ҚАЗАҚСТАН РЕСПУБЛИКАСЫНЫҢ БІЛІМ ЖӘНЕ ҒЫЛЫМ МИНИСТРЛІГІ

СӘТБАЕВ УНИВЕРСИТЕТІ

ИАиИТ институты

ПИ кафедрасы



|  |
| --- |
|  |

ЗЕРТХАНАЛЫҚ ЖҰМЫС

Тақырыбы

Шаблоны проектирования приложений

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Орындалған жұмыстың сапасы** | **Баға диапазоны** | **Алған %** |
| 1 | Орындалмаған  Себепсіз сабақтан қалуы | 0% |  |
| 2 | Орындалуы және білім алушының белсенділігі | 0-50% |  |
| 3 | Жұмыстың рәсімделуі | 0-20% |  |
| 4 | Анықтамаларды, техникалық әдебиеттерді, дәріс конспектілерін, пәннің оқу-әдістемелік кешенін пайдалана білуі | 0-5% |  |
| 5 | Техникалық құралдарды пайдалана білуі | 0-5% |  |
| 6 | Жұмысты қорғауы | 0-20% |  |
|  | Қорытынды | 0-100% |  |

Білім алушының аты-жөні

Қабдрахманова Айжан

Мамандық

Computer scinse

Оқытушының аты-жөні

Иманбекова У.Н.

Алматы 2024 ж

**Тема:** Модуль 03 Базовые принципы проектирования

*Баллы:*

Произведите корректную (правильную) по вашему мнению реализацию с применением принципа DRY:

1. **Использование методов для устранения дублирования кода**

public class OrderService

{

public void CreateOrder(string productName, int quantity, double price)

{

double totalPrice = quantity \* price;

Console.WriteLine($"Order for {productName} created. Total: {totalPrice}");

}

public void UpdateOrder(string productName, int quantity, double price)

{

double totalPrice = quantity \* price;

Console.WriteLine($"Order for {productName} updated. New total: {totalPrice}");

}

}

**Исправленный**

public class OrderService

{

public void CreateOrder(string productName, int quantity, double price)

{

ProcessOrder("created", productName, quantity, price);

}

public void UpdateOrder(string productName, int quantity, double price)

{

ProcessOrder("updated", productName, quantity, price);

}

private void ProcessOrder(string action, string productName, int quantity, double price)

{

double totalPrice = quantity \* price;

Console.WriteLine($"Order for {productName} {action}. Total: {totalPrice}");

}

}

1. **Метод ProcessOrder:** Вынесена общая логика в отдельный метод ProcessOrder, который принимает действие (создан или обновлен) и остальные параметры.
2. **Параметр action:** Этот параметр позволяет динамически формировать сообщение в зависимости от того, создается заказ или обновляется.

Таким образом, я устранили дублирование кода и сделала его более читаемым и поддерживаемым.

1. **Использование общих базовых классов**

public class Car

{

public void Start()

{

Console.WriteLine("Car is starting");

}

public void Stop()

{

Console.WriteLine("Car is stopping");

}

}

public class Truck

{

public void Start()

{

Console.WriteLine("Truck is starting");

}

public void Stop()

{

Console.WriteLine("Truck is stopping");

}

}

**Исправленный**

public class Vehicle

{

public void Start()

{

Console.WriteLine($"{GetType().Name} is starting");

}

public void Stop()

{

Console.WriteLine($"{GetType().Name} is stopping");

}

}

public class Car : Vehicle

{

// Можно добавить специфичные для Car методы или свойства

}

public class Truck : Vehicle

{

// Можно добавить специфичные для Truck методы или свойства

}

Произведите корректную (правильную) по вашему мнению реализацию с применением принципа KISS:

1. **Избегание чрезмерного использования абстракций**

public interface IOperation

{

void Execute();

}

public class AdditionOperation : IOperation

{

private int \_a;

private int \_b;

public AdditionOperation(int a, int b)

{

\_a = a;

\_b = b;

}

public void Execute()

{

Console.WriteLine(\_a + \_b);

}

}

public class Calculator

{

public void PerformOperation(IOperation operation)

{

operation.Execute();

}

}

**Исправленный**

public class Calculator

{

public void Add(int a, int b)

{

Console.WriteLine(a + b);

}

public void Subtract(int a, int b)

{

Console.WriteLine(a - b);

}

}

// Пример использования

class Program

{

static void Main()

{

var calculator = new Calculator();

calculator.Add(5, 3);

calculator.Subtract(5, 3);

}

}

1. **Избегание избыточного использования шаблонов проектирования**

public class Singleton

{

private static Singleton \_instance;

private Singleton() { }

public static Singleton Instance

{

get

{

if (\_instance == null)

{

\_instance = new Singleton();

}

return \_instance;

}

}

public void DoSomething()

{

Console.WriteLine("Doing something...");

}

}

public class Client

{

public void Execute()

{

Singleton.Instance.DoSomething();

}

}

**Исправленный**

public class SimpleClass

{

public void DoSomething()

{

Console.WriteLine("Doing something...");

}

}

public class Client

{

private readonly SimpleClass \_simpleClass = new SimpleClass();

public void Execute()

{

\_simpleClass.DoSomething();

}

}

Произведите корректную (правильную) по вашему мнению реализацию с применением принципа YAGNI:

1. **Избыточное создание абстракций**

public interface IShape

{

double CalculateArea();

}

public class Circle : IShape

{

private double \_radius;

public Circle(double radius)

{

\_radius = radius;

}

public double CalculateArea()

{

return Math.PI \* \_radius \* \_radius;

}

}

public class Square : IShape

{

private double \_side;

public Square(double side)

{

\_side = side;

}

public double CalculateArea()

{

return \_side \* \_side;

}

}

public class Circle

{

private double \_radius;

public Circle(double radius)

{

\_radius = radius;

}

public double CalculateArea()

{

return Math.PI \* \_radius \* \_radius;

}

}

**Исправленный**

public class Square

{

private double \_side;

public Square(double side)

{

\_side = side;

}

public double CalculateArea()

{

return \_side \* \_side;

}

}

// Пример использования

class Program

{

static void Main()

{

var circle = new Circle(5);

Console.WriteLine($"Circle area: {circle.CalculateArea()}");

var square = new Square(4);

Console.WriteLine($"Square area: {square.CalculateArea()}");

}

}

**Объяснение изменений**

1. **Удаление интерфейса:** Убрали интерфейс IShape, так как в текущем контексте он не нужен. Каждая фигура может иметь свои методы, которые достаточно интуитивны и не требуют общей абстракции.
2. **Упрощение классов:** Классы Circle и Square теперь содержат только нужную функциональность. Это делает код легче для понимания и сопровождения.
3. **Конкретизация:** Если в будущем возникнет необходимость обрабатывать множество фигур одновременно, всегда можно вернуть интерфейс, но пока это не требуется.
4. **Излишняя параметризация методов**

public class MathOperations

{

public int Add(int a, int b, bool shouldLog = false)

{

int result = a + b;

if (shouldLog)

{

Console.WriteLine($"Result: {result}");

}

return result;

}

}

**Исправленный**

public class MathOperations

{

public int Add(int a, int b)

{

return a + b;

}

public void AddAndLog(int a, int b)

{

int result = Add(a, b);

Console.WriteLine($"Result: {result}");

}

}

// Пример использования

class Program

{

static void Main()

{

var mathOperations = new MathOperations();

int sum = mathOperations.Add(3, 5);

mathOperations.AddAndLog(3, 5);

}

}