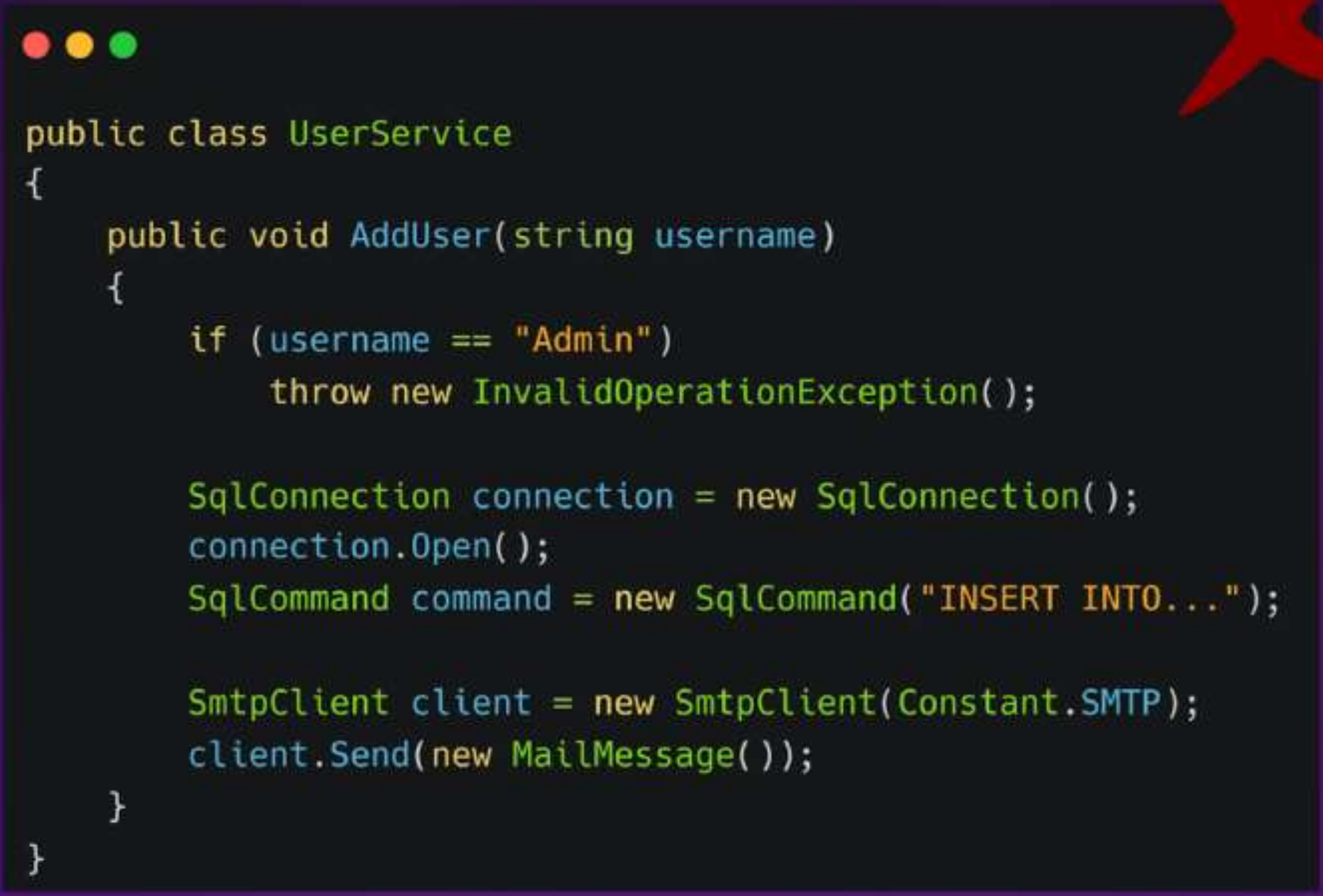
SOLID Principles

Simplifed

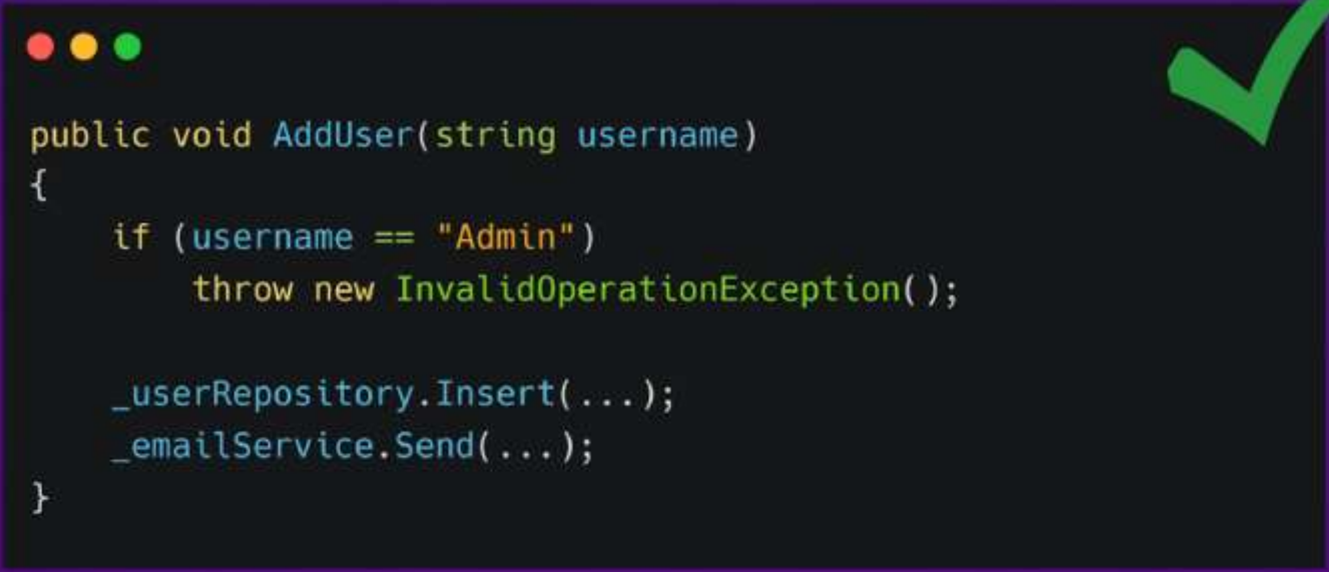
Examples

swipe >>

(S)ingle Responsibility

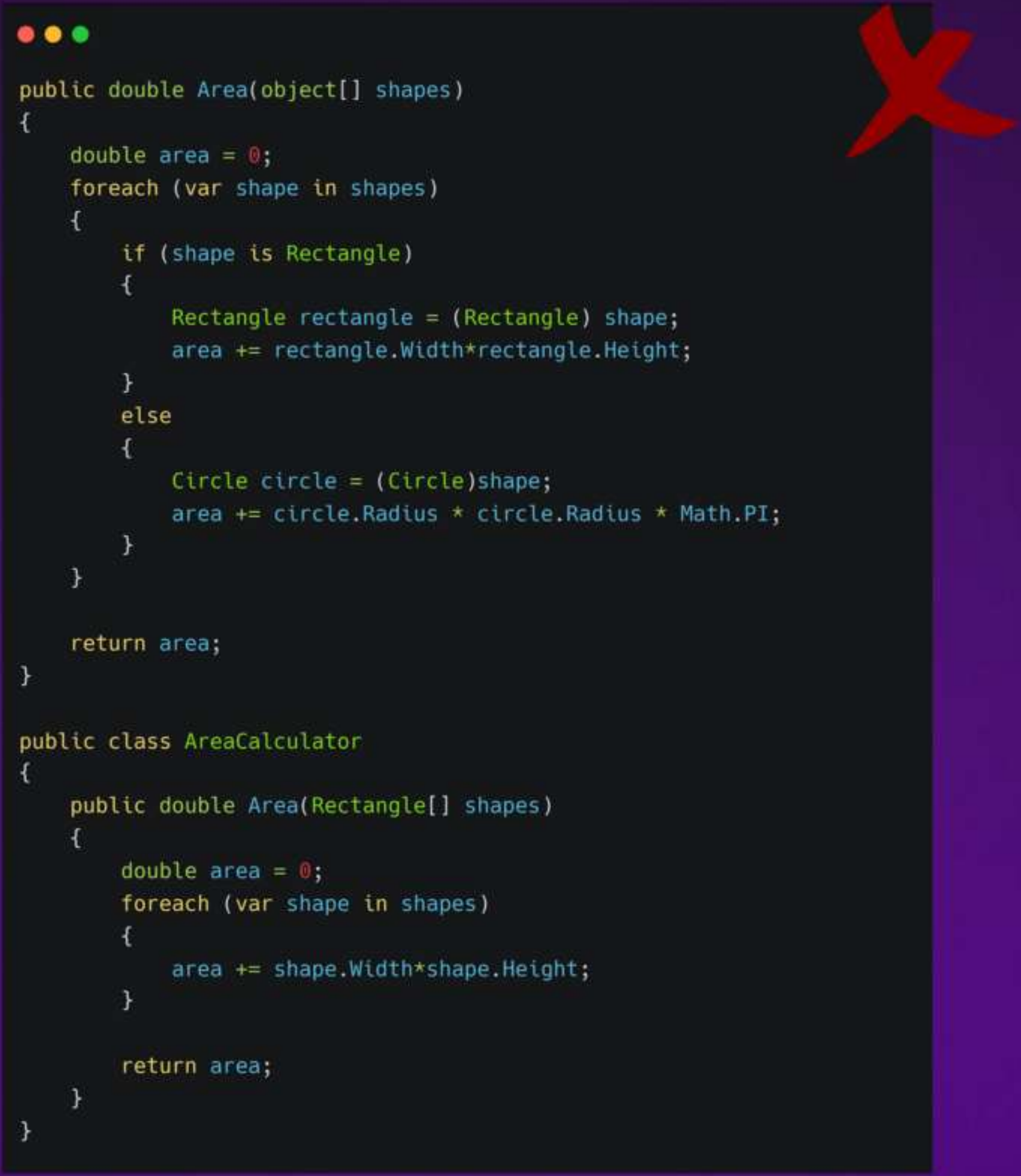


```
public class UserService
{
    public void AddUser(string username)
    {
        if (username == "Admin")
            throw new InvalidOperationException();

        SqlConnection connection = new SqlConnection();
        connection.Open();
        SqlCommand command = new SqlCommand("INSERT INTO...");

        SmtpClient client = new SmtpClient(Constant.SMTP);
        client.Send(new MailMessage());
    }
}
```

