Тема: Принципы SOLID. TDD

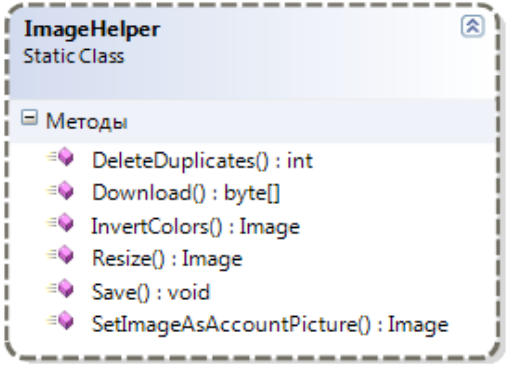
S.O.L.I.D.

- The **S**ingle Responsibility Principle (Принцип единственности ответственности)
- The **O**pen Closed Principle (Принцип открытости/закрытости)
- The **L**iskov Substitution Principle (Принцип замещения Лисков)
- The **I**nterface Segregation Principle (Принцип разделения интерфейса)
- The **D**ependency Inversion Principle (Принцип инверсии зависимости)

Принцип единственности ответственности

- **Не должно быть больше одной причины для изменения класса.**

Пример



DeleteDublicates - удалить из файловой системы все дублирующиеся изображения и вернуть количество удаленных

Download - загрузка битового массива с изображением с помощью HTTP запроса

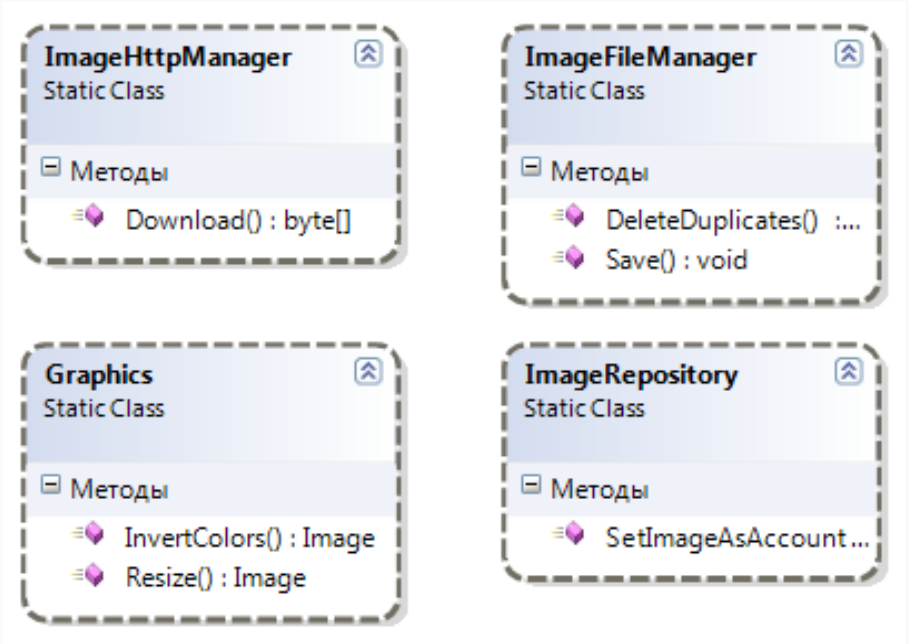
InvertColours - изменить цвета на изображении

Resize - изменить размер изображения

Save - Сохранить изображение в файловой системе

SetImageAsAccountPicture - Запрос к базе данных для сохранения ссылки на это изображение для пользователя

