

# **ПромПрог**

**Лекция 1. Базовые вещи**



docker

# Что такое Docker?

- Докер — это открытая платформа для разработки, доставки и эксплуатации приложений.
- отделить ваше приложение от вашей инфраструктуры
- позволяет запускать практически любое приложение, безопасно изолированное в контейнере

# Когда Docker полезен?

- упаковывание приложения (и так же используемых компонент) в docker контейнеры;
- раздача и доставка этих контейнеров вашим командам для разработки и тестирования;
- размещение контейнеров на серверах, как в дата центры так и в облака.

# Внутри Докера

- образы (images)
- реестр (registries)
- контейнеры

# Как работает Docker?

- Каждый образ состоит из набора уровней.
- Docker использует **union file system** для сочетания этих уровней в один образ.
- В основе каждого образа находится базовый образ.
- использовать образы как базу для создания новых образов.

# Docker

- run - запускает контейнер
  - -i - interactive
  - -t - tty
- –rm - удаляем образ после запуска
- -p, -P - проброс портов
- –name - имя контейнера
- -e - environment

# Пример

- `docker run -it --rm --name myubuntu -e TEST=test  
ubuntu /bin/bash`

# Docker

- docker ps - список контейнеров
- docker images - список images
- docker stop - остановка контейнера
- docker rm - удаление контейнера
- docker rmi - удаление image'a

# Dockerfile

```
FROM eva-dock.sberned.ru/smardata/conda-classifier:0.0.2
```

```
ADD requirements.txt requirements.txt
```

```
RUN pip install --upgrade pip
```

```
RUN pip install -r requirements.txt
```

```
ADD . /Service
```

```
WORKDIR /Service
```

```
RUN mkdir -p /root/.keras && touch ~/.keras/keras.json && \
    echo '{ "floatx": "float32", "epsilon": 1e-07, "backend": "tensorflow",
"image_dim_ordering": "tf" }' > ~/.keras/keras.json
```

```
EXPOSE 8888
```

```
ENTRYPOINT [ "/opt/conda/bin/python", "__main__.py" ]
```

# Сборка image'a

- docker build -t тег .

# **requirements.txt**

- Файлик с библиотеками python

# Docker Compose

- Поднятие нескольких контейнеров одновременно
- Прокидывание сетевого взаимодействия

# ComposeFile

```
version: "3.3"

services:

  cppapp:
    build: .
    restart: always
    entrypoint: python -m service
    networks:
      - backend

  integration_tests:
    build: ./integration_tests
    links:
      - cppapp:cppapp
    environment:
      - SERVICE_HOST=cppapp
    networks:
      - backend

networks:
  backend:
```

# **Базы Данных**

# SQL

- PostgreSQL
  - реляционная
  - SQL
  - легко и понятно

# CAP

- Теорема CAP (известная также как теорема Брюера) – эвристическое утверждение о том, что в любой реализации распределённых вычислений возможно обеспечить не более двух из трёх следующих свойств:
  - согласованность данных (англ. consistency) – во всех вычислительных узлах в один момент времени данные не противоречат друг другу;
  - доступность (англ. availability) – любой запрос к распределённой системе завершается корректным откликом, однако без гарантии, что ответы всех узлов системы совпадают;
  - устойчивость к разделению (англ. partition tolerance) – расщепление распределённой системы на несколько изолированных секций не приводит к некорректности отклика от каждой из секций.

# ACID

- Atomicity – Атомарность. Атомарность гарантирует, что никакая транзакция не будет зафиксирована в системе частично.
- Consistency – Согласованность. Не та же, что в CAP
- Isolation – Изолированность. Во время выполнения транзакции параллельные транзакции не должны оказывать влияние на её результат.
- Durability – Устойчивость. Независимо от проблем на нижних уровнях (к примеру, обесточивание системы или сбои в оборудовании) изменения, сделанные успешно завершённой транзакцией, должны остаться сохранёнными после возвращения системы в работу.

# NoSQL

- MongoDB
  - Легко кластеризуется
  - Документоориентированная
  - Без SQL

# Прочие БД

- InfluxDB - TimeSeries база данных

# Очереди

- RabbitMQ

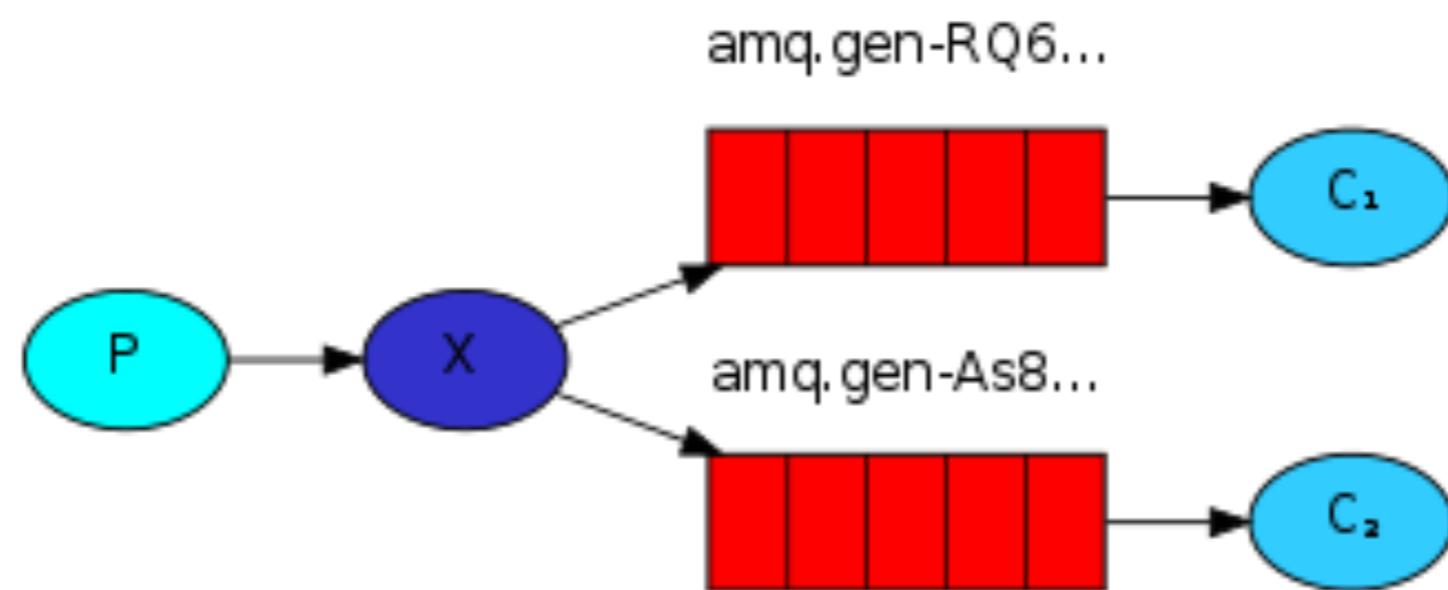
# Терминология

- Producer (поставщик) - программа, отправляющая сообщения
- Queue (очередь) – буффер, хранящий сообщение
- Consumer (подписчик) - программа, принимающая сообщения.

# Точка Доступа (Exchange)

- Основная идея в модели отправки сообщений Rabbit – Поставщик(producer) никогда не отправляет сообщения напрямую в очередь. Фактически, довольно часто поставщик не знает, дошло ли его сообщение до конкретной очереди.
- Вместо этого поставщик отправляет сообщение в точку доступа. В точке доступа нет ничего сложного. Точка доступа выполняет две функции:

# Exchange



# pika

- pika - python обвязка для работы с кроликом

# Producer

```
connection = pika.BlockingConnection(  
    pika.URLParameters("amqp://guest:guest@localhost:  
32769"))  
)  
channel = connection.channel()  
channel.queue_declare(queue='hello')  
data += "a" * (random.randrange(1, 100 * 1024))  
channel.basic_publish(exchange='',  
    routing_key='hello',  
    body=data.encode())  
)
```

# Consumer

```
connection = pika.BlockingConnection(  
    pika.URLParameters("amqp://guest:guest@localhost:32769"))  
channel = connection.channel()  
channel.queue_declare(queue='hello')  
  
def callback(ch, method, properties, body):  
    now_time = datetime.datetime.now().timestamp()  
    recieved = (body.decode())  
    recieved_time = float(recieved)  
  
    print(now_time-recieved_time)  
  
    channel.basic_consume(callback,  
        queue='hello',  
        no_ack=True)  
  
channel.start_consuming()
```

# дз

- Написать 2 программы
  - Одна принимает на вход строчку и кладет ее в очередь
  - Вторая слушает очередь и кладет строчки в базу

# DJANGO

Лекция 2. Пром Прог

# DJANGO

- “The framework for perfectionists with deadlines”
- MVC
- Гибкий шаблонный язык для генерации всего и вся (HTML, CSS, EMAIL и тп)
- ORM для многих баз данных
- Много различных компонент внутри

# КОНЦЕПЦИИ DJANGO

- DRY Principle – “Don’t Repeat Yourself”
- Толстые модели, тонкие вью
- Минимальная логика в шаблонах
- Маленькие, переиспользуемые “apps” (app = python module with models, views, templates, test)

# DJANGO PROJECT LAYOUT

django-admin.py startproject  
<PROJECT\_ROOT>

manage.py

<PROJECT\_DIR>

\_\_init\_\_.py

settings.py

urls.py

wsgi.py

# SETTINGS.PY

- Глобальные настройки проекта
- Как разделить dev vs prod настройки:
  - Создать settings-dev.py и settings-prod.py и использовать symlinks для settings
  - Выделить общие настройки в base-settings.py и импортировать

# ПРИМЕР НАСТРОЕК...

```
DEBUG = True
```

```
TEMPLATE_DEBUG = True
```

```
ALLOWED_HOSTS = []
```

```
# Application definition
```

```
INSTALLED_APPS = (
```

```
    'django.contrib.admin',
```

```
    'django.contrib.auth',
```

```
    'django.contrib.contenttypes',
```

```
    'django.contrib.sessions',
```

```
    'django.contrib.messages',
```

```
    'django.contrib.staticfiles',
```

```
)
```

# DJANGO APPS

- Переиспользуемые модули
- django-admin.py startapp <app\_name>
- Creates stub layout:  
<APP\_ROOT>
  - admin.py
  - models.py
  - templates (directory)
  - tests.py
  - views.py
  - urls.py

# DJANGO MODELS

- Определяются в models.py
- Обычно наследуются от django.db.models.Model

Пример:

```
from django.db import models

class TestModel(models.Model):
    name = models.CharField(max_length = 20)
    age = models.IntegerField()
```

# MODELS

- По умолчанию NOT NULL на все поля. Можно добавить null = True в определение поля:

```
name = models.CharField(max_length=20, null=True)
```

- Отношения определяются через специальный тип поля:

```
models.OneToOneField(model)
```

```
models.ForeignKey(model)
```

```
models.ManyToManyField(model)
```

# MODELS

- Значение по умолчанию - “default”:

```
count = models.IntegerField(default = 0)
```

- Дополнительные опции - через Meta:

```
class TestModel(models.Model):
```

```
    class Meta:
```

```
        abstract = True
```

# MODEL METHODS

- `model.save(self, *args, **kwargs)`
- `model.delete(self, *args, **kwargs)`
- `model.get_absolute_url(self)`
- `model.__str__(self)` [Python 3]  
`model.__unicode__(self)` [Python 2]
- Override with `super(MODEL, self).save(*args, **kwargs)`

# ACTIVATING A MODEL

- Добавить app в INSTALLED\_APPS в settings.py
- Запустить manage.py validate
- Запустить manage.py syncdb
- Migrations
  - Можно использовать кастомные скрипты
  - Можно использовать дефолтные

# SELECTING OBJECTS

- Models включает менеджер объектов по умолчанию objects

- Менеджер позволяет выбрать один или несколько объектов

`Question.objects.all()`

`Question.objects.get(pk = 1)`

Бросает DoesNotExist если объекта нет

`Question.objects.filter(created_date__lt = '2014-01-01')`

- Возвращает спец объект QuerySet

# ПРИМЕР

```
from django.db import models
from datetime import datetime

class TimestampedModel(models.Model):
    created_datetime = models.DateTimeField()
    updated_datetime = models.DateTimeField()

    def save(self, *args, **kwargs):
        if self.id is None:
            self.created_datetime = datetime.now()
        updated_datetime = datetime.now()
        super(TimestampedModel, self).save(*args, **kwargs)

    class Meta:
        abstract = True
```

# ПРИМЕР

```
class Question(TimestampedModel):
    question_text = models.CharField(max_length =
200)

    def __str__(self):
        return self.question_text
```

# FUNCTION VS. CLASS VIEWS

- 2 типа view's – functions и class based views
- Functions – принимают request, должны ввернуть response
- Class based views – поддержка CRUD'a.  
Поддержка Mixin'ов

# ПРИМЕР (ФУНКЦИИ)

```
from .models import Question
from django.shortcuts import render_to_response

def question_list(request):
    questions = Question.objects.all()
    return render_to_response('question_list.html', {
        'questions':questions})
```

# БЫСТРЫЙ CRUD (ШАБЛОННЫЕ КЛАССЫ)

- ListView
- UpdateView
- CreateView
- Если модель указана - сразу генерируем ModelForm
- Форма сохранит данные, если пройдет валидацию
- Переопределить form\_valid() для своей логики при валидации

# ПРИМЕР (КЛАССЫ)

```
from .models import Question
from django.views.generic import ListView

class QuestionList(ListView):
    model = Question
    context_object_name = 'questions'
```

# ШАБЛОНЫ

- Синтаксис:

variables = {{variable\_name}}

template tags = {%tag%}

- Гибки – могут генерировать html, css, xml, email и тп!
- Dot notation – поддерживает указание атрибутов у объектов, ключей и тп

# QUESTION LIST TEMPLATE

```
<!doctype html>
<html lang=en>
<head>
<meta charset=utf-8>
<title>List of Questions</title>
</head>
<body>
{%if questions%}
<ul>
{%for q in questions%}
<li>{{q.question_text}}</li>
{%endfor%}
</ul>
{%else%}
<p>No questions have been defined</p>
{%endif%}
</body>
</html>
```