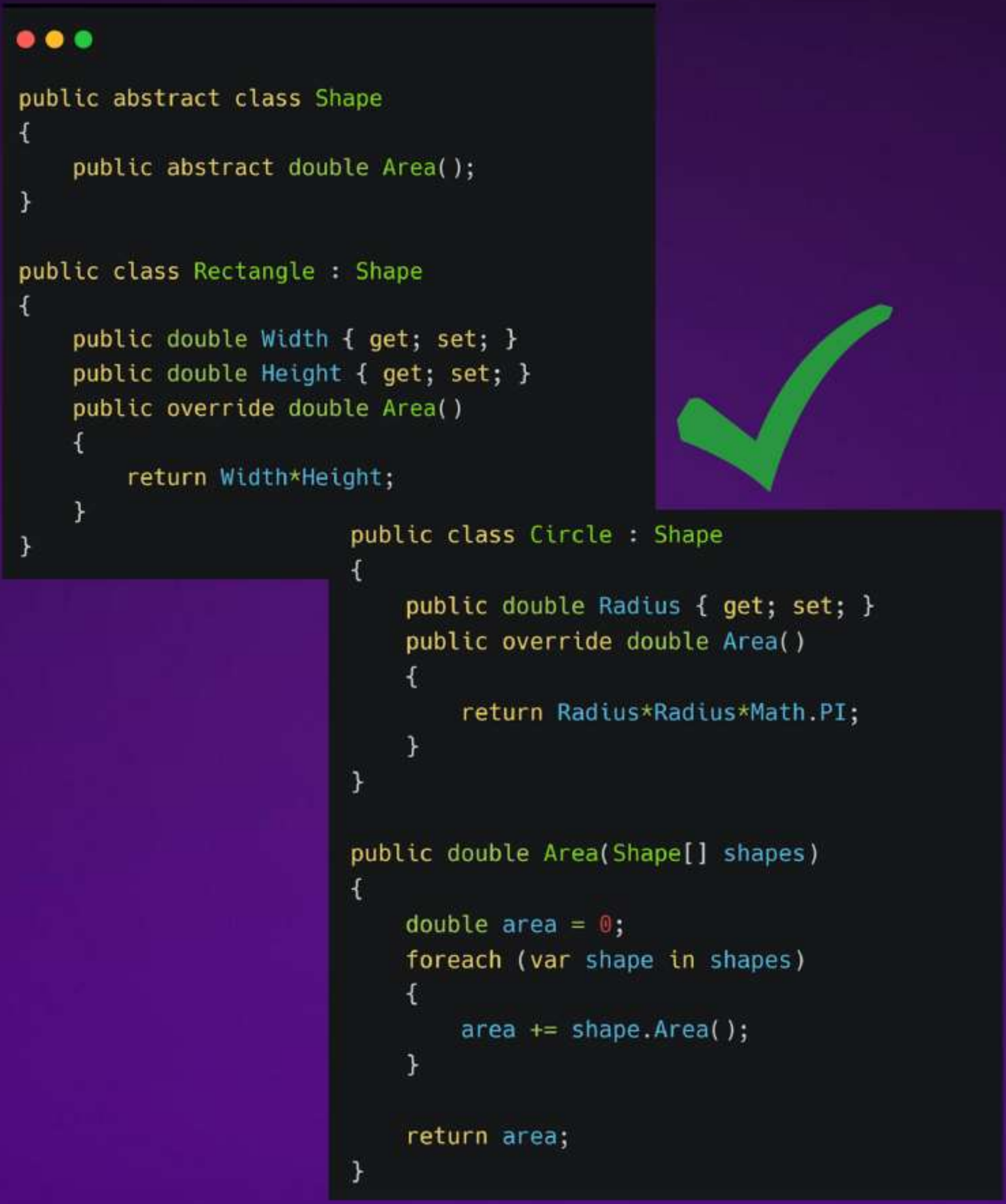


```
public void AddUser(string username)
{
    if (username == "Admin")
        throw new InvalidOperationException();

    _userRepository.Insert(...);
    _emailService.Send(...);
}
```