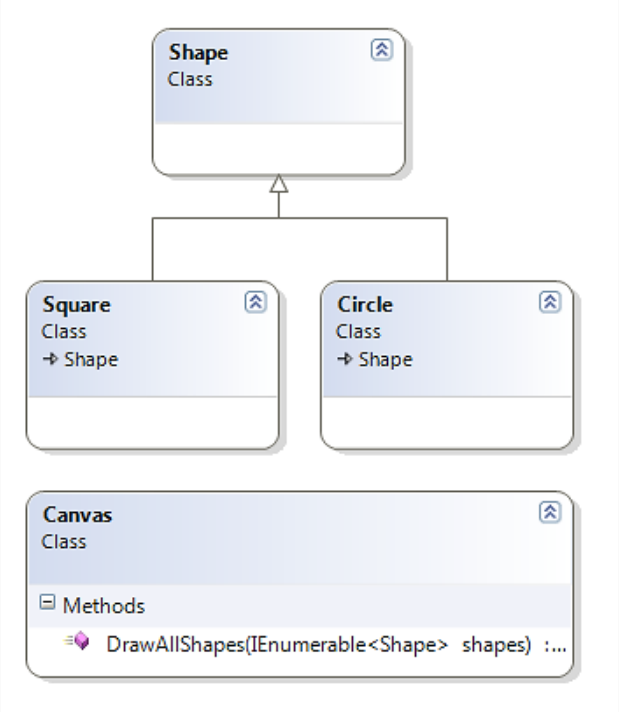
Решение


ImageHttpManager
Static Class
Методы
Download() : byte[]
ImageFileManager
Static Class
Методы
DeleteDuplicates() :...
Save() : void
Graphics
Static Class
Методы
InvertColors() : Image
Resize() : Image
ImageRepository
Static Class
Методы
SetImageAsAccount...

Принцип открытости/закрытости

- **Программные сущности (классы, модули, функции и т.д.) должны быть открыты для расширения, но закрыты для изменения.**

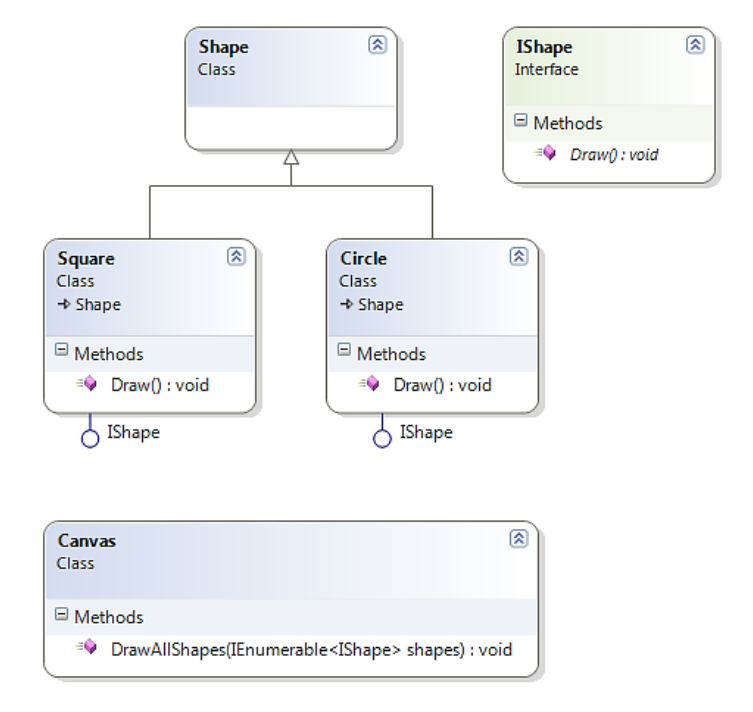
Пример


Shape
Class
Square
Class
→ Shape
Circle
Class
→ Shape
Canvas
Class
Methods
DrawAllShapes(IEnumerable<Shape> shapes) :...

