

ООП

Семестр 2. Лекция 4

Кафедра ИВТ и ПМ
ЗабГУ

2018

План

Прошлые темы
MVC

SOLID

Outline

Прошлые темы

MVC

SOLID

Outline

Прошлые темы
MVC

SOLID

Model-View-Controller (MVC)

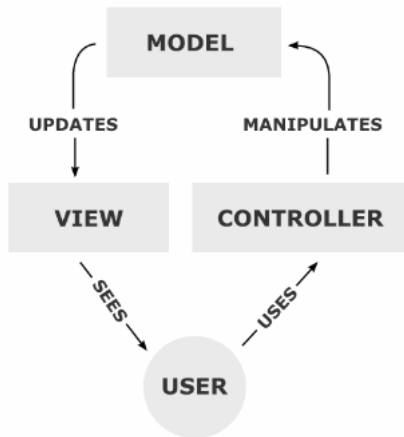
Что представляет собой шаблон проектирования
Model-View-Controller?

Model-View-Controller (MVC)

Что представляет собой шаблон проектирования
Model-View-Controller?

Model-View-Controller (MVC,
Модель-Представление-Контроллер, Модель-Вид-Контроллер)
— схема разделения данных приложения, пользовательского
интерфейса и управляющей логики на три отдельных
компонента: модель, представление и контроллер — таким
образом, что модификация каждого компонента может
осуществляться независимо.

Model-View-Controller (MVC)



Model-View-Controller (MVC)

Модель (Model) предоставляет данные и реагирует на команды контроллера, изменяя свое состояние.

Представление (View) отвечает за отображение данных модели пользователю, реагируя на изменения модели.

Контроллер (Controller) интерпретирует действия пользователя, оповещая модель о необходимости изменений.

Outline

Прошлые темы
MVC

SOLID

SOLID

SOLID - принципы объектно-ориентированного проектирования
- single responsibility, open-closed, Liskov substitution, interface segregation и dependency inversion)

SOLID

- ▶ **S.** Принцип единственной ответственности (The Single Responsibility Principle, SRP)
- ▶ **O.** Принцип открытости/закрытости (The Open Closed Principle, OCP)
- ▶ **L.** Принцип подстановки Барбары Лисков (The Liskov Substitution Principle, LSP)
- ▶ **I.** Принцип разделения интерфейса (The Interface Segregation Principle, ISP)
- ▶ **D.** Принцип инверсии зависимостей (The Dependency Inversion Principle, DIP)

SOLID. Принцип единственной ответственности

- ▶ **S.** Принцип единственной ответственности (The Single Responsibility Principle, SRP)

Каждый объект должен иметь одну ответственность и эта ответственность должна быть полностью инкапсулирована в класс. Все его поведения должны быть направлены исключительно на обеспечение этой ответственности.

SOLID. Принцип единственной ответственности

Применяется практически для любого масштаба: метод, класс, модуль.

Например, согласно этому принципу не стоит помещать бизнес-логику в класс окна приложения.

SOLID. Принцип единственной ответственности

- ▶ Несоблюдение принципа приводит к созданию божественных объектов.

Объект-бог (God object) — антипаттерн объектно-ориентированного программирования, описывающий объект, который хранит в себе «слишком много» или делает «слишком много».

SOLID. Принцип единственной ответственности

Если не соблюдать принцип единственной ответственности...



SOLID. Принцип единственной ответственности

- ▶ Буквальное и неразумное следование приводит - к увеличению числа классов и усложнению приложения.

Пример. Проблема.¹

```
class Person {
  public name : string;
  public surname : string;
  public email : string;
  constructor(name : string, surname : string, email : string){
    this.surname = surname;
    this.name = name;
    if(this.validateEmail(email)) {
      this.email = email;
    } else {
      throw new Error("Invalid email!"); }
  }

  validateEmail(email : string) {
    var re = /^[^\w-]+(?:\. [^\w-]+)*)@((?: [^\w-]+\.)*\w [^\w-]{0,66})\./;
    return re.test(email);
  }

  greet() {
    alert("Hi!");
  }
}
```

¹пример на TypeScript

Пример. Решение

```
class Email {
public email : string;
constructor(email : string){
    if(this.validateEmail(email)) { //...
    }
    else { throw new Error("Invalid email!"); }
}
validateEmail(email : string) {
    var re = /^[^\w-]+(?:\. [\w-]+)*@((?:[\w-]+\.)*\w[\w-]{0,66})\.
    return re.test(email);
}}
```

```
class Person {
    public name : string;
    public surname : string;
    public email : Email;
    // ...
    greet() {
        alert("Hi!");
    }
}
```

Принцип открытости/закрытости



OPEN CLOSED PRINCIPLE

Brain surgery is not necessary when putting on a hat.

Принцип открытости/закрытости

Как можно разработать проект, устойчивый к изменениям, срок жизни которых превышает срок существования первой версии проекта?

Принцип открытости/закрытости

Как можно разработать проект, устойчивый к изменениям, срок жизни которых превышает срок существования первой версии проекта?