(O)pen-closed principle



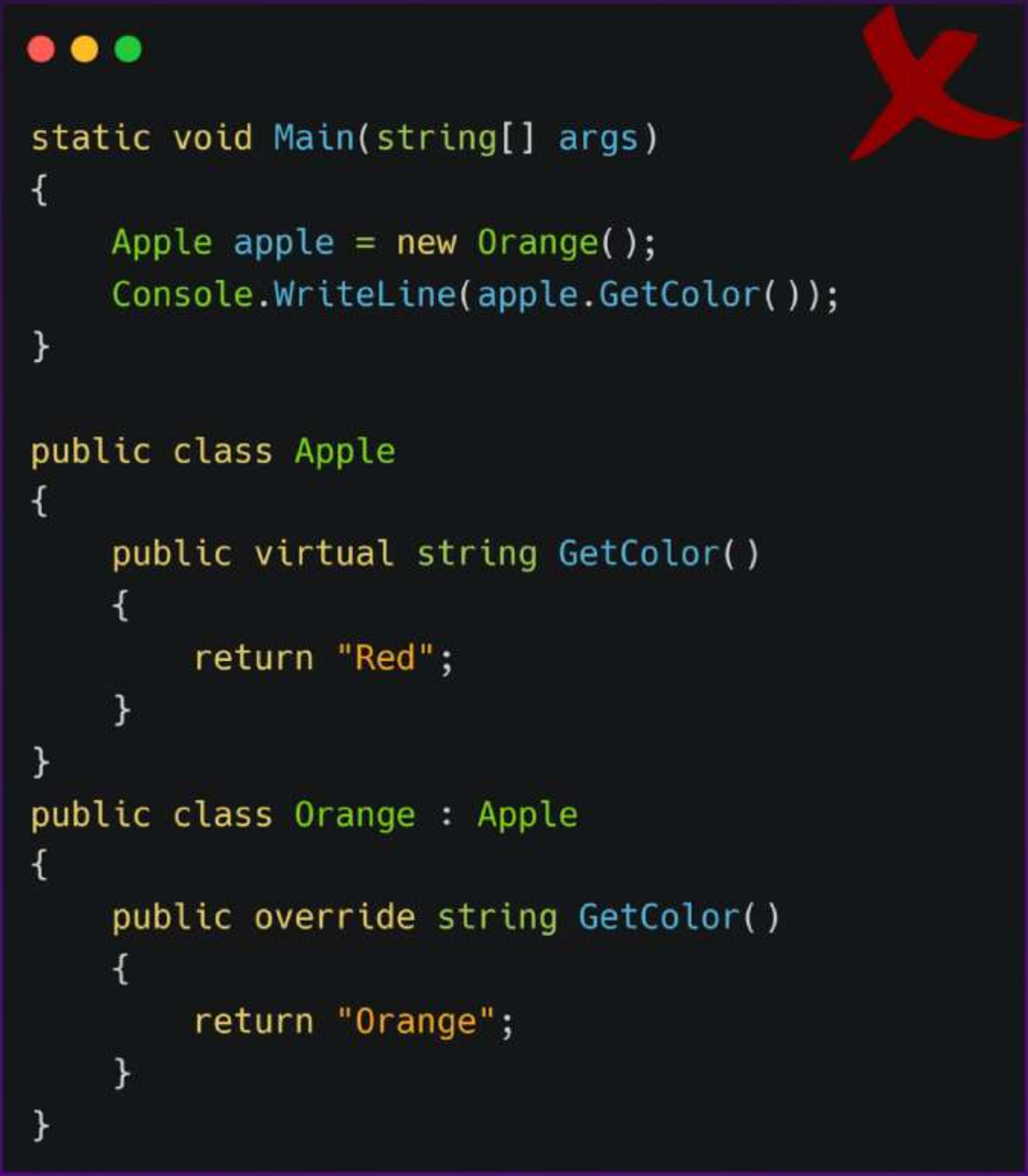
```
public double Area(object[] shapes)
{
    double area = 0;
    foreach (var shape in shapes)
    {
        if (shape is Rectangle)
        {
            Rectangle rectangle = (Rectangle) shape;
            area += rectangle.Width*rectangle.Height;
        }
        else
        {
            Circle circle = (Circle)shape;
            area += circle.Radius * circle.Radius * Math.PI;
        }
    }

    return area;
}

public class AreaCalculator
{
    public double Area(Rectangle[] shapes)
    {
        double area = 0;
        foreach (var shape in shapes)
        {
            area += shape.Width*shape.Height;
        }

        return area;
    }
}
```



