**Questions:**

1. **Create a Simple Class**

Write a C# console application that defines a Person class with properties for Name and Age. Include a method to display these properties.

internal class Person

{

public string name;

public int age;

public Person(string Name, int Age)

{

name = Name;

age = Age;

}

public void display()

{

Console.WriteLine($"{name} {age}");

}

}

internal class Program

{

static void Main(string[] args)

{

Person person = new Person("Ajith", 33);

person.display();

Console.ReadLine();

}

}

**Demonstrate Encapsulation**

1. Modify the Person class to make Age a private field and provide public methods to get and set its value, ensuring the age cannot be set to a negative number.

internal class Person

{

public string name;

private int age;

public int Age

{

get { return age; }

}

public void setAge(int Age)

{

if (Age > 0)

{ this.age = Age; }

else

{

Console.WriteLine("The age must be in poistive number");

}

}

public void display()

{

Console.WriteLine($"{name} {age}");

}

}

internal class Program

{

static void Main(string[] args)

{

Person person = new Person();

person.name = "Ajith";

person.setAge(33);

person.display();

Console.ReadLine();

}

}

**Inheritance Example**

1. Create a Student class that inherits from Person. Add a new property StudentID and override the method to display the properties, including the StudentID.

internal class Person

{

public string name = "Ajith";

public int age = 33;

public virtual void display()

{

Console.WriteLine($"{name} {age}");

}

}

internal class Student : Person

{

int studentId = 101;

public override void display()

{

Console.WriteLine($"{name} {age} {studentId}");

}

}

internal class Program

{

static void Main(string[] args)

{

Person student = new Student();

student.display();

Console.ReadLine();

}

}

**Polymorphism with Method Overloading**

1. Add multiple overloaded constructors to the Student class to initialize objects with different sets of parameters.

internal class Student : Person

{

public int studentId;

public Student(string Name, int Age)

{

name = Name;

age = Age;

}

public Student(string Name, int Age, int StudentId)

{

name = Name;

age = Age;

studentId = StudentId;

}

public override void display()

{

Console.WriteLine($"{name} {age} {studentId}");

}

}

internal class Program

{

static void Main(string[] args)

{

Person student1 = new Student("Ajith", 33);

student1.display();

Person student2 = new Student("Ajith", 33,101);

student2.display();

Console.ReadLine();

}

}

**Polymorphism with Method Overriding**

1. Add a method Study in the Student class and override it in a new class GraduateStudent to provide a different implementation.

internal class Student : Person

{

public int studentId;

public Student() { }

public virtual void study()

{

Console.WriteLine("The student is studying...");

}

}

internal class GraduateStudent : Student

{

public override void study()

{

Console.WriteLine("The Graduate student is also studying...");

}

}

internal class Program

{

static void Main(string[] args)

{

Student graduateStudent = new GraduateStudent();

graduateStudent.study();

Console.ReadLine();

}

}

**Abstract Class and Method**

1. Define an abstract class Shape with an abstract method CalculateArea. Implement this method in derived classes Circle and Rectangle.

internal abstract class Shape

{

public abstract void calculateArea();

}

internal class Circle : Shape

{

public override void calculateArea()

{

Console.WriteLine("The area of Circle");

}

}

internal class Rectangle : Shape

{

public override void calculateArea()

{

Console.WriteLine("The area of Rectangle");

}

}

internal class Program

{

static void Main(string[] args)

{

Circle circle = new Circle();

circle.calculateArea();

Rectangle rectangle = new Rectangle();

rectangle.calculateArea();

Console.ReadLine();

}

}

**Interface Implementation**

1. Create an interface IDriveable with a method Drive. Implement this interface in classes Car and Bike, each providing its own implementation of the Drive method.

internal interface IDriveable

{

void Drive();

}

internal class Car : IDriveable

{

public void Drive()

{

Console.WriteLine("Car");

}

}

internal class Bike : IDriveable

{

public void Drive()

{

Console.WriteLine("Bike");

}

}

internal class Program

{

static void Main(string[] args)

{

Car car = new Car();

car.Drive();

Bike bike = new Bike();

bike.Drive();

Console.ReadLine();

}

}

**Properties with Logic**

1. Add a property BirthYear to the Person class. Modify the Age property to calculate age based on the BirthYear.

internal class Person

{

int birthyear;

public int BirthYear

{

set { birthyear = value; }

get { return birthyear; }

}

public int Age

{

get

{

int thisYear = DateTime.Now.Year;

return thisYear - birthyear;

}

}

}

internal class Program

{

static void Main(string[] args)

{

Person person = new Person();

person.BirthYear = 1990;

Console.WriteLine(person.Age);

Console.ReadLine();

}

}

**Association between Classes**

1. Create a Course class and associate it with the Student class, where each student can enroll in multiple courses.

internal class Course

{

public int courseId;

public string courseName;

public string courseInstructor;

}

internal class Student

{

public List<Course> Courses = new List<Course>()

{

new Course {courseId = 100, courseName = "Java", courseInstructor = "Ajith"},

new Course {courseId = 101, courseName = "Python", courseInstructor = "Alice"},

new Course {courseId = 103, courseName = "C#", courseInstructor = "Bob"}

};

}

**Composition**

1. Create an Address class and use it as a property in the Person class to store address details.

internal class Address

{

public int doorNo;

public string streetName;

public string city;

public string country;

}

internal class Person

{

public Address person1 = new Address()

{

doorNo = 5,

streetName = "New road",

city = "Jaffna",

country = "Srilanka"

};

}

static void Main(string[] args)

{

Person person = new Person();

Console.WriteLine(person.person1.doorNo);

Console.ReadLine();

}

**Aggregation**

1. Create a Department class and a Teacher class where a department can have multiple teachers.

internal class Teacher

{

public int teacherId;

public string teacherName;

}

internal class Department

{

public List<Teacher> Teachers= new List<Teacher>()

{

new Teacher { teacherId = 101, teacherName = "Ajith"},

new Teacher { teacherId = 102, teacherName = "John"}

};

}