# URLS.PY

- Определяет маршруты
- Можно использовать regex'ы
- Можно вытаскивать параметры и передавать в приложение:

```
r('^question/(?P<question_id>\d+)/$',views.question_detail)
```

- Модульные – urls.py могут содержать urls из приложений:

```
r('^question/',include(question.urls))
```

# THE QUESTION LIST

```
from django.conf.urls import patterns, url, include
```

```
urlpatterns = patterns("",
```

```
    (r'^questions/$', 'views.QuestionList')
```

```
)
```

OR:

```
from django.conf.urls import patterns  
from views import QuestionListView
```

```
urlpatterns = patterns("",
```

```
    (r'^questions/$', 'views.QuestionList.as_view())
```

```
)
```

# ФОРМЫ В DJANGO

- django.forms - набор механизмов для работы с формами:

```
from django import forms
```

```
class EditQuestionForm(forms.Form):  
    question_text = forms.CharField(max_length = 200)
```

- Обычно используются для работы с одной моделью

# MODELFORMS

- Автоматическая генерация формы по модели.
- Содержит связанную модель
- Можно указывать поля, на основе которых генерировать
- Пример:

```
from django.forms import ModelForm  
from .models import Question
```

```
class QuestionForm(ModelForm):  
    class Meta:  
        model = Question  
        fields = ['question_text']
```

# ИСПОЛЬЗОВАНИЕ MODELFORM

- Создание:

```
form = QuestionForm()
```

- Обновление:

```
question = Question.objects.get(pk = 1)
```

```
form = QuestionForm(instance = question)
```

- Передать форму в шаблон:

```
form.as_p
```

```
form.as_ul
```

```
form.<field_name>
```

```
form.<field_name>.errors
```

# REQUEST & RESPONSE

- Request инкапсулирует запрос. Дает доступ к кукиам, сессиям, пользователям, заголовкам, параметрам и тд и тп
- Response - возврат в ответ. Содержит тип ответа, длину, тело и тд.
- Заранее готовые классы ответов:

HttpResponceRedirect

Http404

# DJANGO EXTRAS

- CSRF Middleware – проверка формы. Надо добавлять в каждый шаблон с формой:  
`{%csrf_token%}`
- Authentication
- Caching
- Sessions
- Messages
- Email
- Logging

# AUTHENTICATION

- Из коробки - через базу.
- Кастомный тип пользователя.

# AUTH DECORATORS

- Живут в django.contrib.auth.decorators
- login\_required  
  @login\_required  
  def function\_view(request):  
    ....
- @user\_passes\_test(lambda u: u.is\_staff)  
  def function\_view(request):  
    ....
- has\_perms

# PECУРСЫ

- Python – <http://www.python.org>
- Django – <http://www.djangoproject.com>
- Python Packages – <https://pypi.python.org>
- Django Packages – [https://  
www.djangopackages.com](https://www.djangopackages.com)

# Frontend

Состояние современной frontend-разработки

<https://tinyurl.com/front-lecture>

Олег Маслов

# Немного истории

Почему Frontend и JS сегодня так важны и разработчики так востребованы?

- SPA-революция
  - Рост производительности браузеров и устройств (V8-революция)
- Рост популярности веба как платформы для разработки и доставки приложений
- React-революция
- Приближение JS к роли "default-language": серверная, десктопная и мобильная разработка
- В ответ на упрощение frontend-разработки растут потребности, сложность приложений, а не падает спрос

# Что нужно знать

Знания, необходимые для начала большинства проектов (или для старта в профессии):

- **JavaScript** всему голова. Это 90%
  - Сам язык
  - ВОМ -- браузерные API, только основные. `fetch`, `storage`,  
~~XMLHttpRequest~~
  - DOM -- манипулирование web-документом “вручную”. Только основы.
- Азы **HTML** и **CSS**  
[Marksheet.io](https://Marksheet.io)
- Понимание как работает **HTTP**

# Что **НЕ** нужно знать

Точнее не стоит изучать заблаговременно:

- JavaScript библиотеки и фреймворки
- CSS пре- и пост-процессоры
- Webpack
- Deploy
- Языки, компилируемые в JS

# Тренды

- Гонка фреймфорков поутихла. Технологии **стабилизировались**.
- Поддержка старых браузеров становится неактуальной. **IE**-кошмар почти в прошлом.
- Рост внимания к **UX / UI**
  - Борьба за время отклика
- Растёт сложность, функциональность приложений

# Тренды: Maintainability

Борьба за поддерживаемость приложений

- Внимание к качеству кода
- Линтеры, style-guides, паттерны
- State-management
- Typescript
- Тесты
- Нетолерантность к экзотическим решениям

# CSS in JS

- Тривиальное переиспользование кода
  - В сравнении с CSS-препроцессорами
- Динамические стили
  - Стили -- функция от состояния компонента
- Низкая когнитивная нагрузка
  - Не нужно быть профи в SCSS
- Производительность
  - Только простейшие селекторы по имени класса
  - Минимальный размер генерируемого CSS
- Библиотеки: [JSS](#), [Fela](#), [Aphrodite](#)

# Тренды (продолжение)

- CSS in JS
- О JavaScript fatigue
- Игнорировать hype
- JavaScript Weekly
  - <http://javascriptweekly.com/>
- Укрепление роли стандартов, открытости  
лицензий

# Framework Holy War

- **React** доминирует всё больше
- Angular стремительно угасает
- Vue JS стремительно растёт
- Всё похоже на React

# Почему мы выбрали Vue, а не React

- Фреймворк, а не библиотека.
  - Готовая архитектура
  - Не нужно изобретать
  - Целостность
  - Единообразие
- Продуктивность
  - Больше “сахара”: множество мелочей, упрощающих жизнь разработчика
- Непредвзятость
  - Например, документация к React не упоминает о Typescript

# Всё же лучше начать с React

- Проще начать
  - API меньше
  - Create React App
- Меньше сахара -- ближе к браузеру
- С React-а начался **компонентный** подход
  - Паттерны и принципы, перенятые остальными
  - Понять Vue после React-а очень просто
- Шире применимость
  - React Native

# Библиотеки и инструменты

- Router
  - React Router
  -
- State-management
  - Redux
  - Vuex
- Взаимодействие с сервером:
  - **REST API**; fetch и axios
  - GraphQL
  - Web-sockets

# Процесс разработки

- Конфигурировать **webpack** больше не требуется
  - create-react-app
  - vue-cli
  - Next.js и Nuxt.js
  - parcel
- **Chrome** с его DevTools доминирует как браузер для разработки
- **VS Code** -- почти редактор по умолчанию

# Оптимизация скорости загрузки

- Асинхронный апдейт при помощи **Service workers**
  - [Making a Progressive Web App](#)
- Серверный рендеринг, Next / Nuxt.js
- Компонентные библиотеки (Iodash, Element UI).
  - Импортируем только что нужно
- Консервативное подключение библиотек
- `<script type="module">`
- Спрайты, векторная графика
- Code Splitting & Lazy Loading
  - [Lazy Loading with React and Webpack 2 – Frontend Weekly – Medium](#)

# Пример React-приложения

```
import React from 'react'
import ReactDOM from 'react-dom'

class HelloMessage extends React.Component {
  render() {
    return (
      <div>
        Привет, {this.props.name}!
      </div>
    )
  }
}

const mountNode = document.getElementById('root')

ReactDOM.render(
  <HelloMessage name="Вася" />,
  mountNode
)
```

# О профессии Frontend-разработчика

- Это уже твердо **отдельная** профессия
- Попытки совмещения с **backend** провальны (только если он не на Node.JS)
- Профессия **верстальщика** умирает, функции переходят frontend-разработчику
- Если нет **UX / UI** - дизайнера, на макроуровне его роль выполняют "художественный" дизайнер и аналитик. На микроуровне -- сам разработчик
- Навыки графического дизайна почти не требуются

# Домашнее задание

- Перевести ваш ToDo List с постраничного django приложения на **SPA** с обращением к backend по api
- Фронтенд сделать исключительно статическими ресурсами и завернуть его в Docker

# Спасибо за внимание!

Лекцию прочёл **Олег Маслов**

**maslovbox at gmail dot com**

<https://github.com/mleg>

**+7 (967) 234-09-30**

**How ToDo your  
homework**



как написать Django приложение |

как написать Django приложение **если я даже не знаю, что это такое?**

Google Search

I'm Feeling Lucky

[mat3e.github.io/brains](https://mat3e.github.io/brains)

**Разобраться в Django,  
составить архитектуру  
самостоятельно  
написать за месяц**

---

**Пройти турориал  
за 40 минут**

---

**Скачать с гитхаба  
готовый пример  
за 5 минут**





django todo list



All

Images

Videos

Shopping

News

More

Settings

Tools

About 4,970,000 results (0.51 seconds)

### Intro to Django - Building a To-Do List - Code - Envato Tuts+

<https://code.tutsplus.com/articles/intro-to-django-building-a-to-do-list--net-2871> ▾

Feb 2, 2009 - We are going to create a very simple application, a **to-do list**, to showcase how to get started with **Django**. We'll use the automatically generated admin interface to add, edit and delete items. The ORM will be used to query the database. The view will pull get the to-do items and pass them onto the template ...

You visited this page on 3/5/18.

### GitHub - rtzll/django-todolist: exemplary django application - small to ...

<https://github.com/rtzll/django-todolist> ▾

GitHub is where people build software. More than 28 million people use GitHub to discover, fork, and contribute to over 79 million projects.

You visited this page on 3/5/18.

### GitHub - shacker/django-todo: A multi-user, multi-group todo/ticketing ...

<https://github.com/shacker/django-todo> ▾

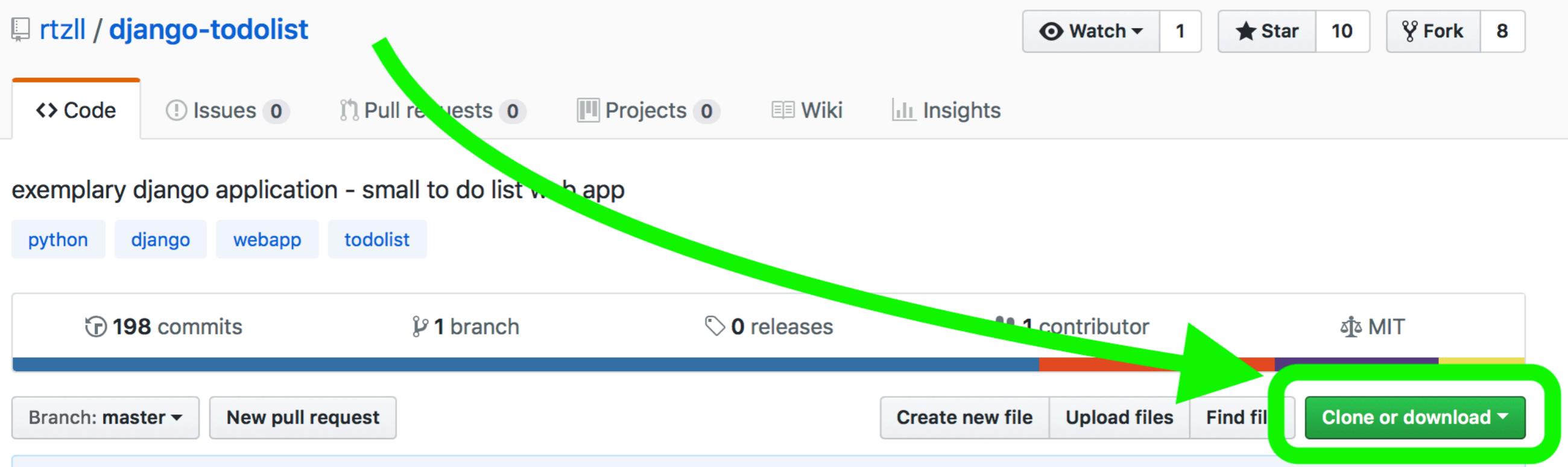
**django-todo** is a pluggable multi-user, multi-group task management and assignment application for **Django**. It can serve as anything from a personal **to-do** system to a complete, working ticketing system for organizations. ... For documentation, see the **django-todo** wiki pages:

### Django Djumpstart: Build a To-do List in 30 Minutes — SitePoint

<https://www.sitepoint.com/build-to-do-list-30-minutes/> ▾

Oct 11, 2006 - **Django** Djumpstart: Build a **To-do List** in 30 Minutes. Being a web developer isn't always exciting. There are lots of tedious things you end up doing over and over again: writing code to talk to a database, writing code to handle page templating, writing an administrative interface ... Sooner or later you start to ...

# Скачиваем готовое приложение



# Скачиваем готовое приложение

`pip3 install -r requirements.txt`

`python3 manage.py migrate`

`python3 manage.py runserver`

\* Works only with python >= 3.4

`http://localhost:8000`

# Tutorial

Google search results for "django todolist app tutorial".

Search term: django todolist app tutorial

Filter: All

About 56,300 results (0.54 seconds)

**Intro to Django - Building a To-Do List - Code - Envato Tuts+**  
<https://code.tutsplus.com/articles/intro-to-django-building-a-to-do-list--net-2871> ▾  
Feb 2, 2009 - In this article, I'll introduce you to **Django** by showing you how to build a simple **to -do list**. ... We are going to create a very simple **application**, a **to-do list**, to showcase how to get started with **Django**. We'll use the automatically ... For this tutorial we are going to use **Django's** test server to test our **application**.

You've visited this page 2 times. Last visit: 3/5/18

**Python Django Tutorial - Build A Todo App - YouTube**  
<https://www.youtube.com/watch?v=2yXfUPwlZTw> ▾  
 Python Django Tutorial  
Jan 29, 2017 - Uploaded by Traversy Media  
If you are new to **Django** and want a better explained **tutorial**, check out the newer crash course - <https://www...>

# Tutorial

Google search results for "django todolist app tutorial".