Пример

```
public class Shape { }
public class Square : Shape { }
public class Circle : Shape { }
public class Canvas
{
    public void DrawAllShapes( IEnumerable<Shape> shapes )
    {
        foreach (var shape in shapes)
        {
            if (shape is Square)
            {
                //рисуем квадрат
            }
            if (shape is Circle)
            {
                //рисуем круг
            }
        }
    }
}
```

Решение


Shape
Class
IShape
Interface
Methods
Draw() : void
Square
Class
Shape
Methods
Draw() : void
IShape
Circle
Class
Shape
Methods
Draw() : void
IShape
Canvas
Class
Methods
DrawAllShapes(IEnumerable<IShape> shapes) : void

Решение

```
public interface IShape
{
    void Draw();
}
public class Circle : Shape, IShape
{
    public void Draw() { }
}
public class Square : Shape, IShape
{
    public void Draw() { }
}
public class Canvas
{
    public void DrawAllShapes(IEnumerable<IShape> shapes)
    {
            foreach (var shape in shapes)
            {
                shape.Draw();
            }
    }
}
```

Принцип замещения Лисков

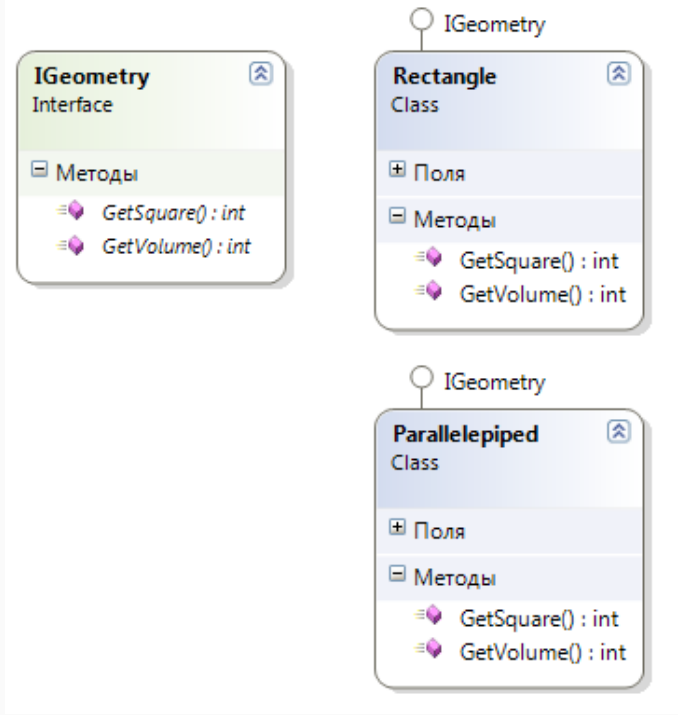
- **Функции, которые используют ссылки на базовые классы, должны иметь возможность использовать объекты производных классов, не зная об этом.**
- **Подтипы должны быть заменяемыми базовыми типами.**

```
public static int DoubleValue( object a )
{
    return ((int)a) * 2;
}
```

Принцип разделения интерфейса

- **Клиенты не должны зависеть от методов, которые они не используют.**

Пример

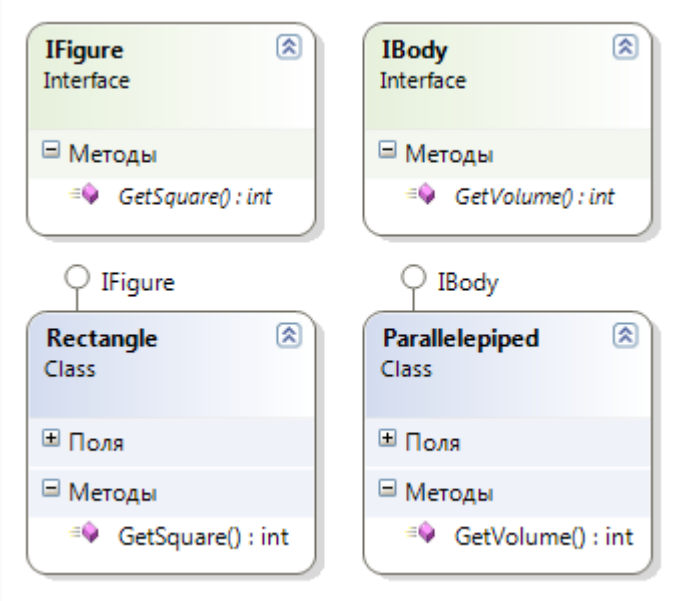

IGeometry
Interface
Методы
GetSquare() : int
GetVolume() : int
IGeometry
Rectangle
Class
Поля
Методы
GetSquare() : int
GetVolume() : int
IGeometry
Parallelepiped
Class
Поля
Методы
GetSquare() : int
GetVolume() : int

