C# Assignments on Inheritance

Lab 1. Single Inheritance

In Single Inheritance, a derived class inherits from a single base class.
Problem:
Create a base class Person with properties like Name and Age. Derive a class Student from Person and add a property Grade. Create objects and display the data.

```
public class Person
{
    public string Name { get; set; }
    public int Age { get; set; }

    public Person(string name, int age)
    {
        Name = name;
        Age = age;

    }

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

public class Student : Person
{
    public double Grade { get; set; }

    public Student(string name, int age, double grade) : base(name, age)
    {
        Grade = grade;
    }

    public override void DisplayInfo()
    {
        base.DisplayInfo();
        Console.WriteLine($"Grade: {Grade}");
    }
}
```

Main program:

```
{
```

```
    Person person1 = new Person("John Doe", 30);
    Student student1 = new Student("Jane Smith", 25, 3.8);

    Console.WriteLine("Person Information:");
    person1.DisplayInfo();

    Console.WriteLine("\nStudent Information:");
    student1.DisplayInfo();


}
```

Lab 2. Multilevel Inheritance

In Multilevel Inheritance, a class is derived from another derived class.
Problem:
Create a base class Animal with a method Eat(). Derive a class Dog that inherits Animal and add
a method Bark(). Further derive a class Puppy from Dog and add a method Weep(). Show the behavior.

```
public class Animal
{
    public void Eat()
    {
        Console.WriteLine("The animal is eating.");
    }
}
public class Dog : Animal
{
    public void Bark()
    {
        Console.WriteLine("The dog is barking.");
    }
}

public class Puppy : Dog
{
    public void Weep()
    {
        Console.WriteLine("The puppy is barking.");
    }
}
```

Main program:

```
 {
    Puppy puppy = new Puppy();
 puppy.Weep();
    puppy.Bark();
    puppy.Eat();

}
```

Lab 3. Multiple Inheritance (via Interfaces)

C# does not support multiple inheritance directly, but it can be achieved using interfaces.
Problem:
Create two interfaces IPrintable and IScannable with respective methods Print() and Scan().
Implement both interfaces in a class PrinterScanner.

```
interface IPrintable
{
   void Print();
}

interface IScannable
{
   void Scan();
}

class PrinterScanner : IPrintable, IScannable
{
   public void Print()
   {
      Console.WriteLine("Printing...");
   }

   public void Scan()
   {
      Console.WriteLine("Scanning...");
   }
}
```

Main program:

```
       {
          PrinterScanner ps = new PrinterScanner();
          ps.Print();
          ps.Scan();
```

```
    }
```

Lab 4. Hierarchical Inheritance

In Hierarchical Inheritance, multiple derived classes inherit from a single base class.
Problem:
Create a base class Shape with a method Draw(). Create two derived classes Circle and Rectangle, both inheriting from Shape, and override the Draw() method to show specific behavior.

```
public class Shape
{
    public virtual void Draw()
    {
        Console.WriteLine("Drawing a shape...");
    }
}

public class Circle : Shape
{
    public override void Draw()
    {
        Console.WriteLine("Drawing a circle...");

    }
}

public class Rectangle : Shape
{
    public override void Draw()
    {
        Console.WriteLine("Drawing a rectangle...");
    }
}
```

Main program:

```
{
    Shape circle = new Circle();
Shape rectangle = new Rectangle();

circle.Draw();
    rectangle.Draw();
}
```

Lab 5. Hybrid Inheritance (Using Interfaces)

Hybrid inheritance combines multiple types of inheritance. Since C# doesn't support direct multiple inheritance, hybrid inheritance is implemented using interfaces.