Search term: django todolist app tutorial

Results:

- Intro to Django - Building a To-Do List - Code - Envato Tuts+**  
https://code.tutsplus.com/articles/intro-to-django-building-a-to-do-list--net-2871 ▾  
Feb 2, 2009 - In this article, I'll introduce you to Django by showing you how to build a simple to -do list. ... we are going to create a very simple application, a to-do list, to showcase how to get started with Django. We'll use the automatically ... For this tutorial we are going to use Django's test server to test our application.  
You've visited this page 2 times. Last visit: 3/5/18
- Python Django Tutorial - Build A Todo App - YouTube**  
Python Django Tutorial  
https://www.youtube.com/watch?v=2yXfUPwlZTw ▾  
Jan 29, 2017 - Uploaded by Traversy Media  
If you are new to Django and want a better explained tutorial, check out the newer crash course - https://www ...

# Tutorial

[Install](#)

[Create First App](#)

# Tutorial

[жми сюда](#)

django-admin startproject todo  
python3 manage.py runserver

[git init](#)

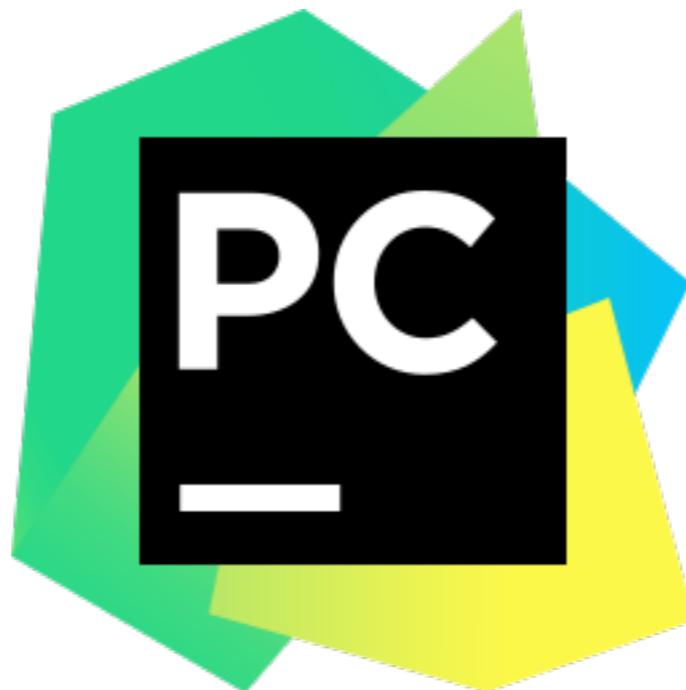
python3 manage.py startapp core

????

PROFIT!!!!

# PyCharm

<https://www.jetbrains.com/pycharm/>



[JetBrains activation server](https://www.jetbrains.com/activation/)

# IDE в игнор!

```
echo .idea > .gitignore
```

Каталог с файлами IDE у каждого разработчика свой

# Writing your first Django app, part 2

Database => sqlite3

Model:

```
from django.db import models

class Todolitem(models.Model):
    text = models.CharField(max_length=200)
    due_date = models.DateTimeField('deadline')
    complete = models.BooleanField()
```

Migrations

Admin

# Writing your first Django app, part 3

## Views

```
def index(request):
    latest.todos = Todoltems.objects[:5]
    context = {'latest.todos': latest.todos}
    return render(request, 'core/index.html', context)
```

## Templates

```
{% if latest.todos %}
<ul>
{% for todo in latest.todos %}
    <li><a href="/todos/{{ todo.id }}"/>{{ todo.text }}</a></li>
{% endfor %}
</ul>
{% else %}
    <p>No todos are available.</p>
{% endif %}
```

## Urls

```
app_name = 'core'
urlpatterns =
    path("", views.index, name='index'),
    path('<int:todoitem_id>', views.detail, name='detail'),
    path('<int:todoitem_id>/toggle', views.toggle, name='toggle'),
]
```

# Writing your first Django app, part 4: Forms

```
<h1>{{ todoitem.text }}</h1>

{% if error_message %}<p><strong>{{ error_message }}</strong></p>{% endif %}

<form action="{% url 'core:toggle' todoitem.id %}" method="post">
    {% csrf_token %}
    <input type="checkbox" name="complete" id="toggle" value="{{ todoitem.complete }}" />
    <label for="toggle">Выполнено</label><br />
    <input type="submit" value="Сохранить" />
</form>
```

# Writing your first Django app, part 4: Forms

```
def toggle(request, todoitem_id):
    todoitem = get_object_or_404(TodoItem, pk=todoitem_id)
    todoitem.completed = request.POST['complete']
    todoitem.save()
    return HttpResponseRedirect(reverse('core:index'))
```

# SPA

**Что гуглим?**

# **Todo list React**

# **Django**

# Готовое решение

[gitlab](#)

git clone https://github.com/dmryutov/react-todo/

cd react-todo

cd backend

pip install -r requirements.txt

python3 manage.py makemigrations

python3 manage.py migrate

python3 manage.py runserver

# Готовое решение

[gitlab](#)

cd frontend

npm install

npm start

browse <http://localhost:8000>

# Tutorial

см. [тут](#) сам турориал и [здесь](#) код

Установить Django Rest Framework

pipenv install djangorestframework

Добавить Serializer

```
class TodolItemSerializer(serializers.ModelSerializer):
    class Meta:
        fields = (
            'id',
            'text',
            'complete',
        )
        model = models.TodolItem
```

# Tutorial

см. [тут](#) сам турориал и [здесь](#) код

## Views

```
class ListTodo(generics.ListCreateAPIView):
    queryset = models.TodoItem.objects.all()
    serializer_class = serializers.TodoItemSerializer

class DetailTodo(generics.RetrieveUpdateDestroyAPIView):
    queryset = models.TodoItem.objects.all()
    serializer_class = serializers.TodoItemSerializer
```

## CORS

pipenv install django-cors-headers

update settings

python3 manage.py runserver

# API. Docs + Tests

Swagger (<https://swagger.io>)

Apiary (<https://apiary.io>)

Postman (<https://www.getpostman.com>)

выбор редакции

# Tutorial

см. [тут](#) сам турориал и [здесь](#) код

```
npm install -g create-react-app
```

```
create-react-app frontend
```

```
cd frontend
```

```
npm start
```

```
browse http://localhost:3000/
```

# Tutorial

см. [тут](#) сам турориал и [здесь](#) код

```
state = {  
  todos: []  
};
```

# Tutorial

см. [тут](#) сам турориал и [здесь](#) код

```
async componentDidMount() {
  try {
    const res = await fetch('http://127.0.0.1:8000/api/');
    const todos = await res.json();
    this.setState({
      todos
    });
  } catch (e) {
    console.log(e);
  }
}
```

# Tutorial

см. [тут](#) сам турориал и [здесь](#) код

```
render() {
  return (
    <div>
      {this.state.todos.map(item => (
        <div key={item.id}>
          <h1 className={`${item.complete ? 'is-complete' : ''}`}>
            {item.text}
          </h1>
          </div>
      ))}
    </div>
  );
}
```

# Dockerize

Django inside docker + postgresql

Frontend: nginx container + static files

# Пром Прог

## Лекция 2

# Код

- Код больше читают, чем пишут
- Код должен быть скорее читаемым и понятным, нежели быстро написанным

# DRY, KISS, YAGNI

- DRY - Don't repeat yourself
- KISS - Keep it stupid simple
- YAGNI - You aren't gonna need it

# HE DRY

```
def parse(self, content):
    strValueNode1 = content.getKey("node1")
    if strValueNode1.startsWith("int"):
        realValueNode1 = int(strValueNode1[4:])
    else:
        raise Exception()

    strValueNode2 = content.getKey("node2")
    if strValueNode2.startsWith("int"):
        realValueNode2 = int(strValueNode2[4:])
    else:
        raise Exception()
```

# DRY

```
def parseIntNode(content, nodeName):
    strValue = content.getKey(nodeName)
    if strValue.startsWith("int"):
        return int(strValueNode1[4:])
    else:
        raise Exception()

def parse(self, content):
    realValueNode1 = parseIntNode("node1")
    realValueNode2 = parseIntNode("node2")
```

# Еще не DRY

```
for i in range(0, N):
    for j in range (0, boot_N):
        av_x += boot_binom[j][i] / (N - 1)
        av_square += boot_binom[j][i]**2 / (N - 1)
    boot_disp_binom[i] = (av_square - av_x**2) / boot_N
    av_square = av_x = 0
    for j in range (0, boot_N):
        av_x += boot_exp[j][i] / (N - 1)
        av_square += boot_exp[j][i]**2 / (N - 1)
    boot_disp_exp[i] = (av_square - av_x**2) / boot_N
    av_square = av_x = 0
    for j in range (0, boot_N):
        av_x += boot_normal_s[j][i] / (N - 1)
        av_square += boot_normal_s[j][i]**2 / (N - 1)
    boot_disp_normal_s[i] = (av_square - av_x**2) / boot_N
    av_square = av_x = 0
    for j in range (0, boot_N):
        av_x += boot_normal_a[j][i] / (N - 1)
        av_square += boot_normal_a[j][i]**2 / (N - 1)
    boot_disp_normal_a[i] = (av_square - av_x**2) / boot_N
```

# Еще все еще не DRY

```
for i in range(0, N):
    av_x_binom = (boot_binom[:, i]).mean()
    av_square_binom = (boot_binom[:, i]**2).mean()
    av_x_exp = (boot_exp[:, i]).mean()
    av_square_exp = (boot_exp[:, i]**2).mean()
    av_x_normal_s = (boot_normal_s[:, i]).mean()
    av_square_normal_s = (boot_normal_s[:, i]**2).mean()
    av_x_normal_a = (boot_normal_a[:, i]).mean()
    av_square_normal_a = (boot_normal_a[:, i]**2).mean()
    boot_disp_binom[i] = (av_square_binom - av_x_binom**2) / boot_N
    boot_disp_exp[i] = (av_square_exp - av_x_exp**2) / boot_N
    boot_disp_normal_s[i] = (av_square_normal_s - av_x_normal_s**2) / boot_N
    boot_disp_normal_a[i] = (av_square_normal_a - av_x_normal_a**2) / boot_N
```

# DRY

```
for i in range(N):
    binom_tuple = calculateTuple(boot_binom, i)
    exp_tuple = calculateTuple(boost_exp, i)
    normal_s_tuple = calculateTuple(normal_s, i)
    normal_a_tuple = calculateTuple(normal_a, i)
```

# HE KISS

```
OutputDeviceFactory::Instantiate(  
    OutputDeviceFactory::StandardOutput).  
CreateWriter().  
WriteBytes(  
    String("Hello, world!").  
BytesRepresentation()  
);
```

# KISS

```
printf(<<Hello, world!>>);
```

# НЕ YAGNI

```
class Vector {  
// ...  
  
    // Реализовать не сложно, вдруг пригодится  
    Vector ToPolar() const;  
  
// ...  
};
```

# YAGNI

```
class Vector {  
// ...  
  
// Реализовать не нужно - вот и не надо  
// VectorToPolar() const;  
  
// ...  
};
```

# SOLID

- Пять основных принципов объектно-ориентированного программирования и проектирования.

# S

- The Single Responsibility Principle
- Каждый объект должен иметь одну ответственность и эта ответственность должна быть полностью инкапсулирована в класс.

# S

- «A class should have only one reason to change.»  
Robert C. Martin
- Класс должен иметь одну и только одну причину для изменений

# S - плохо

```
class Billing:  
    Tarif = 1.5  
  
    def billUser(self, user, minutes):  
        money = minutes * Tarif  
        final_money = self.applySale(user, minutes, money)  
        self.chargeUser(user, final_money)  
        self.sendSms(user, "PRICE IS: {}".format(final_money) )  
  
    def sendSms(self, user, message):  
  
    def chargeUser(self, user, cost):  
  
    def applySale(self, user, minutes, money):
```

# S - xopollio

```
class UserTariff:
```

```
    def __init__(self, user):
```

```
    ...
```

```
    def calculate_cost(self, user, minutes):
```

```
    ...
```

```
class Promo:
```

```
    def apply(self, user, minutes, cost):
```

```
    ...
```

```
class SmsSender:
```

```
    def send(self, phone, message):
```

```
    ...
```

# S - xopollio

```
class Billing:  
    def billUser(self, user, minutes):  
        tarif = UserTariff(user)  
        cost = tarif.calculate_cost(user, minutes)  
  
        sale = Promo(user, tarif, minutes)  
        final_money = sale.apply(user, minutes, cost)  
  
        self.smsSender.send(user, "PRICE IS {}".format(final_money))  
  
        user.balance -= final_money  
        user.save()
```

# S

- Принцип - это не закон
  - Если при изменении кода, отвечающего за одну ответственность, в приложении появляются исправления кода, отвечающего за другую ответственность, то это первый сигнал о нарушении SRP.
  - Если же изменения кода, отвечающего за одну ответственность, не вносят изменения в код, отвечающий за другую ответственность, то этот принцип можно не применять.

# S

- Объекту класса становится позволительно слишком много;
- Любое изменение логики поведения объекта приводит к изменениям в других местах приложения, где это не подразумевалось изначально;
- Приходится тестировать, исправлять ошибки, компилировать различные места приложения, даже если за их работоспособность отвечает третья сторона;
- Невозможно легко отделить и применить класс в другой сфере приложения, так как это потянет ненужные зависимости.

# O

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

# О

- Открыты для расширения - 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 используют.

# O

```
class Obj:
```

...

```
class CSVWriter:
```

```
    def writeToCSV(self, object):
```

...

Надо добавить сохранение в JSON

# О-ПЛОХО

```
class CSVWriter:  
    def writeToCSV(self, object):  
        ...  
  
    def writeToJson(self, object):  
        #... Так как много общего кода
```

# O - Хорошо

```
class Writer:  
    # Общий код
```

```
class CSVWriter(Writer):  
    def write(self, object):  
        ...
```

```
class JSONWriter(Writer):  
    def write(self, object):  
        ...
```

# L

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

# L

- Понятия замещения – если S является подтиповом T, тогда объекты типа T в программе могут быть замещены объектами типа S без каких-либо изменений желательных свойств этой программы

# L

- Проектирование по контракту
  - Предусловия не могут быть усилены в подклассе.
  - Постусловия не могут быть ослаблены в подклассе.