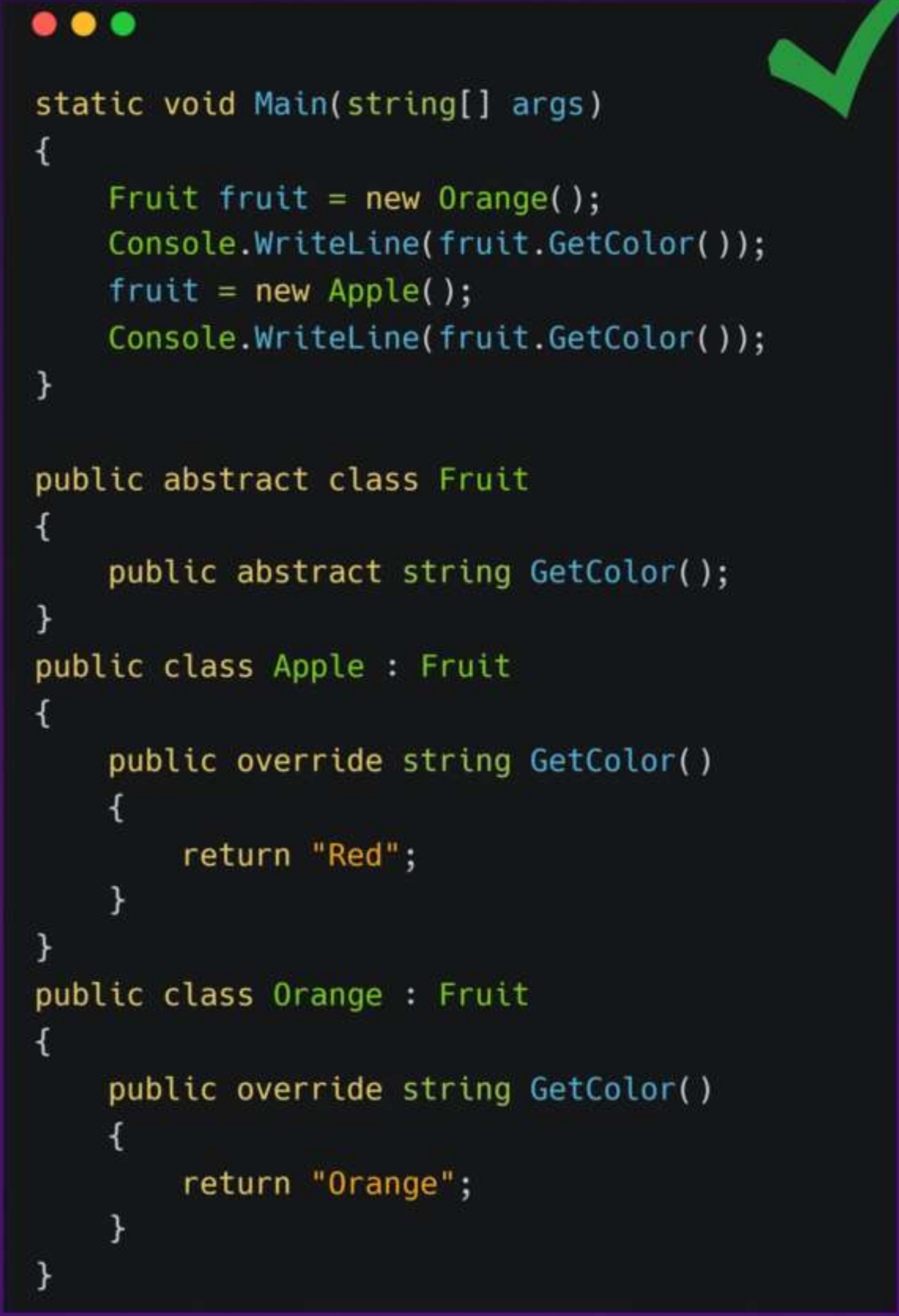
```
public abstract class Shape
{
    public abstract double Area();
}

public class Rectangle : Shape
{
    public double Width { get; set; }
    public double Height { get; set; }
    public override double Area()
    {
        return Width*Height;
    }
}
```

```
public class Circle : Shape
{
    public double Radius { get; set; }
    public override double Area()
    {
        return Radius*Radius*Math.PI;
    }
}

public double Area(Shape[] shapes)
{
    double area = 0;
    foreach (var shape in shapes)
    {
        area += shape.Area();
    }

    return area;
}
```