Problem:

Create two interfaces IMovable and IRechargeable. Create a class Vehicle implementing IMovable and derive ElectricCar from Vehicle implementing both interfaces.

```
public interface IMovable
{
    void Move();
}

public interface IRechargeable
{
    void Recharge();
}

public class Vehicle : IMovable
{
    public virtual void Move()
    {
        Console.WriteLine("Vehicle is moving.");
    }
}

public class ElectricCar : Vehicle, IRechargeable
{
    public override void Move()
    {
        Console.WriteLine("Electric car is moving silently.");
    }

    public void Recharge()
    {
        Console.WriteLine("Electric car is recharging.");
    }
}
```

Main program:

```
 {
     Vehicle vehicle = new Vehicle();
     vehicle.Move();

     ElectricCar electricCar = new ElectricCar();
```

```
    electricCar.Move();
    electricCar.Recharge();
}
```

Lab 6. Overriding Methods in Inheritance

Demonstrate method overriding where a base class method is overridden in the derived class.
Problem:
Create a class Employee with a method Work(). Derive a class Manager that overrides the Work() method to show a different implementation.

```
public class Employee
{
   public virtual void Work()
   {
      Console.WriteLine("Employee is working.");
   }
}

public class Manager : Employee
{
   public override void Work()
   {
      Console.WriteLine("Manager is working.");
   }
```

Main program:

```
   {
      Employee employee = new Employee();
employee.Work();

      Manager manager = new Manager();
manager.Work();
   }
```

Lab 7. Abstract Classes

Create an abstract class and demonstrate inheritance with abstract methods.
Problem:
Create an abstract class Vehicle with an abstract method Drive(). Create two derived classes Car and Bike that implement the Drive() method.

```
public abstract class Vehicle
   {
      public abstract void Drive();
```

```
}

public class Car : Vehicle
{
    public override void Drive()
    {
        Console.WriteLine("Car is driving on the road.");
    }
}

public class Bike : Vehicle
{
    public override void Drive()
    {
        Console.WriteLine("Bike is riding on the road.");
    }
}
```

Main program:

```
    {
        Car car = new Car();
car.Drive();

Bike bike = new Bike();
bike.Drive();
    }
```

Lab 8. Sealed Classes

Create a class that cannot be inherited using the sealed keyword.
Problem:
Create a sealed class MathOperations with a method Add(). Show that it cannot be inherited.

```
public sealed class MathOperations
  {
     public int Add(int a, int b)
     {
        return a + b;
     }
  }
  public class math : MathOperations
  { }
```

Main program:

```
        {
            MathOperations math = new MathOperations();
int result = math.Add(5, 10);
Console.WriteLine("Result: " + result);
        }
```

Lab 9. Constructor Chaining

Demonstrate how constructors are called in a class hierarchy.
Problem:
Create a base class Person with a parameterized constructor. Create a derived class Employee that calls the base class constructor.

```
class Person
    {
        public string Name { get; set; }

        public Person(string name)
        {
            Name = name;
            Console.WriteLine("Person constructor called: " + Name);
        }
    }

    class Student : Person
    {
        public int RollNumber { get; set; }

        public Student(string name, int rollNumber) : base(name)
        {
            RollNumber = rollNumber;
            Console.WriteLine("Student constructor called: " + Name + ", " + RollNumber);
        }
    }
```

Main program:

```
{
    Student student = new Student("Alice", 10);
}
```

Lab 10. Interface Inheritance

Demonstrate that one interface can inherit from another.

Problem:
Create an interface IDriveable with a method Drive(). Create another interface IRaceable that inherits from IDriveable and adds a method Race().

```
interface IDriveable
   {
      void Drive();
   }

   interface IRaceable : IDriveable
   {
      void Race();
   }

   class Car : IRaceable
   {
      public void Drive()
      {
         Console.WriteLine("Car is driving");
      }

      public void Race()
      {
         Console.WriteLine("Car is racing");
      }
   }
```

Main program:

```
      {
         Car car = new Car();
car.Drive();
car.Race();
```

Lab 11. IS-A Relationship (Inheritance)