# L - ПЛОХО

```
class Sort:
```

```
    def process(self, iterable):
```

```
        # Логика сортировки
```

```
class SmartSort:
```

```
    def process(self, iterable):
```

```
        answer = MessageBox(
```

```
            "Do you want ASC or DESC sort?"
```

```
)
```

```
# ...
```

# L - хорошо

```
class SmartSort:  
    def process(self, iterable, order='asc'):  
        # ...  
  
#somewhere  
order = MessageBox(...)  
smartSort.process(iterable, order)
```

# L - ПЛОХО

```
def Hello(message):  
    print(message)
```

```
def SmartHello(message):  
    if len(message) == 0:  
        raise Exception()
```

```
    print("SMART" + message)
```

# L-ПЛОХО

```
class Key:
```

```
    def read(self):
```

```
    ...
```

```
    def write(self):
```

```
    ...
```

```
class ReadOnlyKey(Key):
```

```
    def write(self):
```

```
        assert False, "Unsupported"
```

- Принцип разделения интерфейса
- Клиенты не должны зависеть от методов, которые они не используют

- Слишком «толстые» интерфейсы необходимо разделять на более маленькие и специфические
- Клиенты маленьких интерфейсов знают только о методах, которые необходимы им в работе
- При изменении метода интерфейса не должны меняться клиенты, которые этот метод не используют

# I - ПЛОХО

```
class ICollection {  
public:  
    virtual Item& Get(int index) = 0;  
    virtual int Length() = 0;  
    virtual void Sort() = 0;  
    virtual void Dump() = 0;  
    virtual void Append(const Item& item) = 0;  
  
    // Для строк  
    virtual void UpperCase() = 0;  
};
```

# I - Хорошо

```
class IArrayLikeCollection {  
public:  
    virtual Item& Get(int index) = 0;  
    virtual void Append(const Item& item) = 0;  
    virtual int Length() = 0;  
    virtual void Dump() = 0;  
};  
  
class ISortable : ICollection {  
    virtual void Sort() = 0;  
};  
  
class IString : ICollection {  
    virtual void UpperCase() = 0;  
}
```

# D

- Принцип инверсии зависимостей

# D

- 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.

# D-плохо

```
class ISortable : public Array {
```

```
}
```

# D-хорошо

```
class ISortable : public ICollection {
```

```
}
```

# SOLID

- SOLID - это принципы, а не строгие правила
- Это не значит что на них можно забывать

# **Промышленное программирование**

**Лекция 5. «АнтиПаттерны»**

# Антипаттерны

- Широко встречающийся «паттерн» в коде, вредящий качеству создаваемого РО. Иначе - плохая практика

# **Причины антипаттернов**

# Спешка

- Агрессивные deadline
- Низкие требования к качеству кода
- Недостаточное тестирование
- Нарастающий технический долг

# Апатия

- Демотивация
- Нежелание искать подходящее техническое решение
- Сделать кое-как

# узкий кругозор

- Нехватка знаний в области bestpractice
- Непонимание тех или иных базовых принципов

# Лень

- Желание сделать что-то проще
- «И так сойдет»

# Желание выпендриться

- Создание сложных и/или не нужных абстракций
- Взрывной рост сложности ПО

# **Организационные антипаттерны**

# Аналитический паралич

- «Прежде чем принять решение, надо подумать и все взвесить»

# Аналитический паралич

- Решения не принимаются
- Много экспериментов и тестов без бизнес value на выходе

# Аналитический паралич

- Почему
  - Задача переуплотнена
  - Страх ошибки
  - Попытка найти идеальное решение
- Как бороться:
  - Попробуйте уже что-то
  - Поменьше бюрократии

# Аналитический паралич

- На другой стороне - «Принятие решений по наитию» - тоже анти паттерн

# Проектирование комитетом

- Много людей принимает решение о дизайне архитектуры и софта без единого вижена
- Причины:
  - Слабое лидерство
  - Это
  - Недостаток знаний
  - Недостаток соглашений

**«Верблюд - это лошадь  
спроектированная  
комитетом»**

# Проектирование комитетом

- Приводит:
  - Внутренняя неконсистентность
  - Безумная сложность
  - Логические ошибки
  - Отсутствие центральной идеи

# Блокировка на вендоре

- Блокирует ключевые бизнес технологии на конкретном поставщике чего-либо
- Примеры:
  - Оператор связи
  - Канал коммуникации
  - ...

# Блокировка на вендоре

- Результат - потеря контроля
  - Вендор может что-то делать для вас не в срок
  - Может поменять продукт и сломать вам систему
- Решение - уровень изоляции

# Другие орг антипаттерны

- Мешок денег - профитный продукт, который не думает о конкурентах
- Самодурство - управление без оглядки на другие мнения
- «грибной» менеджмент - держим людей в тени, а когда они вырастают - срезаем их =)
- Синдром свидетеля - когда очевидно есть серьезные проблемы, но мы ничего не делаем тк боимся зааффектить кучу других сервисов

# **Project Management anti patterns**

# Death March

- Каждый проект - это произошение
- Мы всегда горим
- Наши сроки - это вчера

# Проблемы

- Выгорание людей
- «Действительно» срочные задачи воспринимаются как обычные и рядовые

# Решение

- Нужны грамотные ПО / ПМ
- Политика управления загрузкой людей
- Ограничение и планирование задач

# Hero Mode

- Есть несколько героев, которые поддерживают на себе всю компанию
- Решение:
  - Правильные ожидания
  - Управление задачами
  - Распределение задач

# Чайка - менеджмент

- «Чайка-менеджер прилетает, создает много звука, заваливает всех и улетает»
  - Взаимодействует только по срочным проблемам
  - Принимает решения в спешке
  - Часто их меняет

# Жажда метрик

- Неоправданное желание кучи красивых «циферок» без понимания что за ними стоит
- Решение:
  - Обучение менеджмента
  - Сбор востребованных метрик
  - Понимание, что за метрики и для чего мы собираем

# **Анти паттерны design'a софта**

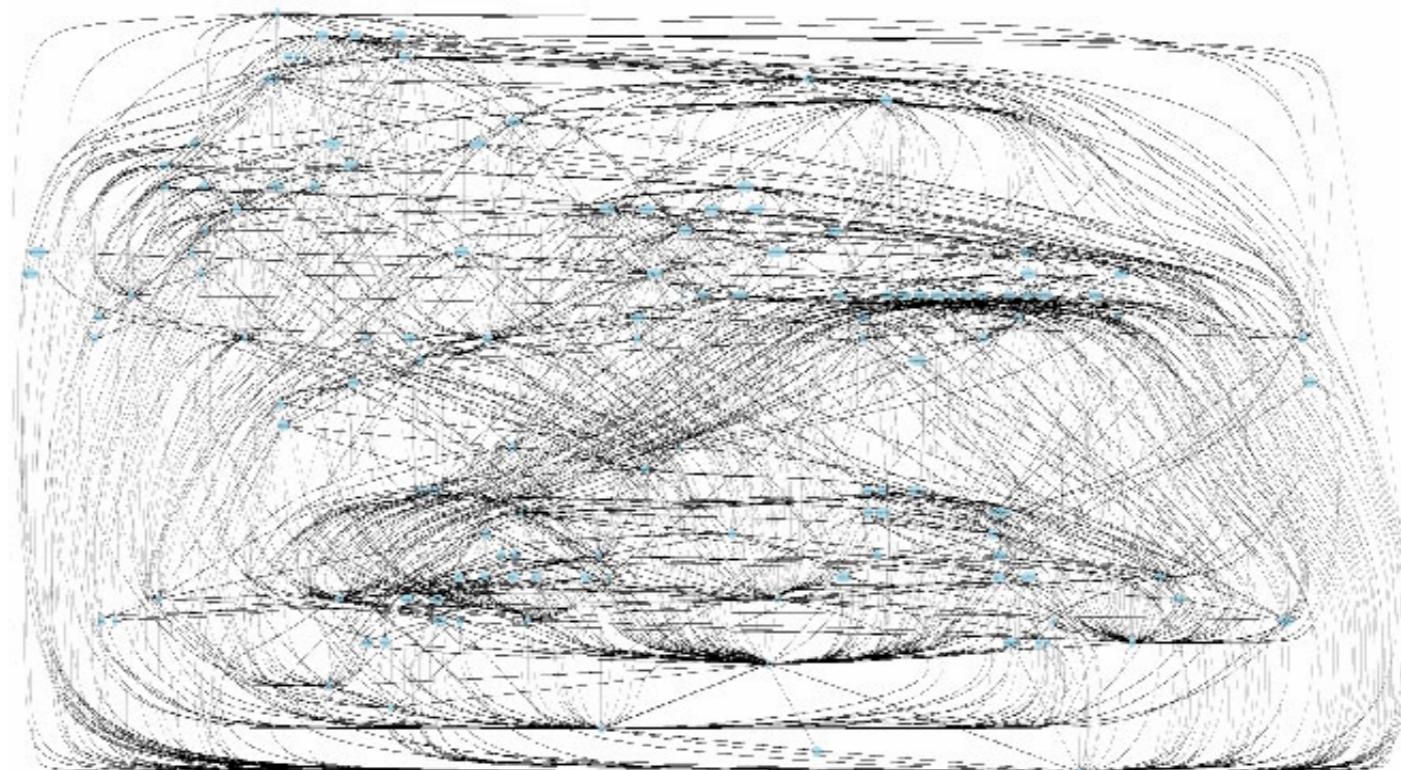
# Раздутье софта

- Каждая следующая версия требует больше и больше ресурсов
- Проблема - ненужные фичи занимают все большую и большую часть ПО
- Результат - работают все медленнее и медленнее, делают не то, что нужно. Действия пере усложнены

# Раздутье софта

- Решение
  - Системы плагинов
  - Unix-philosophy

# Big ball of Mud



- Spaghetti-code
- Граф зависимости между классами почти полный

# Golden hammer

- Лучший паттерн – Observer!
- **Cargo cult programming**
  - использование паттернов без необходимости и понимания

# Hard code

- Magic numbers

```
C:\Documents and settings\Milashka\MyProgram\Data.txt
```

```
CreateProcess(  
L"C:\\\\Users\\\\Mikhail\\\\AppData\\\\Local\\\\Programs\\\\Python\\\\Python35-32\\\\python.exe
```

# Soft code

- всё кастомизируемо
- Файл конфигурации – функциональный язык
- **Accidental Complexity**
- **Cargo cult programming**

# God Object

- Раздувание интерфейса
- **Accidental Complexity**
- **Single Responsibility**

# Раздувание интерфейса

- **Interface soup**
  - Смешивание понятий внутренней реализации
- **Interface bloat**
  - Сложный интерфейс для реализации другим модулем

# Смешивание логики и UI-кода

- Magic pushbutton
- **MVC, MVP, MVVM**

# Better call Super

- CallSuper
- Для реализации прикладной функциональности методу класса-потомка требуется в обязательном порядке вызывать те же методы класса-предка.
- Антипаттерн для Шаблонный метод

# Пром Прог

Лекция 6. Антипаттерны 2

# **Антипаттерны ООП**

# Слабая объектная модель

- Бизнес логика не связана с бизнес объектами

# Слабая объектная модель

- PROS:
  - Разделение между логикой и данными
  - Поддержка Single Responsibility (данные меняются редко, а логика - часто)

# Слабая объектная модель

- CONS:
  - Не истинное ООП
  - Противоречит принципам инкапсуляции
  - Модель не может гарантировать своей корректности ввиду того, что данные и логика разделены
  - Увеличение связанности
  - Дублирование кода

# BaseBean

- Служебный класс, от которого наследуются, чтобы поддержать функционал
- Производный класс становится зависимым
- Стоит предпочесть отношение has-a, а не is-a

# Circle-Ellipse

- Наследуемая от круга, чтобы создать ellipse
- Что нарушаем?

# Yo-Yo проблем

- Чтобы разобраться с любой проблемой нужно постоянно переключаться между кучей исходников и классов
- Объектная модель пере усложнена

# Yo-Yo проблем

- Решения:
  - Граф объектов (особенно наследования) должен быть максимально узким
  - Лучше включение, чем наследование
  - Магическое число - 7 +- 2
  - Понятное именование

# **Кодовые анти паттерны**

**COPY-PASTE, IT'S EVERYWHERE!**



# Copy And Paste

- Почему плохо?

# Copy and Paste

- Увеличение сложности программы
- Однаковость притупляет внимание => увеличение количества ошибок
- «Размножение» ошибок

# Copy And Paste

```
fhead[11] = '\0';
fhead[12] = '\0';
fhead[13] = '\0';
fhead[13] = '\0';
```

# Copy and Paste Programming

- Функциональная декомпозиция
- Не бойтесь методов, которые вызываются ровно в одном месте

```

ourRecordsSubarray.Empty();
ourRecordsSubarray.SetBufferSize( endEqualI - i );
for( int ai = i; ai < endEqualI; ai++ ) {
    ourRecordsSubarray.Add( (*ourRecords)[ai] );
}
foreignRecordsSubarray.Empty();
foreignRecordsSubarray.SetBufferSize( endEqualJ - startJSearch );
for( int aj = startJSearch; aj < endEqualJ; aj++ ) {
    foreignRecordsSubarray.Add( (*foreignRecords)[aj] );
}
presume( ourRecordsSubarray.Size() == endEqualI - i );
presume( foreignRecordsSubarray.Size() == endEqualJ - startJSearch );

```

DRY

```

inline void copySubArray( const CArray& src, int from, int to, CArray& dest )
{
    dest.Empty();
    dest.SetBufferSize( to - from );
    for( int i = from; i < to; i++ ) {
        dest.Add( src[i] );
    }
    presume( dest.Size() == to - from );
}
//...
copySubArray(*ourRecords, i, endEqualI, ourRecordsSubarray);
copySubArray(*foreignRecords, startJSearch, endEqualJ, foreignRecordsSubarray);

```

# Accidental Complexity

```
public int adjustNumber(int x) {  
    int y=x-(x-1) <= 0 ? 1 : x-(x-1);  
    return x % ++y == 0  
        ? x*++y/3*2 : ++x*--y-1;  
}
```

---

```
public int doubleIfEven(int x) {  
    if (x % 2 == 0)  
        return x*2;  
    else  
        return x;  
}
```

# Accidental Complexity

- Избегать «хитроумного» кода
- Вернутся к коду на следующий день и убедится - все ли понятно
- KISS

# Cryptic code

- Использование аббревиатур вместо мнемоничных имён

```
p = 17
```

```
n = 10
```

```
frq2CpFrmDtFctr = 14.32
```

# Cryptic Code

- Почему плохо?