(L)iskov substitution principle

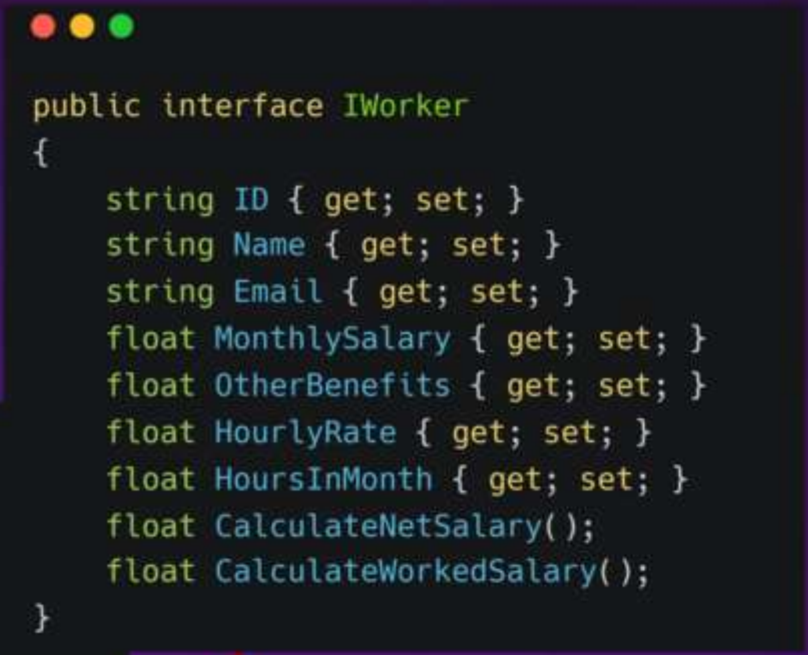


```
static void Main(string[] args)
{
    Apple apple = new Orange();
    Console.WriteLine(apple.GetColor());
}

public class Apple
{
    public virtual string GetColor()
    {
        return "Red";
    }
}
public class Orange : Apple
{
    public override string GetColor()
    {
        return "Orange";
    }
}
```

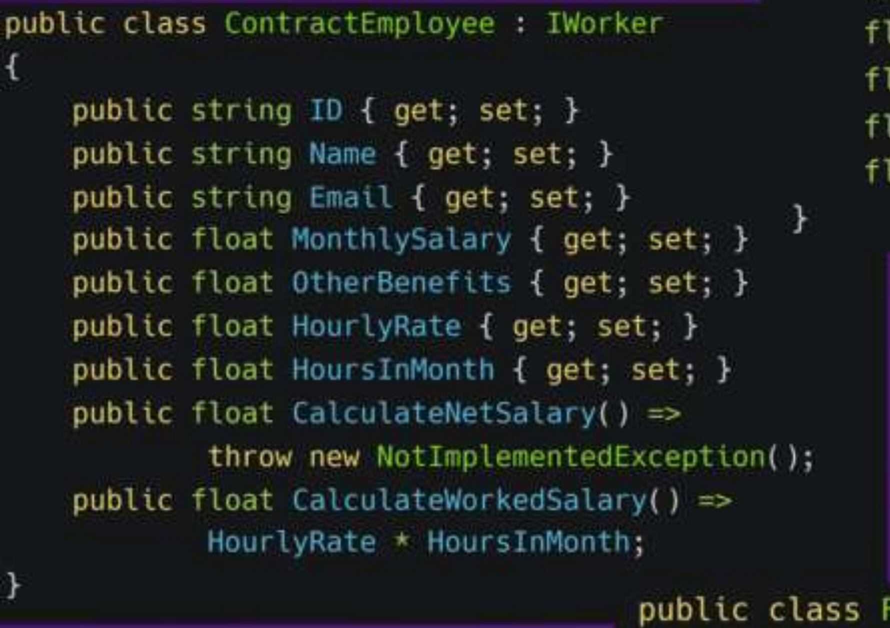


```
static void Main(string[] args)
{
    Fruit fruit = new Orange();
    Console.WriteLine(fruit.GetColor());
    fruit = new Apple();
    Console.WriteLine(fruit.GetColor());
}

public abstract class Fruit
{
    public abstract string GetColor();
}
public class Apple : Fruit
{
    public override string GetColor()
    {
        return "Red";
    }
}
public class Orange : Fruit
{
    public override string GetColor()
    {
        return "Orange";
    }
}
```