Problem:
Create a base class Animal with properties like Name and methods like Eat(). Create a derived class
Dog that inherits from Animal and adds its own method Bark(). Show how the IS-A relationship works.

```
class Animal
   {
      public string Name { get; set; }

      public void Eat()
```

```
    {
        Console.WriteLine($"{Name} is eating.");
    }
}

class Dog : Animal
{
    public void Bark()
    {
        Console.WriteLine($"{Name} is barking.");

    }
}
```

Main program:

```
    {
Dog dog = new Dog();
dog.Name = "Buddy";
dog.Eat(); // Inherited from Animal
dog.Bark(); // Specific to Dog
}
```

Lab 12.HAS-A Relationship (Composition)

Problem:
Create a class Engine with properties like HorsePower. Create a class Car that contains an instance
of Engine and shows the HAS-A relationship. Demonstrate how the Car can use its Engine to show
engine-related details.

```
class Engine
{
    public int HorsePower { get; set; }

    public Engine(int horsepower)
    {
        HorsePower = horsepower;
    }
}

class Car
{
    private Engine engine;
```

```
    public Car(Engine engine)
    {
        this.engine = engine;
    }

    public void ShowEngineDetails()
    {
        Console.WriteLine("Car's Engine Horsepower: " + engine.HorsePower);
    }
}
```

Main program:

```
{
    Engine carEngine = new Engine(200);
Car myCar = new Car(carEngine);

myCar.ShowEngineDetails();
    }
```

Lab 13. Calling Base Class Method Using base

Problem:
Create a base class Person with a method DisplayInfo(). Derive a class Employee that overrides DisplayInfo() but still calls the base class's DisplayInfo() using base.

```
class Person
{
    public string Name { get; set; }
    public int Age { get; set; }

    public Person(string name, int age)
    {
        Name = name;
        Age = age;
    }

    public virtual void DisplayInfo()
    {
        Console.WriteLine("Name: " + Name);
        Console.WriteLine("Age: " + Age);
    }
}
```

```
class Employee : Person
{
    public string Designation { get; set; }

    public Employee(string name, int age, string designation) : base(name, age)
    {
        Designation = designation;
    }

    public override void DisplayInfo()
    {
        base.DisplayInfo(); // Call the base class's DisplayInfo()
        Console.WriteLine("Designation: " + Designation);
    }
}
```

Main program:

```
    {
        Employee employee = new Employee("Alice", 30, "Software Engineer");
employee.DisplayInfo();
    }
```

Lab 14. Accessing Base Class Variable Using base

Problem:
Create a base class Person with a property Name. In the derived class Student, hide the Name property using the new keyword and use base to access the base class's Name property.
Lab 14. Accessing Base Class Variable Using base

```
class Person
{
    public string Name { get; set; }
}

class Student : Person
{
    public new string Name { get; set; }


    public void DisplayNames()
    {
        Console.WriteLine("Base Class Name: " + base.Name);
        Console.WriteLine("Derived Class Name: " + Name);
    }
}
```

Main program:

```
      {
         Student student = new Student();
student.Name = "Alice"; // Sets the derived class's Name
Person person = new Person();
person.Name = "bob";
student.DisplayNames();
      }
```

Lab 15. Calling Base Class Constructor Using base

This assignment demonstrates how to use the base keyword to call the base class constructor from the derived class constructor.

Problem:
Create a base class Vehicle with a constructor that accepts brand. Derive a class Car that passes
values to the base class constructor using base.

```
 class Vehicle
    {
       public string Brand { get; set; }

       public Vehicle(string brand)
       {
          Brand = brand;
          Console.WriteLine("Vehicle brand: " + Brand);
       }
    }

    class Car : Vehicle
    {
       public string Model { get; set; }

       public Car(string brand, string model) : base(brand)
       {
          Model = model;
          Console.WriteLine("Car model: " + Model);
       }
    }

}
```

Main program:

```
{
    Car car = new Car("Toyota", "Camry");
}
```