# Cryptic Code

- Увеличение сложности чтение
- Требование пояснения
- Увеличение сложности валидации программы

# Бездумное комментирование

```
// Ключи  
Keys = []
```

# Бездумное комментирование

- Ощущение что читающий - идиот
- Лишняя работа
- Увеличение текста программы => увеличение сложности программы

# Автогенерированные комментарии

```
/**  
 * <p>Проверяет, допустимый ли ход.</p>  
 * <p>Например, чтобы задать ход e2–e4, напишите isValidMove(5,2,5,4);  
 * Чтобы записать рокировку, укажите, откуда и куда ходит король.  
 * Например, для короткой рокировки чёрных запишите isValidMove(5,8,7,8);</p>  
 *  
 * @param fromCol Вертикаль, на которой находится фигура (1=a, 8=h)  
 * @param fromRow Горизонталь, на которой находится фигура (1...8)  
 * @param toCol Вертикаль клетки, на которую выполняется ход (1=a, 8=h)  
 * @param toRow Горизонталь клетки, на которую выполняется ход (1...8)  
 * @return true, если ход допустим, и false, если недопустим  
 */  
boolean isValidMove(int fromCol, int fromRow, int toCol, int toRow) {  
    . . .  
}
```

# Автогенерированные комментарии

- Нужны или нет?
- В чем плюсы?
- В чем минусы?

# Reinventing the wheel

- Создание своей реализации чего-то, что уже есть в библиотеках, оптимизировано и широко используется

# Reinventing the wheel

- Почему плохо?

# Reinventing the wheel

- Отдельная проблема - reinventing the square wheel. То что уже есть, но мы добавим свой костыль

# Programming by permutations

Number of objects	Permutations
1  1	1
2   $2 \times 1 = 2!$	2
3    $3 \times 2 \times 1 = 3!$	6
4    	$4!$
5      $5!$	120
6       $6!$	720
7        $7!$	5,040

# Programming by permutations

- Плодятся проблемы, тк непонятно была новое решение корректно или нет
- Нет понимания что именно было исправлено и почему проблема возникла
- Рост сложности ПО

# Programming by permutations

```
// не знаю, как это работает. НЕ УДАЛЯТЬ И НЕ МЕНЯТЬ!!!
```

- Разберитесь
- Перепишите, наконец!

# Optimisation By Permutation



# Boat Anchor

- Оставление (закомментированного) кода, когда он на самом деле не нужен

# Boat Anchor

- Почему плохо?

# Boat anchor

// код давно не используется. Руки не доходят

- Выкидывайте! Git же есть!
- Программирование «на будущее»
- **Lava flow**
  - Сохранение лишнего или некачественного кода по причине того, что его удаление слишком дорого или будет иметь непредсказуемые последствия
  - Часто антипаттерн для project-manager'a

# Blind faith

- «Фактор невероятности»
- Delivery без тестирования
- Запуск функции без проверки результатов и/или аргументов

```
inline const Element& CDataArray::operator[]( int index ) const
{
    assert( 0 <= index && index < Size() );
    // ...
}
```

# Кодирование исключений

- Добавляем новый кусок кода для каждого «особого» варианта, встретившегося нам
- Почему плохо?

# Кодирование исключений

- Попытаться выработать общее правило
- Периодический рефакторинг
- «У хорошего софта очень мало частных особенностей»

# «Выстрел в темноте»

- «Пользователи говорят, что наш софт тормозит. Я уверен, что виновата очередь! Чиним очередь!»
- Что пропущено?

# «Выстрел в темноте»

- Профилирование
- Сохранение логов
- Сохранение стек трейсов
- Сценарии воспроизведения багов

# Исключительная логика

- Нормальный flow программы построен на исключениях
- Сложно следить за логикой и flow
- Ставится похоже на goto

**Спасибо за  
внимание**

# Пром Прог 7

Паттерны

# Паттерны проектирования

- они же Шаблоны
  - **Стандартные приемы проектирования классов или систем классов**
  - не ограничены конкретным языком
- 
- Стандартные решения стандартных задач
  - Проще поддерживать, чем остроумный авторский код

Опасайтесь «паттернизации  
головного мозга»

# Класс-данные

- Класс для хранения данных
- Внутренний инвариант, согласованное состояние
- Возможность **сериализации**

# Класс-данные

1. Рассмотрите возможность **иммутабельности** данных
2. Нет внутренних состояний  
методы-сеттеры можно вызывать в любом порядке
3. Возможно состояние `IsInitialized`

# Класс-механизм

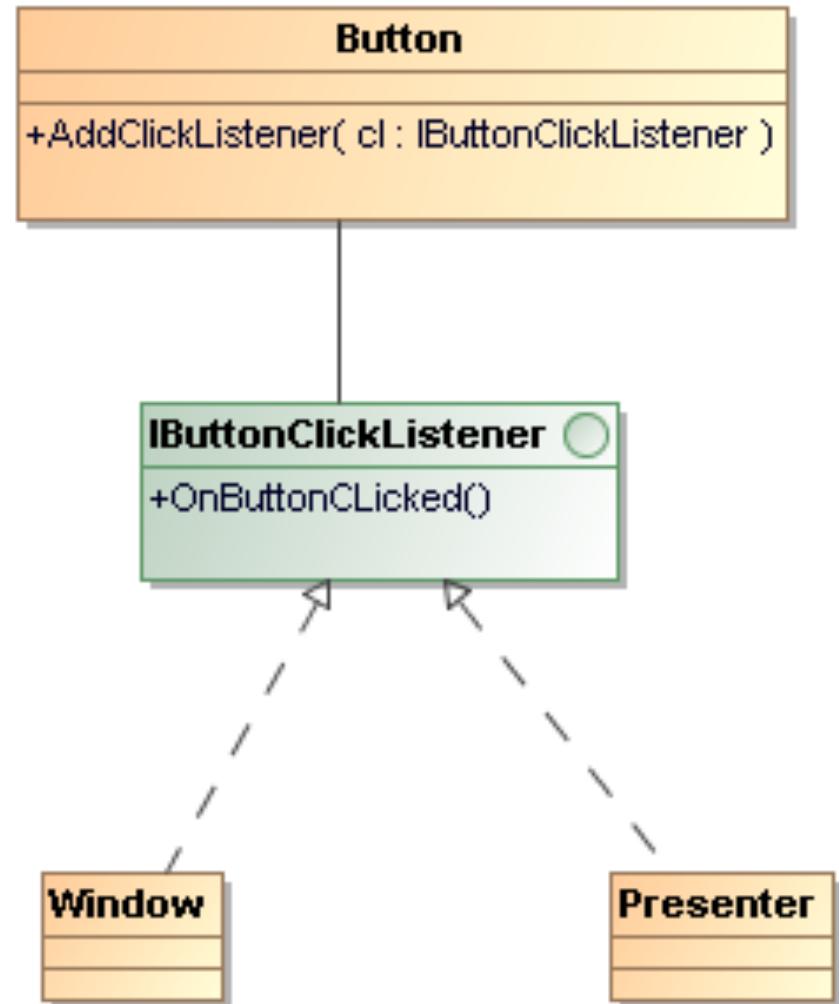
- Сложная операция или несколько похожих операций
  - Достаточно сложна, чтобы не быть методом класса-данных
- Создаётся прямо перед выполнением операции
- Основной интерфейс – единственный метод
- Возможно, метод вызывается для многих запросов
  - неизменные параметры – аргументы конструктора
  - параметры запроса – аргументы метода

# Класс-механизм

- Между запросами не накапливает данных, влияющих на алгоритм
  - только на скорость
- Не зависит истории вызовов
- Не зависит явно от глобального состояния

# Слушатель. Listener. Observer

- Объект, в котором происходит событие ничего не знает о «потребителе» этого события
- IoC
- Часто нужен во фреймворках, на границах подсистем



# Mediator. Нотификация с посредником

- Центральный объект, отвечающий за коммуникацию
- Классы, в которых генерируются события
  - регистрируют события в медиаторе
  - оповещают медиатор
- Классы-потребители
  - регистрируются в медиаторе (с указанием типа события)
- Медиатор
  - делает всю остальную работу
- Фреймворки часто берут роль медиатора на себя
  - Backbone.js
  - WinApi Messages

# Переключатель состояния (\*Switcher, \*Lock)

- Захват ресурсов
- python - менеджер контекста
- Примеры:
  - Connection/Cursor
  - File
  - Примитивы синхронизации

# Синглтон

- Ровно один объект в системе
- Лёгкий доступ к объекту
- Хранение настроек программы
- Проблемы:
  - Глобальное состояние
  - Параллельный доступ
  - Порядок создания/разрушения по отношению к другим синглтонам

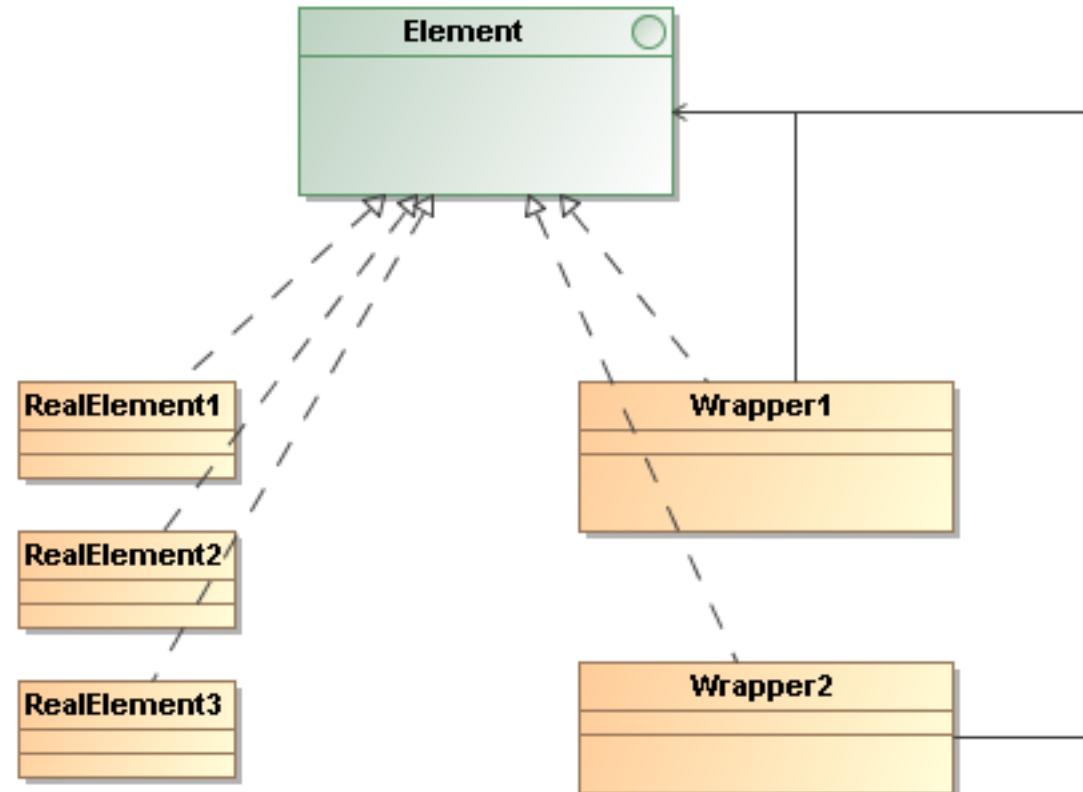
# Переходник, Adapter

- 2 интерфейса делают одно и то же, но методы называются по-разному
- Нужен «переходник»



# Декоратор, обёртка, Decorator

- Добавить какое-то поведению  
оригинальному объекту
- Неважно, какой объект из иерархии  
обращивается
- **Можно обернуть уже обернутый**
- Не меняет семантику интерфейса

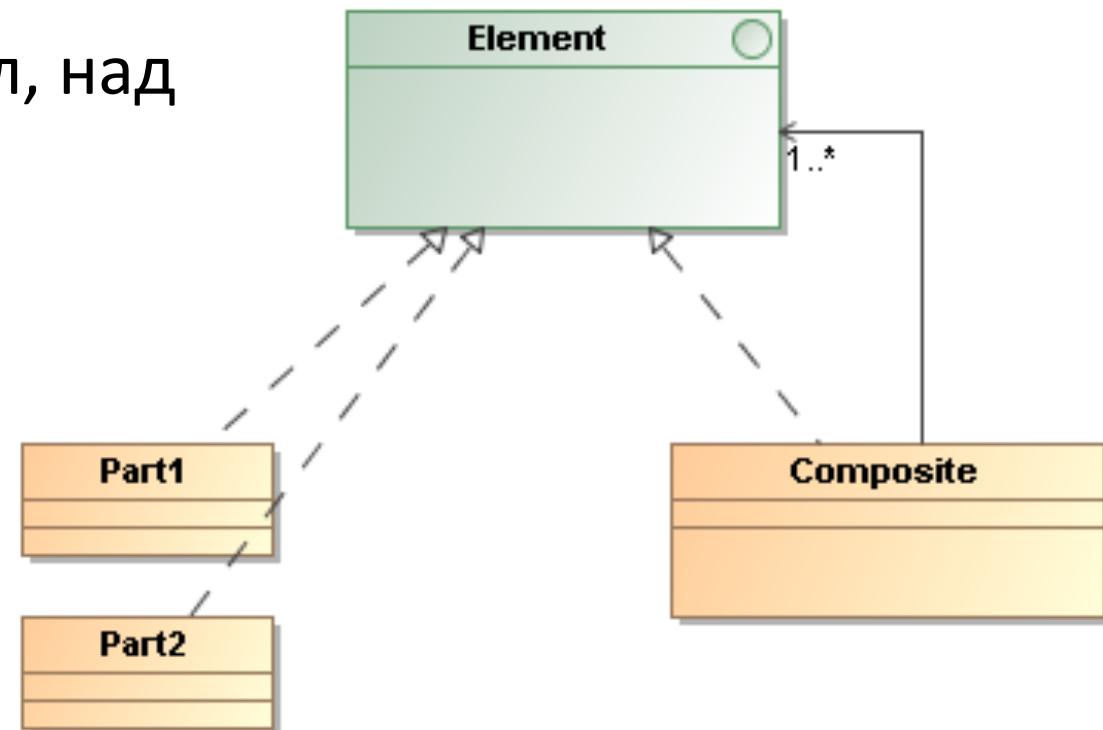


# Прокси, Proxy

- Декоратор, который **меняет** семантику
- Плохой пример – контроль доступа
- Хороший пример – ленивая инициализация объекта
- Обычно, не навешивают несколько прокси подряд,
  - однако можно навесить прокси на декоратор

# Композит, Composite

- Для древовидных структур
- Целое ведет себя так же, как часть
- Важно, чтобы **Composite** не различал, над какой из реализаций он построен



# Композит

- Юнит
  - Лучник
  - Мечник
  - **Отряд - целое ведет себя как частное**

# Builder

- Отделяет создание сложного объекта от его представления
  - В процессе конструирования могут получиться различные реализации интерфейса
- Схема работы
  - добавить запчасть,
  - добавить запчасть,
  - ...
  - получить результат
- Примеры
  - `StringBuilder`
  - `UriBuilder`, `Uri.Builder`

# Builder

- Продукту конструирования не нужны сеттеры
- Параметры по умолчанию – ответственность Builder'a
- Можно продукт оставить иммутабельным!