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

Принцип открытости/закрытости

- ▶ открыты для расширения: означает, что поведение сущности может быть расширено путём создания новых типов сущностей.
- ▶ закрыты для изменения: в результате расширения поведения сущности, не должны вноситься изменения в код, который эти сущности использует.

Принцип открытости/закрытости

С помощью какого механизма в ООП можно добиться соблюдения принципа?

Наследование.

Принцип открытости/закрытости

С помощью какого механизма в ООП можно добиться соблюдения принципа?

Наследование.

Наследование и полиморфизм

Спецификации интерфейсов могут быть переиспользованы через наследование, но реализации изменяться не должны. Существующий интерфейс должен быть закрыт для модификаций, а новые реализации должны, по меньшей мере, реализовывать этот интерфейс.

Принцип открытости/закрытости

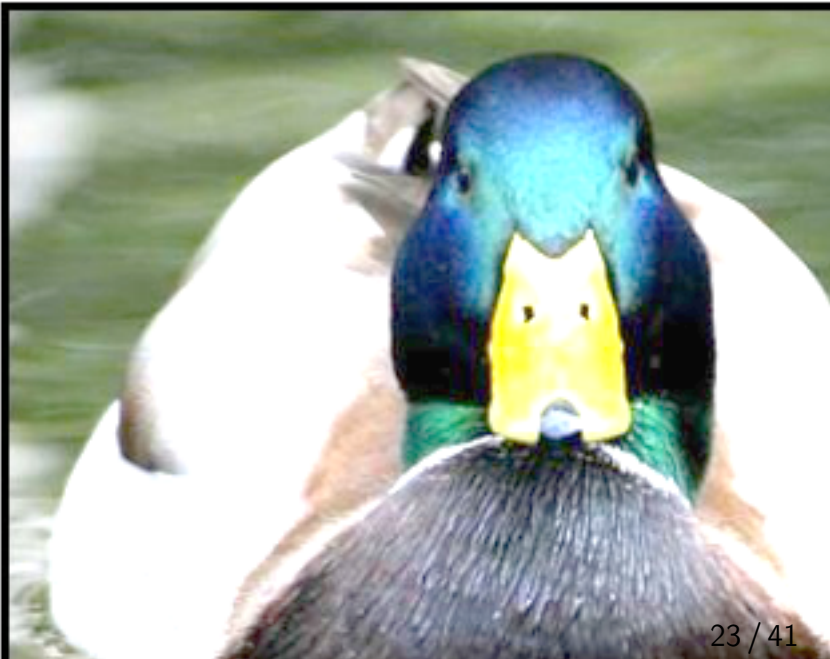
Примеры?

Принцип открытости/закрытости

Примеры?

Создание классов в GUI фреймворках на основе существующих.

Принцип подстановки Барбары Лисков



Принцип подстановки Барбары Лисков

Пусть $q(x)$ является свойством, верным относительно объектов x некоторого типа T .

Тогда $q(y)$ также должно быть верным для объектов y типа S , где S является подтипом типа T .

Принцип подстановки Барбары Лисков

Другими словами...

Функции, которые используют базовый тип, должны иметь возможность использовать подтипы базового типа, не зная об этом.

Принцип подстановки Барбары Лисков

```
class Vehicle {  
    function startEngine() { /*...*/ }  
  
    function accelerate() { /*...*/ }  
}  
  
class Driver {  
    function go(Vehicle $v) {  
        $v->startEngine();  
        $v->accelerate();  
    }  
}
```

Принцип подстановки Барбары Лисков

Если класс Driver использует Vehicle, то и производные от Vehicle классы должны подходить для Driver.

```
class Car extends Vehicle {
  function startEngine() {
    $this->engageIgnition();
    parent::startEngine();}

  private function engageIgnition() {
    // Ignition procedure
  } }

class ElectricBus extends Vehicle {

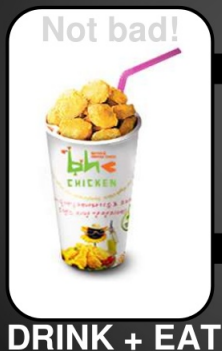
  function accelerate() {
    $this->increaseVoltage();
    $this->connectIndividualEngines();}

  private function increaseVoltage() {
    // Electric logic
  }

  private function connectIndividualEngines() {
```

5 Major Design Principles for OOP

ISP (Interface Segregation Principle)



Принцип разделения интерфейса

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

Принцип разделения интерфейса

Слишком «толстые» интерфейсы необходимо разделять на более маленькие и специфические, чтобы клиенты маленьких интерфейсов знали только о методах, которые необходимы им в работе.

В итоге, при изменении метода интерфейса не должны меняться клиенты, которые этот метод не используют.

