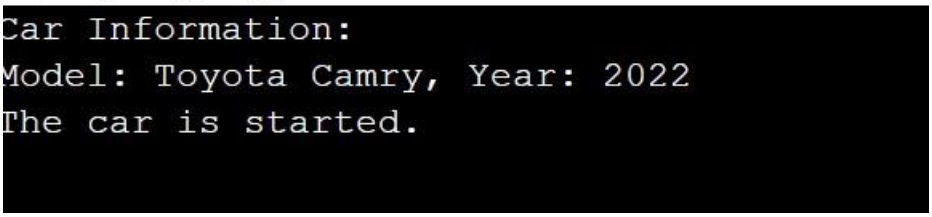
3.Encapsulation and Abstraction

```
using System;

// Example of encapsulation and abstraction with a Car class
public class Car {
  // Private fields encapsulated within the Car
class    private string model;    private int year;

  // Public properties providing controlled access to the private
fields    public string Model {        get { return model; }
      set { model = value; }
  }
```

```
    public int Year {
get { return year; }
        set {
            // Ensuring that the year is within a reasonable range (e.g., current year to 1900)
if (value <= DateTime.Now.Year && value >= 1900) {
                year = value;
            } else {
                Console.WriteLine("Invalid year!");
            }
        }
    }

    // Method to perform a car-specific action (Abstraction)
public void Start() {
        Console.WriteLine("The car is started.");
    }

    // Method to display car information (Abstraction)
public void DisplayInfo() {
        Console.WriteLine($"Model: {Model}, Year: {Year}");
    }
}

class Program {     static void
Main(string[] args) {

        Car myCar = new Car();

        myCar.Model = "Toyota Camry";
        myCar.Year = 2022;
```

```
        Console.WriteLine("Car Information:");
myCar.DisplayInfo();

        myCar.Start();
    }
}
```

Output:

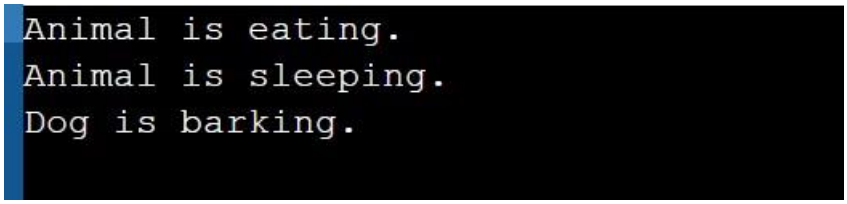


4.Inheritance using

```
System;

// Base class (superclass)
public class Animal {
public void Eat() {
        Console.WriteLine("Animal is eating.");
    }

    public void Sleep() {
        Console.WriteLine("Animal is sleeping.");
    }
}

// Derived class (subclass) inheriting from Animal
public class Dog : Animal {     public void Bark() {

        Console.WriteLine("Dog is barking.");
    }
}
```

```
class Program {     static void
Main(string[] args) {
        // Creating an instance of the derived class (Dog)
        Dog myDog = new Dog();

        // Accessing methods from the base class (Animal)
myDog.Eat();    // Inherited method
myDog.Sleep();  // Inherited method

        // Accessing method from the derived class (Dog)
myDog.Bark();
    }
}
```

Output:



5. Polymorphism(compile-time,run-time)

Comple-time using System;

```
public class MathOperations {
    // Method overloading (Compile-time polymorphism)
    public int Add(int a, int b) {
return a + b;
    }

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

```
class Program {     static void
Main(string[] args) {
        MathOperations math = new MathOperations();

        // Calling overloaded methods         int result1 = math.Add(5, 10);
// Calls the first Add method         double result2 = math.Add(3.5, 2.5);    //
Calls the second Add method

        Console.WriteLine("Result 1: " + result1);
Console.WriteLine("Result 2: " + result2);
    }
}
```

Run-time using

```
System;

public class Animal {     public
virtual void Sound() {
        Console.WriteLine("Animal makes a sound.");
    }
}

public class Dog : Animal {
    public override void Sound() {
        Console.WriteLine("Dog barks.");
    }
}
```

```
public class Cat : Animal {
public override void Sound() {
      Console.WriteLine("Cat meows.");
   }
}

class Program {     static void
Main(string[] args) {
      Animal myAnimal;

      myAnimal = new Dog();
myAnimal.Sound();   // Output: Dog barks.

      myAnimal = new Cat();
myAnimal.Sound();   // Output: Cat meows.
   }
}
```

6. Constructors-destructors using

```
System;

public class MyClass {
// Constructor     public
MyClass() {
Console.WriteLine("Co
nstructor called: Object
created.");

   }
```

```
    // Parameterized constructor
public MyClass(int value) {
        Console.WriteLine("Parameterized constructor called: Object created with value " + value);
    }

    // Destructor
    ~MyClass() {
        Console.WriteLine("Destructor called: Object destroyed.");
    }
}

class Program {     static void
Main(string[] args) {
        // Creating objects of MyClass
        MyClass obj1 = new MyClass();              // Calls the default constructor
        MyClass obj2 = new MyClass(10);            // Calls the parameterized constructor

        // Destructor will be called automatically when the objects go out of scope
    }
}
```

```
Compilation succeeded - 2 warning(s)
Constructor called: Object created.
Parameterized constructor called: Object created with value 10
Destructor called: Object destroyed.
Destructor called: Object destroyed.
```

7. SealedKeyword using

```
System;

// Base class public class
Vehicle {     public virtual
void Drive() {
        Console.WriteLine("Vehicle is being driven.");
```

```
    }
}

// Derived class inheriting from
Vehicle public class Car : Vehicle {
public sealed override void Drive() {
        Console.WriteLine("Car is being driven.");
    }
}




class Program {     static void Main(string[] args) {
Car myCar = new Car();         myCar.Drive();  //
Output: Car is being driven.

        // Attempting to inherit from Car and override the sealed method will result in a compilation
error
    }
}
```