Пример

```
public interface IGeometry
{
    int GetSquare();
    int GetVolume();
}
public class Rectangle : IGeometry
{
    public int GetSquare() { return height * width; }
    public int GetVolume() { throw new Exception("Операция не поддерживается!"); }
}

public class Parallelepiped : IGeometry
{
    public int GetSquare() { throw new Exception("Операция не поддерживается!"); }
    public int GetVolume() { return height * width * depth; }
}
```

Решение


IFigure
Interface
Методы
GetSquare() : int
IBody
Interface
Методы
GetVolume() : int
IFigure
IBody
Rectangle
Class
Поля
Методы
GetSquare() : int
Parallelepiped
Class
Поля
Методы
GetVolume() : int

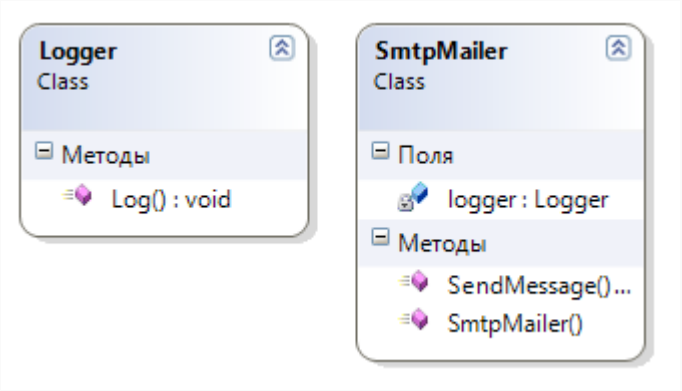
Решение

```
public interface IFigure
{
    int GetSquare();
}
public interface IBody
{
    int GetVolume();
}
public class Rectangle : IFigure
{
    public int GetSquare() { return height * width; }
}
public class Parallelepiped : IBody
{
    public int GetVolume() { return height * width * depth; }
}
```

Принцип инверсии зависимости

- **Модули верхнего уровня не должны зависеть от модулей нижнего уровня. Оба должны зависеть от абстракции.**
- **Абстракции не должны зависеть от деталей. Детали должны зависеть от абстракций.**

Пример


Logger
Class
Методы
Log() : void
SmtpMailer
Class
Поля
logger : Logger
Методы
SendMessage()...
SmtpMailer()

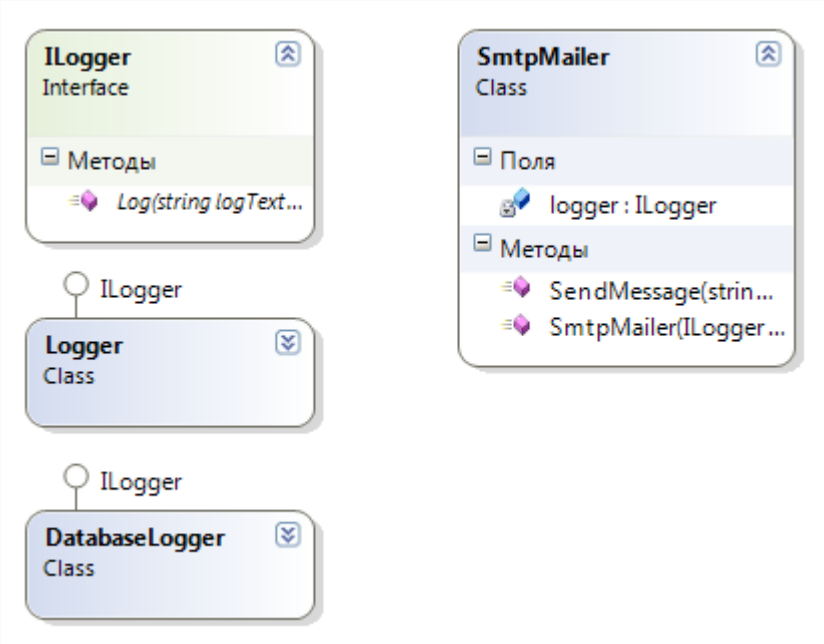
Пример

```
public class Logger
{
    // сохранить лог в файле
    public void Log(string logText) { }
}
public class SmtpMailer
{
    private readonly Logger logger;
    public SmtpMailer() { logger = new Logger(); }

    // отсылка сообщения
    public void SendMessage(string message) { logger.Log(string.Format("Отправлено '{0}'", message)); }
}

public class DatabaseLogger
{
    // сохранить лог в базе
    public void Log(string logText) { }
}
```