# Текущий интерфейс. Fluent interface

- Очень характерен для Builder'ов в Java
- Каждый метод, устанавливающий что-либо, возвращает this

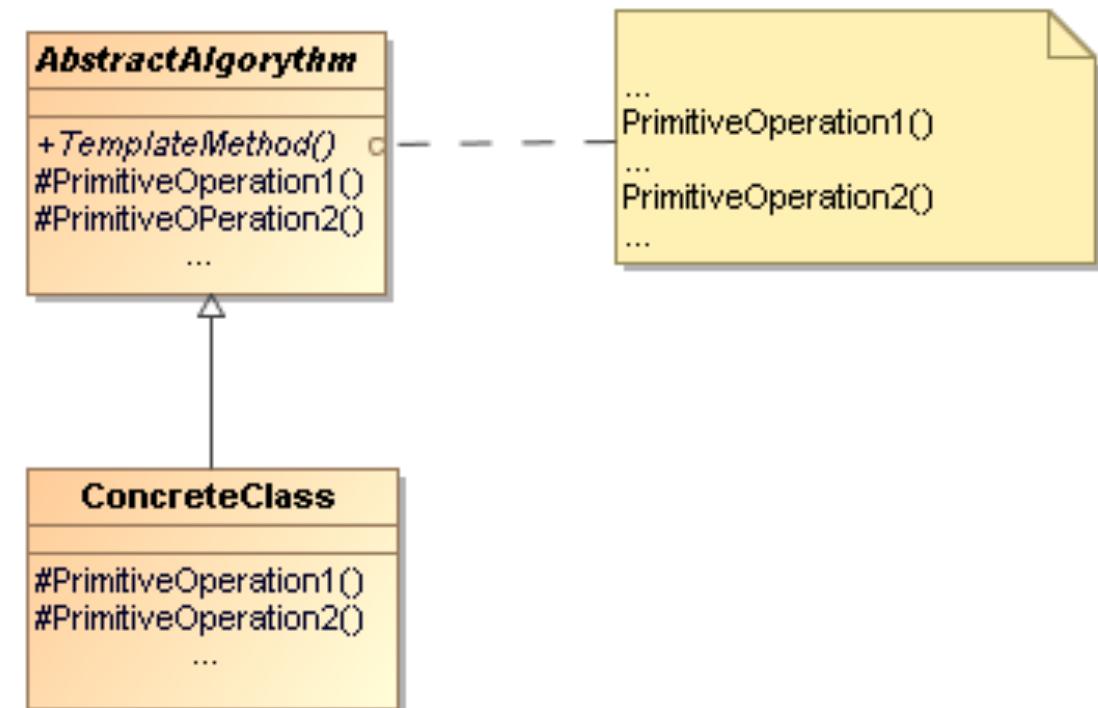
```
Uri.Builder builder = new Uri.Builder();
builder.scheme("http")
    .authority("www.example.com")
    .appendPath("api")
    .appendPath("1.0")
    .appendPath("sample.aspx")
    .appendQueryParameter("key", "[redacted]")
    .appendQueryParameter("mapid", value);
return builder.build();
```

# Еще порождающие шаблоны

- Фабрика
- Фабричный метод
- Прототип

# Шаблонный метод, Template Method

- Базовый класс определяет основу алгоритма, но остаётся абстрактным
- Наследники доопределяют поведения алгоритма в конкретных методах



# Chain of Responsibility

```
handlers = [handle_fuel, handle_km, handle_oil]
garage = Garage()
for handle in handlers:
    garage.add_handler(handle)
garage.handle_car(Car())
```

# Chain of Responsibility

```
def handle_fuel(car):
    if car.fuel < 10:
        print "added fuel"
        car.fuel = 100

class Garage:
    def __init__(self):
        self.handlers = [ ]

    def add_handler(self, handler):
        self.handlers.append(handler)

    def handle_car(self, car):
        for handler in self.handlers:
            handler(car)
```

# Chain of Responsibility

- Много разных обработчиков могут обработать один и тот же объект в зависимости от разных ситуаций
- Не хотим указывать кто именно должен обработать тот или иной случай
- Динамическое создание обработчиков

# Command

- Инкапсулирует **запрос и получателя** запроса. Все команды имеют единый интерфейс
- Хранится в **инициаторе** запроса, так что **инициатор** не связан с **получателем**
- Команды – объекты. Их легко объединять в коллекции
- Легко добавить новую команду

# Iterator / Cursor

- Ответственность по перебору коллекции переносим в вспомогательный класс
  - Сегрегация
- Одновременно обходим коллекцию по-разному
  - Состояние обхода коллекции хранится в итераторе
- Общий интерфейс перебора разных коллекций

# State

- Все действия над объектом выносятся в общий интерфейс IState
- Конкретное состояние реализуется конкретным классом, реализующем IState
- Если действие изменяет состояние, нужно вернуть новое состояние контексту
- Минус
  - Семантика «размазана» на несколько классов. Не инкапсулирована

# Strategy

- Набор классов-механизмов с общим интерфейсом
- Механизмы делают одно и то же, но разными способами
- Рекомендуется использовать паттерн, если есть механизм, часто ветвящийся в зависимости от своего типа
- Объектами, следующими единой стратегии, параметризуется другой класс

# Visitor

- Реализуем обход коллекции элементов разными алгоритмами
- Разные классы-данные
  - реализуют общий интерфейс Element
  - реализуют метод Accept( IVisitor v), вызывая v.ProcessConcreteElementA( this )
- Разные алгоритмы
  - реализуют общий шаблон IVisitor
  - IVisitor содержит методы ProcessConcreteElementA, ProcessConcreteElementB, ...
  - Конкретный алгоритм-visitor реализует обработку конкретных элементов так, как ему нужно

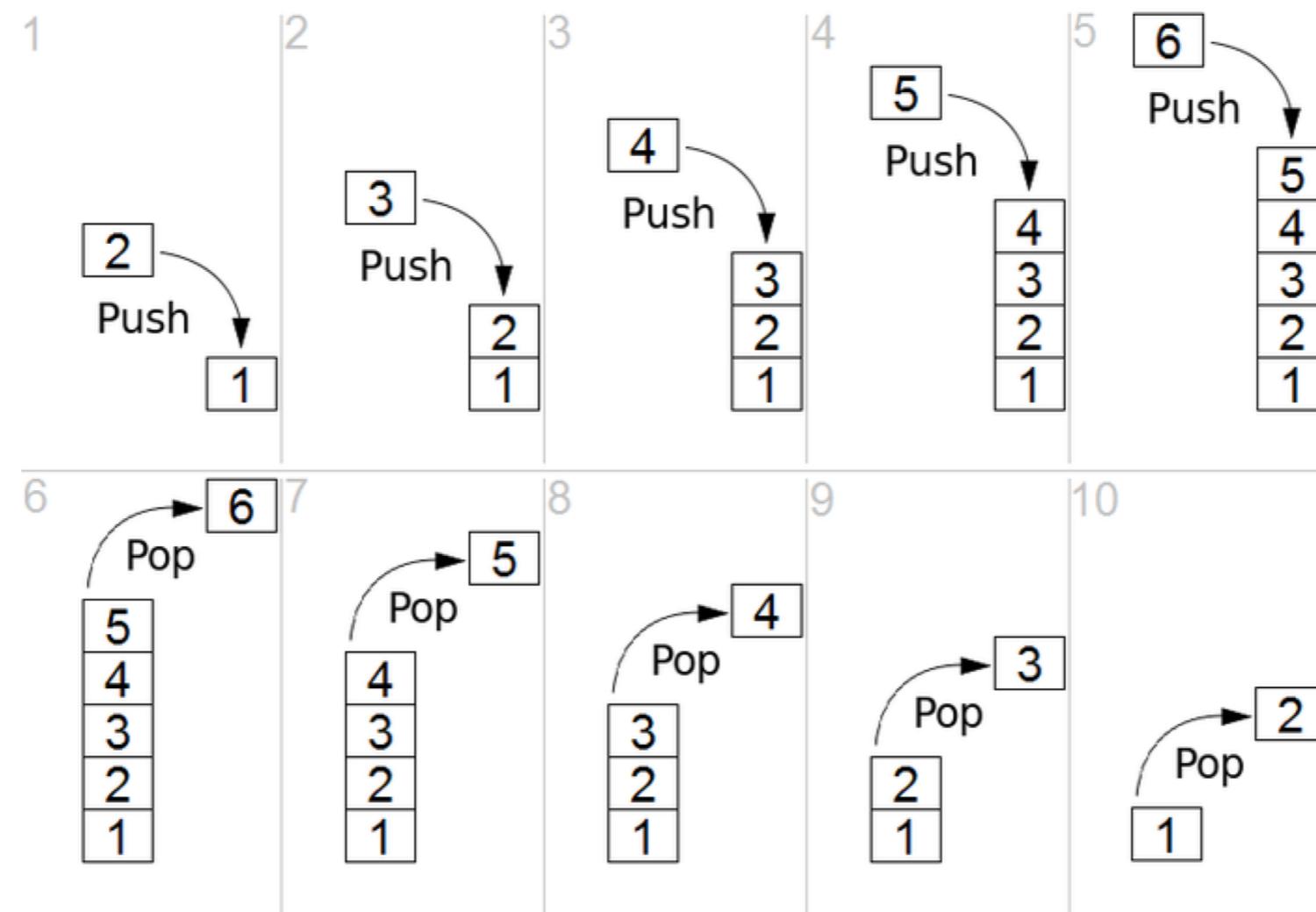
Спасибо за внимание

# **Пром Прог**

## **Лекция 9. ООП**

# ООП

**Абстрактный тип данных (АТД)** - это математическая модель для типов данных, где тип данных определяется поведением (семантикой) с точки зрения пользователя данных, а именно в терминах возможных значений, возможных операций над данными этого типа и поведения этих операций.



# Класс

- Расширяет АТД
- Вводит понятие типа
- Моделирует
  - Сущность реального мира
  - Алгоритм
  - Программный объект

# ООП

- **Абстракция**

Выделенный набор значимых характеристик объекта

- **Инкапсуляция**

Объединение данных и методов, работающих с ними, в классе, и скрытие деталей реализации от пользователя

- **Полиморфизм**

Использование объектов с одинаковым интерфейсом без информации о типе и внутренней структуре объекта

- **Наследование**

Переиспользование кода с обобщением или уточнением

Абстракция

Данных

Процесса

- Структуры
- Классы / объекты

- Структурное программирование
- Подпрограммы (процедуры, функции)

# АТД

- Определяет множество объектов
  - Определяя интерфейс этих объектов
  - Структура данных скрыта для клиентов этих объектов
- ООП
  - Интерфейсы
  - public-интерфейс класса

# Пример

```
type
  // Шаблон класса Stack
  Stack<T> = class
    constructor Create;

    // Кладет элемент x на вершину стека
    procedure Push( x: T );

    // Возвращает элемент типа T, снимая его с вершины
    // стека
    function Pop : T;
    // Возвращает значение элемента на вершине стека
    function Top : T;

    // Возвращает истину, если стек пуст
    function IsEmpty : boolean;
  end;
```

# Инкапсуляция

- Без инкапсуляции
  - Структура данных известна
  - Можно описать любой алгоритм над данными
  - Обработку данных можно проводить где угодно
- Инкапсуляция
  - Объединение структуры данных и алгоритмов
  - Скрытие данных

# Инкапсуляция

- **Файл**
  - Один файл - одна сущность
  - Один файл - похожие маленькие сущности
  - Один файл - законченная маленькая логика
- **Модуль**
  - Объединили логически связанные подпрограммы
  - Конкретный функционал для большой подзадачи
- **Сервис**
  - Большая задача

# Пакет / Модуль

- Цель: задание области видимости и сокрытие деталей
- Включение пакетов друг в друга
- Иногда файловая структура повторяет структуру пакетов
- `import`

