1. Single Responsibility Principle (SRP)

```
using System;

// Class for managing user data
public class User {
    public string Name;

    // Save user data
    public void Save() {
        Console.WriteLine("User saved.");
    }
}

// Class for logging
public class Log {
    // Log user actions
    public void Write() {
        Console.WriteLine("Action logged.");
    }
}

class Program {
    static void Main() {
        // **SRP Example**: Separate user data management and logging
        User u = new User { Name = "John" };
        Log l = new Log();

        u.Save();  // Save user data
        l.Write(); // Log action
    }
}
```

---

2. Open/Closed Principle (OCP)

```
using System;

// Shape interface for calculating area
public interface IShape {
    double Area();
}

public class Circle : IShape {
    public double R;
```

```
    public double Area() {
        return Math.PI * R * R;  // Circle area
    }
}

public class Rect : IShape {
    public double W, H;

    public double Area() {
        return W * H;  // Rectangle area
    }
}

class Program {
    static void Main() {
        // **OCP Example**: Add new shapes without modifying AreaCalculator
        IShape c = new Circle { R = 5 };
        IShape r = new Rect { W = 4, H = 6 };

        Console.WriteLine("Circle area: " + c.Area());
        Console.WriteLine("Rect area: " + r.Area());
    }
}
```

---

3. Liskov Substitution Principle (LSP)

```
using System;

// Base class Bird
public class Bird {
    public virtual void Move() {
        Console.WriteLine("Flying");
    }
}

// Derived class Sparrow
public class Spar : Bird {
    public override void Move() {
        Console.WriteLine("Sparrow flying");
    }
}

// Derived class Ostrich
public class Ostr : Bird {
```

```
    public override void Move() {
        Console.WriteLine("Ostrich running");
    }
}

class Program {
    static void Main() {
        // **LSP Example**: Substitution of Bird with derived classes
        Bird s = new Spar();
        Bird o = new Ostr();

        s.Move();  // Sparrow
        o.Move();  // Ostrich
    }
}
```

---

4. Interface Segregation Principle (ISP)

```
using System;

// Worker interface for working
public interface IWork {
    void DoWork();
}

// Rest interface for resting
public interface IRest {
    void Rest();
}

public class Wkr : IWork, IRest {
    public void DoWork() {
        Console.WriteLine("Working");
    }
    public void Rest() {
        Console.WriteLine("Resting");
    }
}

public class Bot : IWork {
    public void DoWork() {
        Console.WriteLine("Robot working");
    }
}
```

```
class Program {
    static void Main() {
        // **ISP Example**: Robot only implements IWork, not IRest
        IWork w = new Wkr();
        w.DoWork();

        IRest r = new Wkr();
        r.Rest();

        IWork b = new Bot();
        b.DoWork();  // Robot doesn't need Rest
    }
}
```

---

5. Dependency Inversion Principle (DIP)

```
using System;

// Interface for devices
public interface IDevice {
    void On();
    void Off();
}

// LightBulb class implementing IDevice
public class Bulb : IDevice {
    public void On() {
        Console.WriteLine("Bulb ON");
    }
    public void Off() {
        Console.WriteLine("Bulb OFF");
    }
}

// Switch class depends on IDevice abstraction
public class Swtch {
    private readonly IDevice _d;

    public Swtch(IDevice d) {
        _d = d;
    }

    public void Operate() {
```

```
        _d.On();
    }
}

class Program {
    static void Main() {
        // **DIP Example**: Switch depends on abstraction, not on concrete class
        IDevice b = new Bulb();
        Swtch s = new Swtch(b);
        s.Operate();  // Turns on the Bulb
    }
}
```

Explanation:

1. **Single Responsibility Principle (SRP)**:
   - The `User` class manages user data, and the `Log` class handles logging. Each class has a single responsibility.
2. **Open/Closed Principle (OCP)**:
   - The `Area()` method in `IShape` interface allows adding new shapes (like `Circle`, `Rect`) without modifying existing classes.
3. **Liskov Substitution Principle (LSP)**:
   - Derived classes `Spar` and `Ostr` can substitute their base class `Bird` without breaking the behavior.
4. **Interface Segregation Principle (ISP)**:
   - `IWork` and `IRest` are segregated, so classes only implement the methods they actually need.
5. **Dependency Inversion Principle (DIP)**:
   - `Switch` depends on the `IDevice` abstraction, which allows for easy replacement of devices (e.g., `Bulb`).

How to Run:

1. Copy each code block into separate `.cs` files (e.g., `SRPExample.cs`, `OCPExample.cs`, etc.).
2. Compile and run each example with `csc <filename.cs>` or through Visual Studio.