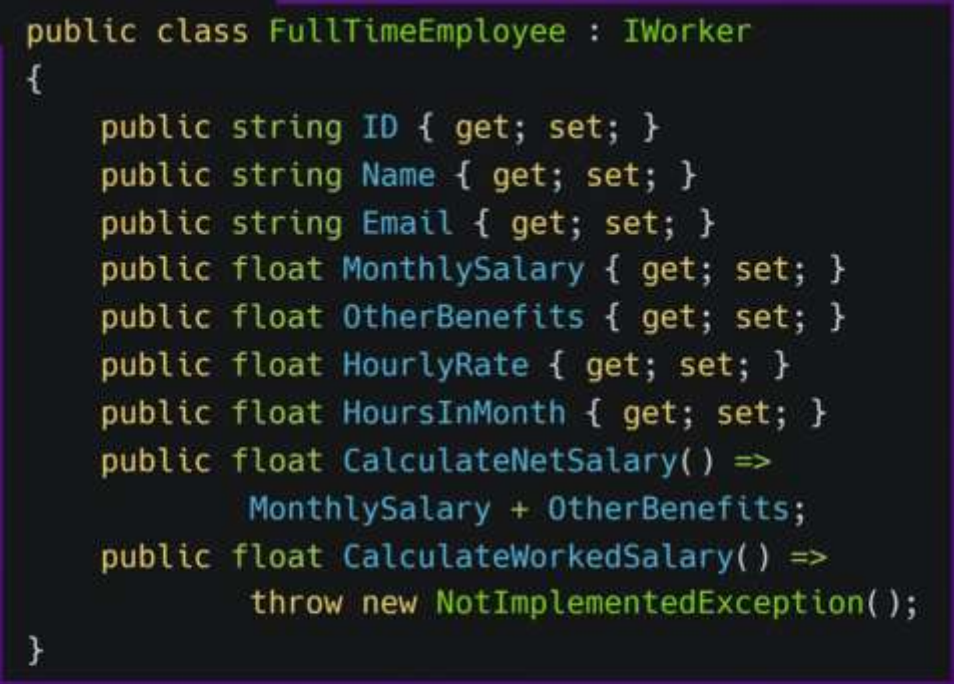
(I)nterface segregation principle



```
public interface IWorker
{
    string ID { get; set; }
    string Name { get; set; }
    string Email { get; set; }
    float MonthlySalary { get; set; }
    float OtherBenefits { get; set; }
    float HourlyRate { get; set; }
    float HoursInMonth { get; set; }
    float CalculateNetSalary();
    float CalculateWorkedSalary();
}
```



```
public class ContractEmployee : IWorker
{
    public string ID { get; set; }
    public string Name { get; set; }
    public string Email { get; set; }
    public float MonthlySalary { get; set; }
    public float OtherBenefits { get; set; }
    public float HourlyRate { get; set; }
    public float HoursInMonth { get; set; }
    public float CalculateNetSalary() =>
            throw new NotImplementedException();
    public float CalculateWorkedSalary() =>
            HourlyRate * HoursInMonth;
}
```