# Пакет / Модуль

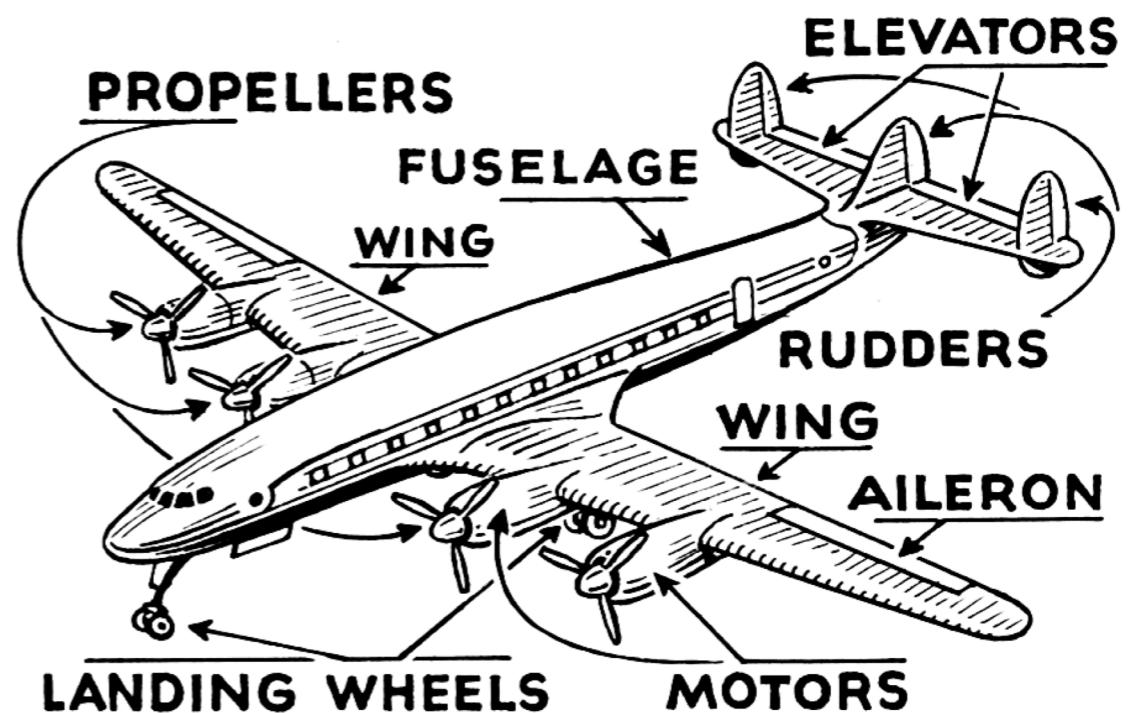
- Программная единица
- Скрывает детали реализации
- Явно экспортирует внешний интерфейс
- «*Для написания одного модуля должно быть достаточно минимальных знаний о тексте другого*»
  - Д. Парнас (David Parnas)

# Сокрытие информации

- Цель: согласованное изменение данных и управление ресурсами
- Интерфейс класса
  - public
- Реализация класса
  - private
- Интерфейс для наследования
  - protected
- Внутренний интерфейс для пакета
  - “package-private”

# Члены класса

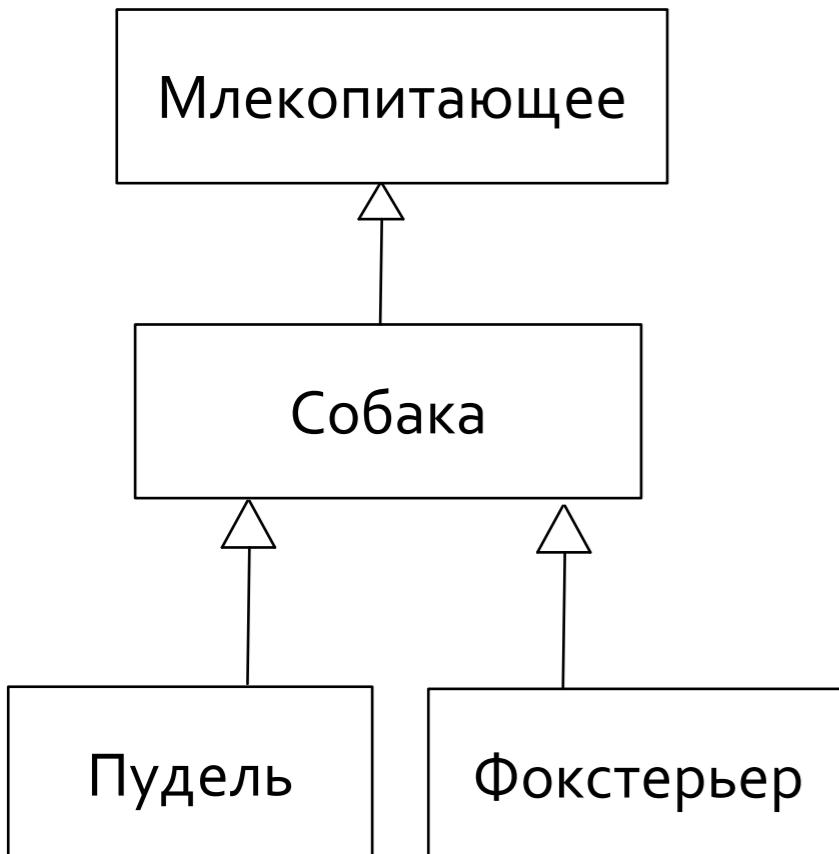
- Агрегация
- Отношение HAS-A
- Часто хорошей идеей является не давать public-доступ к членам



# Наследование

## Динамический полиморфизм

# Априорное понимание наследования



- Птица является животным
- Массив целых чисел является массивом
- Диалоговое окно является окном
- Средство повторного использования кода

Отношение “IS A”

# Природа наследования

- Наследование – это «расширение» или «сужение» ?
  - Отношение обобщения. Базовый – более общий
  - Наследник добавляет методы, члены. Описывает больше объектов.  
Наследник – шире, значит более общий?
- Отношение IS-A // хорошая практика
- Принцип подстановки Лисков
  - Определяет не подклассы, а подтипы
  - Требование соблюдения контракта: предусловия, постусловия

# Формы наследования

- Наследование для реализации
  - Реализация абстрактных методов предка
  - *Framework*
  - Подтип
- Наследование для конструирования
  - Содержимое надкласса используется по своему усмотрению
  - Используется для реализации непредусмотренного поведения
  - Не подтип

# Формы наследования

- Наследование для специализации
  - Фиксация атрибутов предка
  - Реализация собственных методов с помощью методов предка
  - Подтип
- Наследование для обобщения
  - Добавление атрибутов
  - Переопределение методов предка, расширяя интерфейс и функциональные возможности
  - Не подтип

# Формы наследования

- Наследование для расширения интерфейса
  - Дополнение методов предка новыми (удобными) методами
  - Подтип
- Наследование для ограничения
  - Скрытие методов родителя, assert на невалидные методы
  - Не подтип

# Наследование интерфейса и реализации

- Интерфейс
  - обязанность выполнить контракт
- Наследование интерфейса
  - Абстрактный метод
  - Чистый абстрактный класс
- Наследование реализации
  - Наследование, как средство переиспользования кода
  - Часто агрегации достаточно

# Интерфейс

- Отделяет свойства объекта во взаимодействии от прочих черт
- ~Объявление функции
- Протокол

# Множественность наследования

- Single-наследование
  - public, protected, private – предки
- Множественное наследование
  - «Потому что у тебя есть мама и папа». // Б. Страуструп
  - Проблема ромба
  - virtual – наследование \*

# Наследование, конструктор, деструктор

- Порядок конструирования
  - Ключевое слово base, super
  - Порядок разрушения объекта
- Виртуальный деструктор
  - Обязателен для класса с виртуальными методами
  - Не является абстрактным
- Виртуальный конструктор

# Полиморфизм

- «Динамический» полиморфизм
  - позднее связывание
  - ≠ Статический полиморфизм // связывание на этапе компиляции
- Виртуальный метод
  - Динамическое связывание вызова метода с реализацией метода
  - Вызываем разные методы для объектов разных наследников общего предка

# Иерархия классов

- (`Object`) `everything` – наследование от общего корня
  - Java, C#, Python, Objective-C
  - Гарантиированная минимальная функциональность
  - Обобщенные коллекции  
`Collection<?>`
- Произвольные иерархии наследования
  - C++

# Полиморфизм

- Реализация
  - Таблица виртуальных функций
    - Проблемы
      - Лишние инструкции процессора
      - Необходимость заранее знать/указывать какие функции виртуальные, а какие просто перегружены
  - Языки без «невиртуальных» методов
    - Проблемы - overhead на вызов каждого метода

# Полиморфизм

- Принцип Open-Closed
  - Для реализации новой функциональности не нужно переписывать существующий код
  - Нужно добавить новый класс и там описать новую функциональность
  - Общий интерфейс гарантирует, что новый класс заработает со старым кодом
- Шаблон проектирования «Абстрактный метод»
  - Общая реализация алгоритма в базовом классе
  - Виртуальные (protected) методы, реализованные в наследнике уточняют поведение алгоритма в конкретных шагах

# Mixin

- Примеси
  - Классы определяли атрибуты, а методы к ним «примешивались»
  - «ортогональная» функциональность
  - аспект
- может реализовать интерфейс

```
import json

# это Mixin!
class JsonSerializable():

    def toJSON(self):
        return json.dumps(self.__dict__)

    # подмешиваем
class Person(JsonSerializable):

    def __init__(self, name, age):
        self.name = name
        self.age = age

signer = Person("Kurt Cobain", 27)
print(signer.toJSON())
# {
#     "name": "Kurt Cobain",
#     "age": 27
# }
```

# Templates, Generics

Статический полиморфизм

# Templates

- Параметрический полиморфизм
  - «Утиная типизация»
  - Принцип SFINAE Substitution Error Is Not A Failure
  - Вывод типов: иногда не нужно явно указывать T
- Столько копий класса, сколько различных параметров T
  - Явное инстанцирование
- Специализация шаблона
  - `Bender<std::string>` будет делать иначе `DoYourJob(std::string*)`

```
template<class T>
class Bender {
    // Сгибаем что угодно
    void DoYourJob( T* bendable ) {
        bendable->Bend( /* */ );
        std::cout << "Bite my ...";
    }
};
```

# Generics

- C++:

```
std::list<Person>* foo = new std::list<Person>(); // фактически, новый класс
```

- C#:

```
List<Person> foo = new List<Person>(); // тоже новый псевдо-класс (JIT)
```

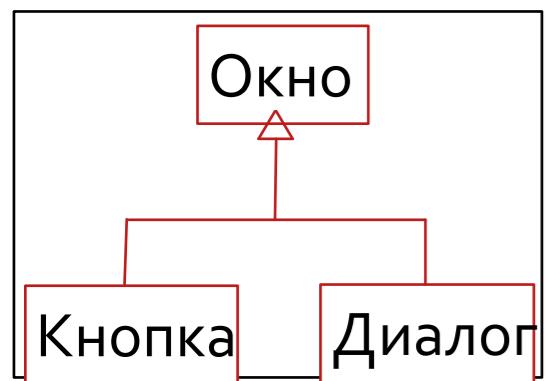
- Java:

```
ArrayList<Person> foo = new ArrayList<Person>(); // информация о типе выкидывается  
// фактически, эквивалентно ArrayList или ArrayList<Object>
```

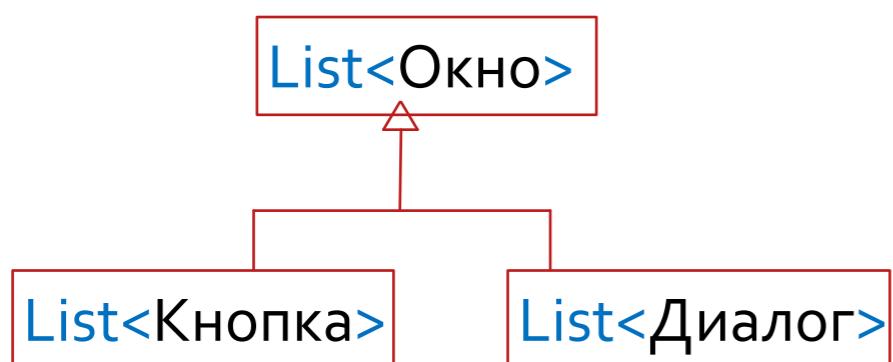
# Вариативность

- Ковариации типов
- Контрвариации типов

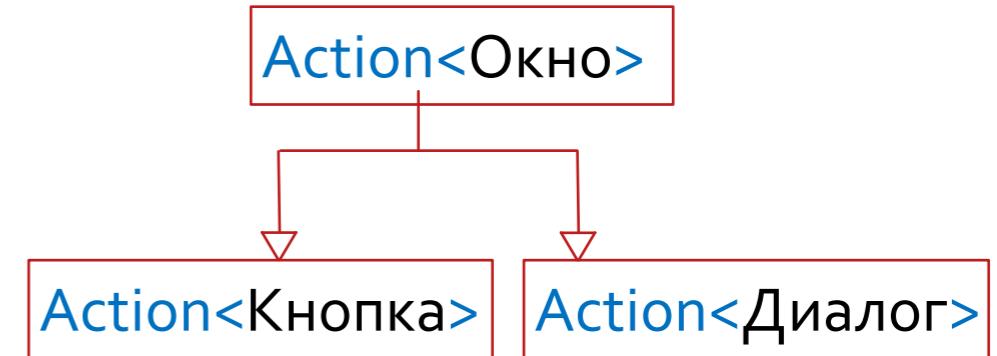
# Производные типы и наследование



Ковариация



Контрвариация



```
// скрыть все окна
public void hideAll( List<Окно> );
hideAll( new List<Кнопка> {
    кнопкаОткрыть, кнопкаЗакрыть } );
// => List<Кнопка> : List<Окно>
```

```
Action<Окно> раскрасить =
окно => окно.SetColor( Color.Red );

List<Кнопка> кнопки = new List<Кнопка> {...} ;
кнопки.ForEach( раскрасить );
// по факту мы должны передать в метод ForEach
// объект делегата Action<Кнопка>
```

# **Интроспекция и Рефлексия**

# Reflection

- Отражения, Рефлексия
- Метаданные для Type inspection
- Вся информация о типах в Runtime:
  - Сборки
  - Модули
  - Классов
  - Методы
  - Поля и данные
  - Модификаторы, атрибуты
- Получить тип объекта
- Создать экземпляр объекта по типу
- Вызвать метод созданного экземпляра
- Перечислить интерфейсы, реализованные классом
- Получить сигнатуру метода
- ...
- Создать Proxy-обёртку класса, подменить метод

# Reflection

- Выполнение исходного кода, полученного извне
  - Java:  
`eval("console.log('eval is evil')");`
- Модификация или анализ кода, полученного извне
  - Python: модуль dis  
`>>> foo.func_code.co_code 'd\x01\x00}\x01\x00|\x01\x00|\x00\x00\x17S'`

# Reflection

- Когда использовать
  - Дополнительные возможности позднего связывания
  - Системы плагинов
  - Фреймворки
  - IDE

```
[HttpPost]
[Authorized]
[ValidateAntiforgeryToken]
void PostsController::Add( Post
post )
{
    if( post.Title.IsEmpty() ||
post.Body.IsEmpty() ) {
        throw new InvalidPost();
    }

    db.posts.Add( post );
}
```

# Reflection. ORM

- Другой пример ORM - Object-Relational Mapping
  - Связь схемы БД с объектами
  - Конвертировать методы в запросы – легко. Как конвертировать данные?

```
public class Phone
{
    public int Id { get; set; }
    public string Name { get; set; }
    public int Price { get; set; }
}
```

```
public class PhoneContext : DbContext
{
    public DbSet<Phone> Phones { get; set; }
}
```

# На что похоже Reflection?

```
Class c = obj.getClass();

Method[] methods = c.getMethods();

for (Method method : methods) {
    System.out.println("Имя: " + method.getName());
    System.out.println("Возвращаемый тип: " + method.getReturnType().getName());

    Class[] paramTypes = method.getParameterTypes();
    System.out.print("Типы параметров: ");
    for (Class paramType : paramTypes) {
        System.out.print(" " + paramType.getName());
    }
    System.out.println();
}
```

# Интроспекция vs Рефлексия

- Интроспекция
  - runtime
  - Исследовать объекты
  - Узнать тип
  - Определить наличие свойства
- Рефлексия
  - Обобщение интроспекции
  - Изменять структуру и поведение в runtime
  - Изменять свойства
  - Добавлять методы

# **Функциональное программирование**

# Отсутствие состояния

Функции без побочных эффектов

!