Решение


ILogger
Interface
Методы
Log(string logText...
ILogger
Logger
Class
ILogger
DatabaseLogger
Class
SmtpMailer
Class
Поля
logger : ILogger
Методы
SendMessage(strin...
SmtpMailer(ILogger...

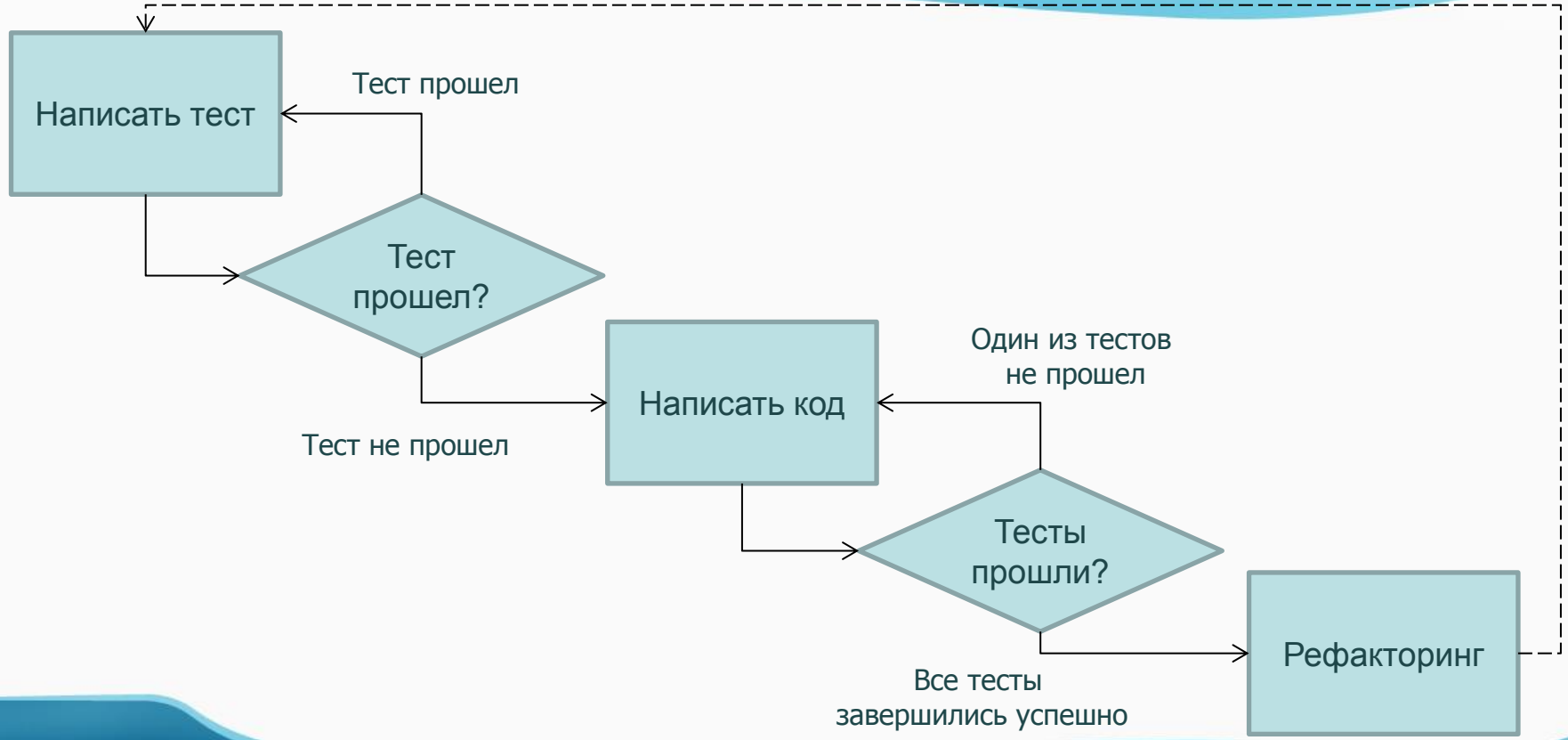
Решение

```
public interface ILogger
{
    void Log(string logText);
}
public class Logger : ILogger
{
    public void Log(string logText) { }
}
public class SmtpMailer
{
    private readonly ILogger logger;
    public SmtpMailer(ILogger logger) { this.logger = logger; }
    public void SendMessage(string message) { logger.Log(string.Format("Отправлено '{0}'", message)); }
}
```