Принцип разделения интерфейса. Плохой пример.

```
interface ISmartDevice
{
    void Print();
    void Fax();
    void Scan();
}

class AllInOnePrinter : ISmartDevice
{
    public void Print(){
        // Printing code.
    }

    public void Fax(){
        // Beep booop biiiiip.
    }

    public void Scan(){
        // Scanning code.
    }
}
```

Принцип разделения интерфейса. Плохой пример.

Продолжение

```
class EconomicPrinter : ISmartDevice
{
    public void Print() {
        //Yes I can print.
    }

    public void Fax(){
        throw new NotSupportedException();
    }

    public void Scan(){
        throw new NotSupportedException();
    }
}
```

Производный класс EconomicPrinter будет содержать несвойственные для него методы.

Принцип разделения интерфейса. Решение.

Решение?

Принцип разделения интерфейса. Решение.

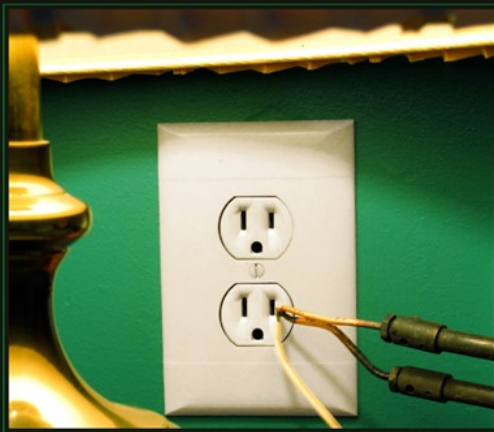
Решение?

```
interface IPrinter{  
  
    void Print();  
  
}
```

```
interface IFax{  
  
    void Fax();  
  
}
```

```
interface IScanner{  
  
    void Scan();  
  
}
```

Принцип инверсии зависимостей



DEPENDENCY INVERSION

Would you solder a lamp directly to the electrical wiring in a wall?

Принцип инверсии зависимостей

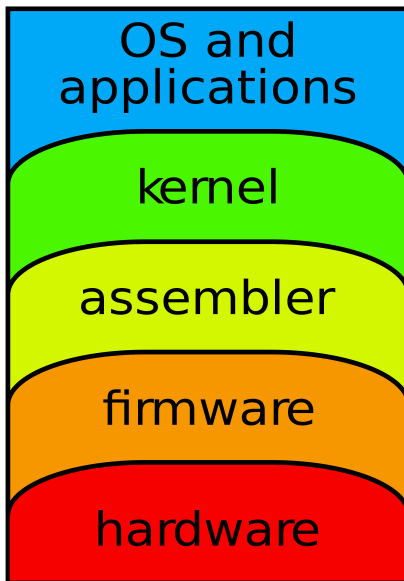
- ▶ Модули верхних уровней не должны зависеть от модулей нижних уровней. Оба типа модулей должны зависеть от абстракций.
- ▶ Абстракции не должны зависеть от деталей. Детали должны зависеть от абстракций.

Уровни абстракции

Абстракция — это модель некоего объекта или явления реального мира, откидывающая незначительные детали, не играющие существенной роли в данном приближении. И

уровень абстракции — это ступень приближения.

Пример уровней абстракции



Плохой пример.

```
class Worker {
    public void work() {
        // ....working
    }
}

class Manager {
    Worker worker;
    public void setWorker(Worker w) {
        worker = w;
    }

    public void manage() {
        worker.work();
    }
}

// Добавим класс SuperWorker
class SuperWorker {
    public void work() {
        //.... working much more
    }
}
```

Проблема?

Плохой пример.

```
class Worker {
    public void work() {
        // ....working
    }
}

class Manager {
    Worker worker;
    public void setWorker(Worker w) {
        worker = w;
    }

    public void manage() {
        worker.work();
    }
}

// Добавим класс SuperWorker
class SuperWorker {
    public void work() {
        //.... working much more
    }
}
```

Проблема?

С классом SuperWorker класс Manager не работает...

Хороший пример.

```
interface IWorker {  
    public void work();  
}  
  
class Worker implements IWorker{  
    public void work() {  
        // ....working  
    }  
}  
  
class SuperWorker implements IWorker{  
    public void work() {  
        //.... working much more  
    }  
}  
  
class Manager {  
    IWorker worker;  
    public void setWorker(IWorker w) {  
        worker = w;  
    }  
    public void manage() {  
        worker.work();  
    }  
}
```

Ссылки и литература

1. Г. Буч. Объектно-ориентированный анализ и проектирование с примерами приложений. 720 с. 2010 г. 700 страниц. Теория. Примеры на C++. Картинки! Вторая половина книги - примеры OOA и OOD с UML диаграммами.
2. MSDN - Microsoft Developer Network
3. Qt 5.X. Профессиональное программирование на C++. Макс Шлее. 2015 и более поздние издания г. 928 с. Книга периодически обновляется с выходом новых версий фреймворка Qt.
4. www.stackoverflow.com - система вопросов и ответов
5. draw.io — создание диаграмм.

Материалы курса

Слайды, вопросы к экзамену, задания, примеры

github.com/VetrovSV/OOP