SOLID

Principles for Robust and Maintainable Code

S *Single Responsibility Principle*

O *Open-Closed Principle*

L *Liskov Substitution Principle*

I *Interface Segregation Principle*

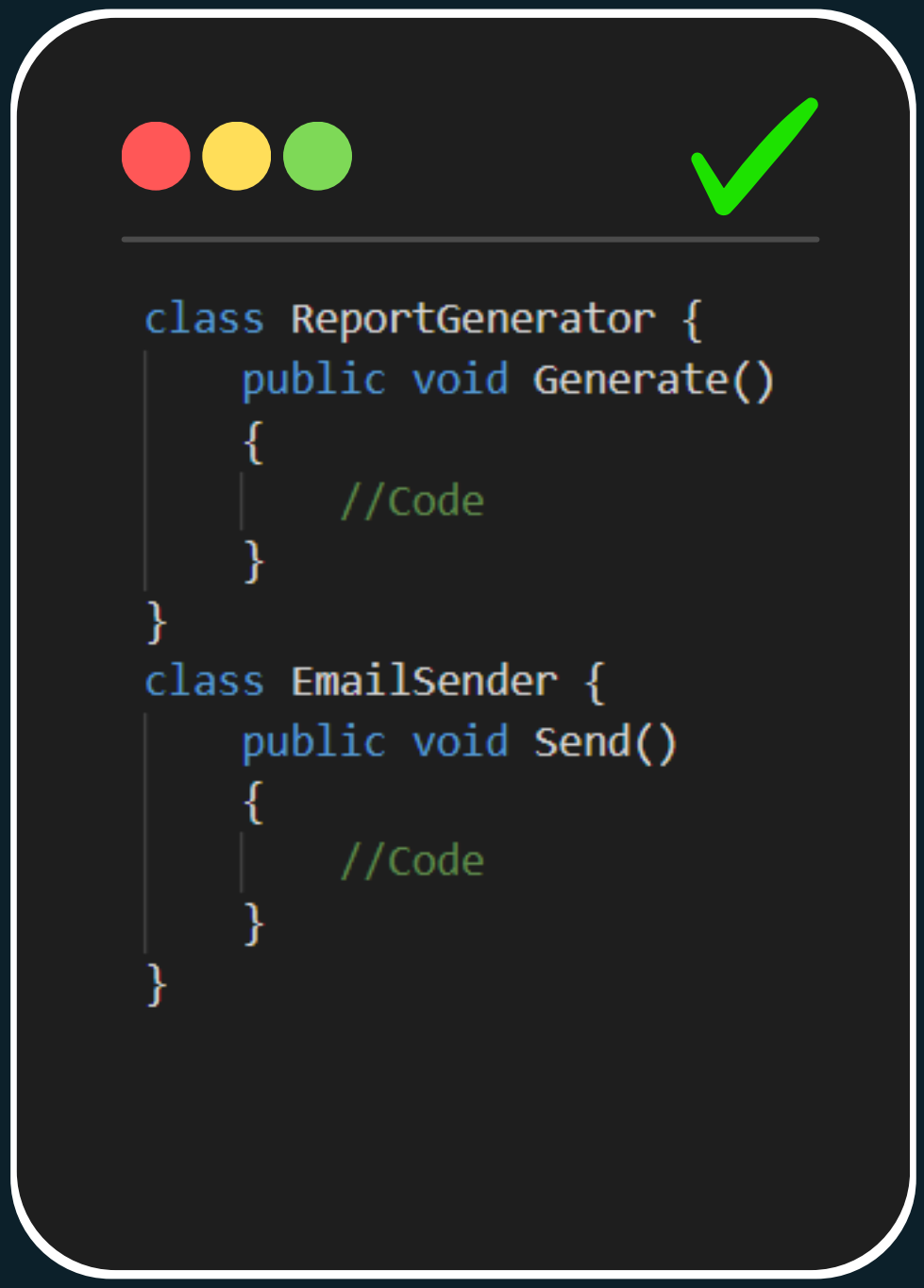
D *Dependency Inversion Principle*

Jihene Labidi

Single Responsibility Principle

A class should have only one responsibility.

```
class Report {
    public void GenerateReport()
    {
        //Code
    }
    public void SendEmail()
    {
        //Code
    }
}
```



```
class ReportGenerator {
    public void Generate()
    {
        //Code
    }
}
class EmailSender {
    public void Send()
    {
        //Code
    }
}
```

Jihene Labidi

Open-Closed Principle

A class should be open for extension but closed for modification.

```
class PaymentProcessor {
    public void ProcessPayment(string paymentType)
    {
        if (paymentType == "CreditCard")
        {
            // Credit card logic
        }
        else if (paymentType == "PayPal")
        {
            // PayPal logic
        }
    }
}
```

```
interface IPaymentMethod
{
    void ProcessPayment(decimal amount);
}

class CreditCardPayment : IPaymentMethod
{
    public void ProcessPayment(decimal amount)
    {
        // Credit card logic
    }
}

class PayPalPayment : IPaymentMethod
{
    public void ProcessPayment(decimal amount)
    {
        // PayPal logic
    }
}
```

Jihene Labidi

Liskov Substitution Principle

Derived classes must be substitutable for their base classes.


```
class Rectangle {
    public virtual double GetArea()
    {
        return Width * Height;
    }
    public double Width { get; set; }
    public double Height { get; set; }
}

class Square : Rectangle {
    public override double GetArea()
    { return Width * Width; } // Violates LSP
}
```


```
interface IShape {
    double GetArea();
}

class Rectangle : IShape {
    public double Width { get; set; }
    public double Height { get; set; }
    public double GetArea() => Width * Height;
}

class Square : IShape {
    public double Side { get; set; }
    public double GetArea() => Side * Side;
}
```

Jihene Labidi

Interface Segregation Principle

Clients should not be forced to depend on interfaces they don't use.


```
interface IUser
{
    void Register(string username, string password);
    void Login(string username, string password);
    void SendResetPasswordEmail(string email);
}
```


```
interface IRegistration
{
    void Register(string username, string password);
}

interface IAuthentication
{
    void Login(string username, string password);
}

interface IPasswordReset
{
    void SendResetPasswordEmail(string email);
}
```

Jihene Labidi

Dependency Inversion Principle

High-level modules should depend on abstractions, not on concretions.

```
class ProductService
{
    private readonly ProductRepository repository = new
    ProductRepository();

    public List<Product> GetProducts()
    {
        return repository.GetAllProducts();
    }
}
```

```
interface IProductRepository
{
    List<Product> GetAllProducts();
}

class ProductRepository : IProductRepository
{
    public List<Product> GetAllProducts()
    {
        // Code
    }
}

class ProductService
{
    private readonly IProductRepository repository;
    public ProductService(IProductRepository repository)
    {
        this.repository = repository;
    }

    public List<Product> GetProducts()
    {
        return repository.GetAllProducts();
    }
}
```

Jihene Labidi