Test-driven development

- Разработка через тестирование (Test-driven development, TDD) – итерационный процесс, при котором сначала пишутся тесты, а потом код, при котором тесты проходят.

Test-driven development



Структура теста

- Arrange-Act-Assert

```
[TestMethod]
public void CalcDistanceReturnsCorrectAnswer()
{
    //Arrange
    var a = new Point(1, 2);
    var b = new Point(4, 6);
    var expectedDistance = 5.0;
    var eps = 0.0001;

    //Act
    var actualDistance = LineSegment.CalcDistance(a, b);

    //Assert
    Assert.AreEqual(expectedDistance, actualDistance, eps);
}
```

Mock-объекты

- Mock – объекты, которые заменяют реальный объект в условиях теста и позволяют проверять вызовы своих членов как часть системы или unit-теста.

```
public IEnumerable<User> GetUsersByBirthMonth( EMonth month )
{
    var list = _userRepository.List();
    return list.Where(x => x.Birthday.Month == (int)month);
}
```

Пример

```
public class CongratulationManager
{
    private IUserRepository _userRepository;

    public CongratulationManager( IUserRepository userRepository )
    {
        _userRepository = userRepository;
    }

    public IEnumerable<User> GetUsersByBirthMonth( EMonth month )
    {
        var list = _userRepository.List();
        return list.Where(x => x.Birthday.Month == (int)month);
    }
    ...
}
```

Пример

```
[TestClass]
public class CongratulationManagerTest
{
    private Mock<IUserRepository> _userRepository;
    private CongratulationManager _congratsManager;

    public CongratulationManagerTest()
    {
        _userRepository = new Mock<IUserRepository>();
        _userRepository.SetupAllProperties();

        _congratsManager = new CongratulationManager(_userRepository.Object);
    }

    ...

}
```

Пример

```
[TestClass]
public class CongratulationManagerTest
{
    ...
    [TestMethod]
    public void ReturnsUsersWithBirtdayInMarch()
    {
        var allUsers = new List<User>();
        var Oleg = new User { Name = "Oleg", Age = 27, Birthday = new DateTime(1988, 3, 1)};
        var Ivan = new User { Name = "Ivan", Age = 25, Birthday = new DateTime(1990, 1, 1)};
        allUsers.Add(Oleg);
        allUsers.Add(Ivan);

        _userRepository.Setup(x => x.List()).Returns(allUsers.AsEnumerable());

        var expectedList = new List<User> { Oleg };

        var actualList = _congratsManager.GetUsersByBirthMonth(EMonth.March);

        CollectionAssert.AreEqual(expectedList, actualList.ToList());
    }
}
```

Спасибо за внимание