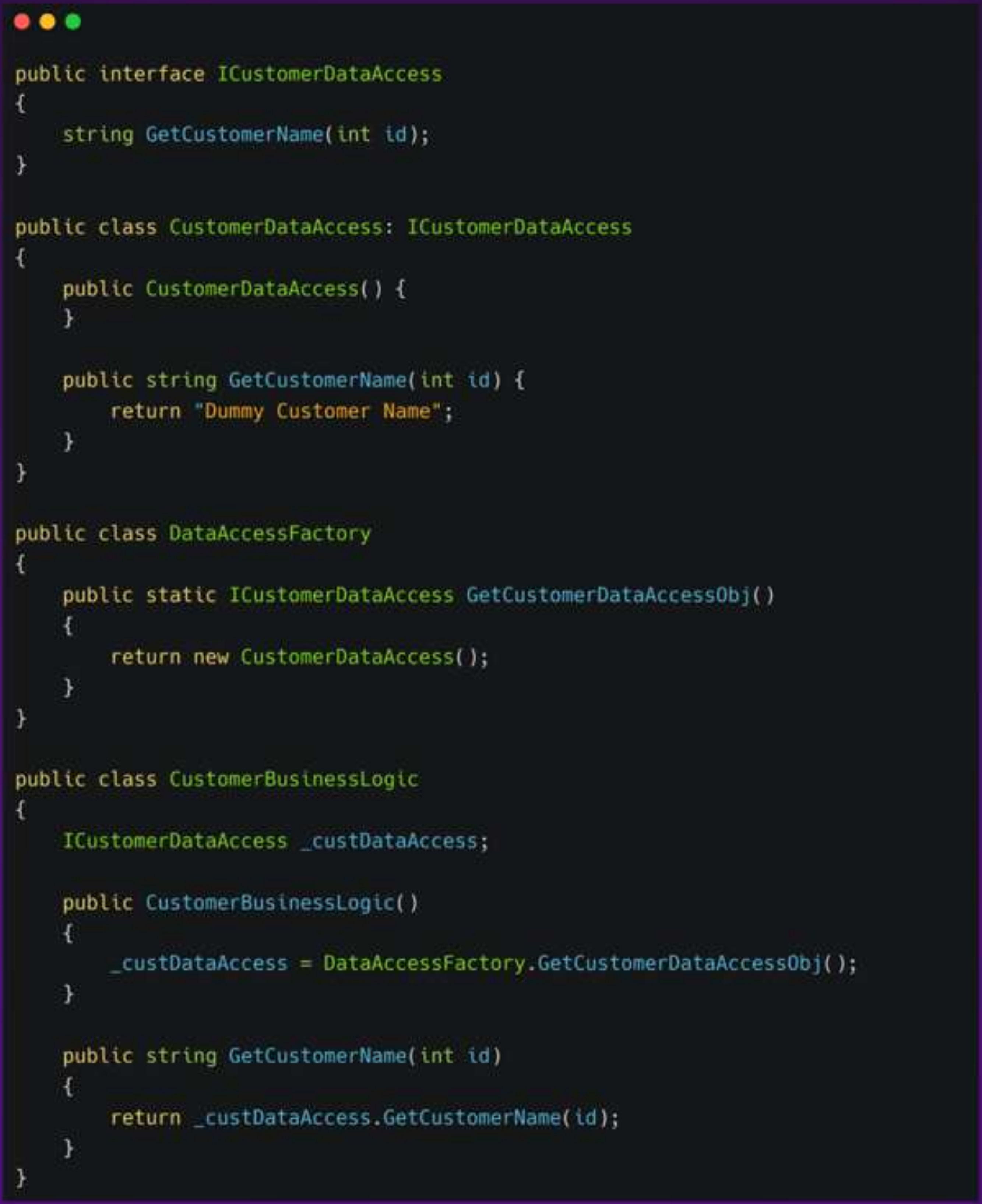
```
public class FullTimeEmployee : IWorker
{
    public string ID { get; set; }
    public string Name { get; set; }
    public string Email { get; set; }
    public float MonthlySalary { get; set; }
    public float OtherBenefits { get; set; }
    public float HourlyRate { get; set; }
    public float HoursInMonth { get; set; }
    public float CalculateNetSalary() =>
            MonthlySalary + OtherBenefits;
    public float CalculateWorkedSalary() =>
            throw new NotImplementedException();
}
```

```
public interface IBaseWorker
{
    string ID { get; set; }
    string Name { get; set; }
    string Email { get; set; }
}
```

```
public interface IFullTimeWorkerSalary : IBaseWorker
{
    float MonthlySalary { get; set; }
    float OtherBenefits { get; set; }
    float CalculateNetSalary();
}

public interface IContractWorkerSalary : IBaseWorker
{
    float HourlyRate { get; set; }
    float HoursInMonth { get; set; }
    float CalculateWorkedSalary();
}
```

```
public class FullTimeEmployeeFixed : IFullTimeWorkerSalary
{
    public string ID { get; set; }
    public string Name { get; set; }
    public string Email { get; set; }
    public float MonthlySalary { get; set; }
    public float OtherBenefits { get; set; }
    public float CalculateNetSalary() => MonthlySalary + OtherBenefits;
}

public class ContractEmployeeFixed : IContractWorkerSalary
{
    public string ID { get; set; }
    public string Name { get; set; }
    public string Email { get; set; }
    public float HourlyRate { get; set; }
    public float HoursInMonth { get; set; }
    public float CalculateWorkedSalary() => HourlyRate * HoursInMonth;
}
```

(D)ependency inversion principle



```
public interface ICustomerDataAccess
{
    string GetCustomerName(int id);
}

public class CustomerDataAccess: ICustomerDataAccess
{
    public CustomerDataAccess() {
    }

    public string GetCustomerName(int id) {
        return "Dummy Customer Name";
    }
}

public class DataAccessFactory
{
    public static ICustomerDataAccess GetCustomerDataAccessObj()
    {
        return new CustomerDataAccess();
    }
}

public class CustomerBusinessLogic
{
    ICustomerDataAccess _custDataAccess;

    public CustomerBusinessLogic()
    {
        _custDataAccess = DataAccessFactory.GetCustomerDataAccessObj();
    }

    public string GetCustomerName(int id)
    {
        return _custDataAccess.GetCustomerName(id);
    }
}
```