# Нет переменных

- Нет такого понятия
- Можно оперировать значениями
- Значение можно связать с именем
  - по сути, это имя функции, которая возвращает всё время одно и то же значение
- Типы иммутабельны
  - значение можно заменить, но не модифицировать

# Нет состояния

- Функция: аргументы -> результат
- Синтаксис функции позволяет лишь описать преобразование аргументов, по которым можно получить результат
- Результат программы = результат вызова одних функций над результатами вызова других функций

# «Сылочная прозрачность» и «чистота» функций

Выражение  $e$  называется *ссылочно-прозрачным*, если для любой программы все вхождения  $e$  можно заменить на результат вычисления  $e$ , при этом видимое поведение программы не изменится.

Функцию  $f$  называют чистой функцией если:

- 1) значение  $f$  зависит только от переданных аргументов  
(не зависит от скрытого состояния)
- 2) для любого ссылочно-прозрачного выражения  $x$  также ссылочно-прозрачным является выражение  $f(x)$   
(не обладает побочными эффектами)

# Зачем нужна «ссылочная прозрачность»

- Широкая возможность для компилятора по оптимизации программы:
  - Мемоизация
  - Выделение общих подвыражений
  - Ленивые вычисления
  - Автоматическое распараллеливание

# Пришло из функциональных языков

- Лямбды
- Функции как объект
- Замыкания
- map, fold, sum, ...
- ...

**Спасибо за  
внимание**



# РАЗБОР ДЗ

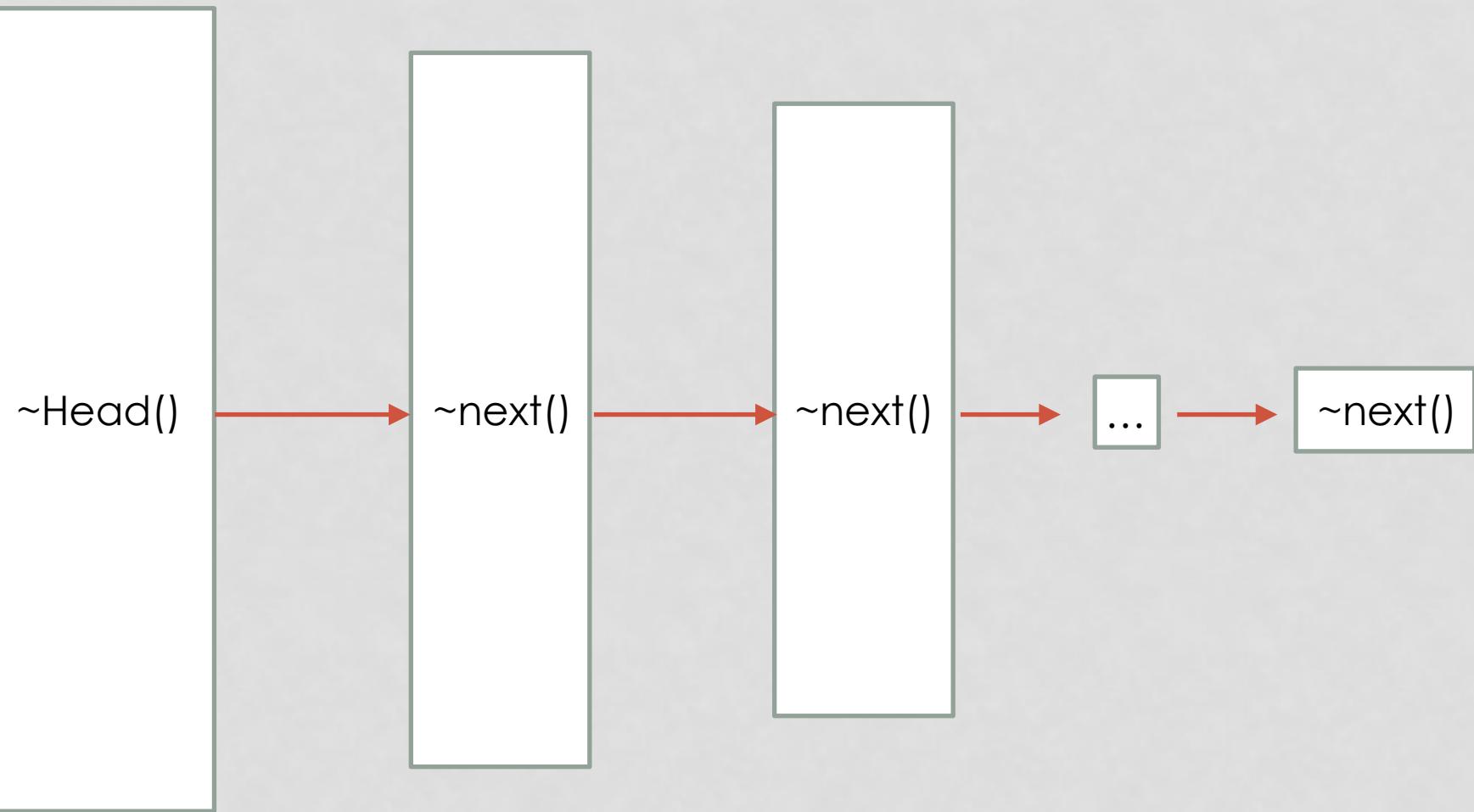
- Знай свой инструмент!!!
- Включает язык программирования!

# ПРИМЕР 1

```
struct Node {  
    unique_ptr<Node> next;  
};
```

```
struct List {  
    unique_ptr<Node> head;  
};
```

# ПРИМЕР 1



# ПРИМЕР 1

- Переполнение стека деструкторов!

## ПРИМЕР 2

```
def quick_sort(array):
    if len(array) > 1:
        pivot = array[0]
        first_array = qsort([x for x in array if x < pivot])
        second_array = qsort([x for x in array if x > pivot])
        middle_array = [x for x in array if x == pivot]
        return first_array + middle_array + second_array
    else:
        return array
```

## ПРИМЕР 2

- Каждый шаг quick sort:
  - 3 прохода по массиву
  - Сливание массивов в конце

# ЛЕКЦИЯ 6

ОБРАБОТКА ОШИБОК

**Windows**

An exception 06 has occurred at 0028:C11B3ADC in VxD DiskTSD(03) + 00001660. This was called from 0028:C11B40C8 in VxD voltrack(04) + 00000000. It may be possible to continue normally.

- \* Press any key to attempt to continue.
- \* Press CTRL+ALT+RESET to restart your computer. You will lose any unsaved information in all applications.

Press any key to continue

# ОБРАБОТКА ОШИБОК

- Баг — жаргонное слово, обычно обозначающее ошибку в программе или системе, из-за которой программа выдает неожиданное поведение и, как следствие, результат. Большинство багов возникают из-за ошибок, допущенных разработчиками программы в её исходном коде, либо в её дизайне. Также некоторые баги возникают из-за некорректной работы компилятора, вырабатывающего некорректный код. Программу, которая содержит большое число багов и/или баги, серьёзно ограничивающие её работоспособность, называют **нестабильной** или, на жаргонном языке, «глючной», «глюкнутой», «забагованной», «бажной», «баг(а)нутой»).

# ОБРАБОТКА ОШИБОК

- Ошибки
  - **ОШИБКИ ДО КОДИРОВАНИЯ**
  - Ошибки, допущенные на этапе проектирования и сбора требований
- **ОШИБКИ КОДИРОВАНИЯ**
- Ошибки, допущенные на этапе разработки программы

# ОШИБКИ ДО КОДИРОВАНИЯ

- Ошибки сбора требований
- Ошибки выявления боли пользователя
- Ошибки в составлении команды для проекта
- И т.п.



ВОТ КАКОЙ  
РАССЕЯННЫЙ

ИЛЛЮСТРАЦИИ А. М. КАНЕВСКОГО

# ОШИБКИ КОДИРОВАНИЯ

- **ДЛЯ ОПЫТНЫХ:**
  - Ошибки выбора ЯП
  - Ошибки в архитектуре ПО
- **ДЛЯ ВАС:**
  - Ошибки в алгоритмах программы
  - Ошибки пользовательского ввода
  - «Допустимые» ошибки



# «ДОПУСТИМЫЕ» ОШИБКИ

- «Допустимые» ошибки – ситуации, требующие особого поведения в программе, но при этом являющиеся допустимыми.

```
if( strlen( a ) == 0 ) {  
// Пропускаем обработку а  
} else {  
// Обработка а  
}
```

НИЧОСИ



# ОШИБКИ В АЛГОРИТМАХ

- «ОШИБКИ В АЛГОРИТМАХ» - ситуации, которые не должны происходить ни при каких условиях в программе.
- Пример:

```
int a = 5;  
if( pow( a, 2 ) < 0 ) {  
// ВОТ ТАК ШЛЯПА  
}
```

# ОШИБКИ В АЛГОРИТМАХ

- Не путайте, пожалуйста, ошибки в алгоритмах с собственным идиотизмом:

```
double step = 0.1;
double result = 0;

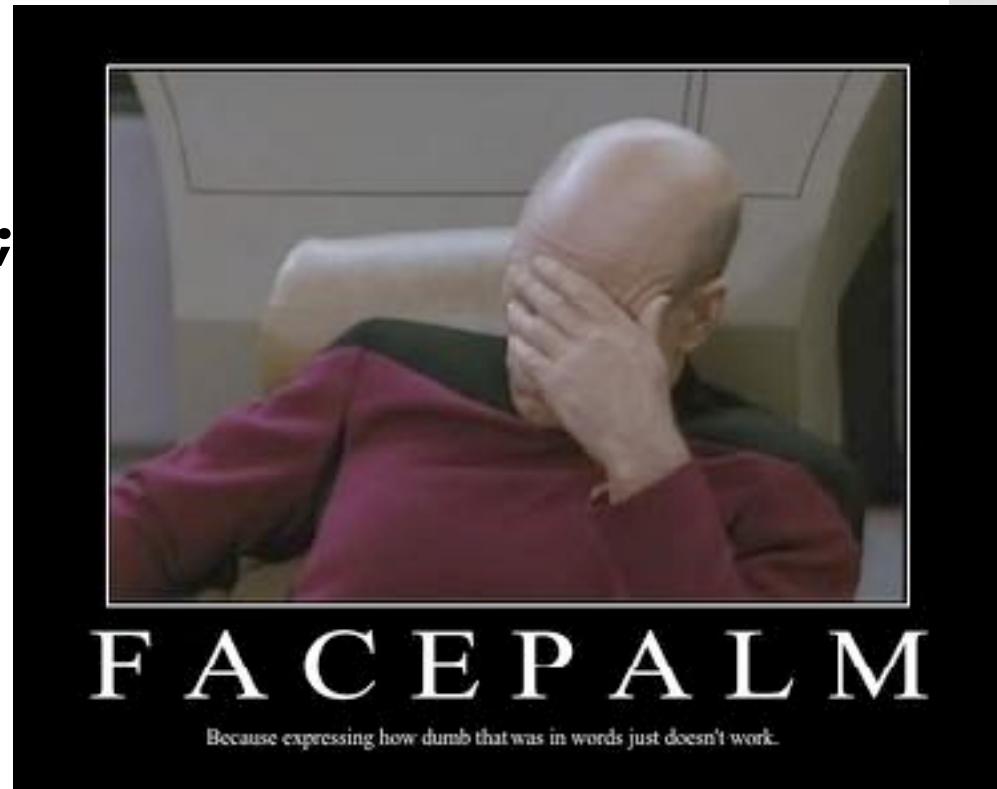
for( int i = 0; i < 10; i++ ) {
    result += step;
}

if( result != 1.0 ) {
    // Вы не поверите
}
```

# ОШИБКИ В АЛГОРИТМАХ

- Ещё пример

```
int x = INT_MAX - 2;  
int y = x + 100;  
  
if( y < 0 ) {  
    // ???  
}
```



# ОШИБКИ В АЛГОРИТМАХ

- Для проверки ошибок в алгоритмах используются замечательные инструменты:

## Assert

Приводит к аварийному  
Завершению программы  
С диагностикой ошибки

## Presume

Работает только в debug

ПОРАЗИТЕЛЬНО



# ОШИБКИ В АЛГОРИТМАХ

```
#define assert( c ) if(!(c)) { abort(); }

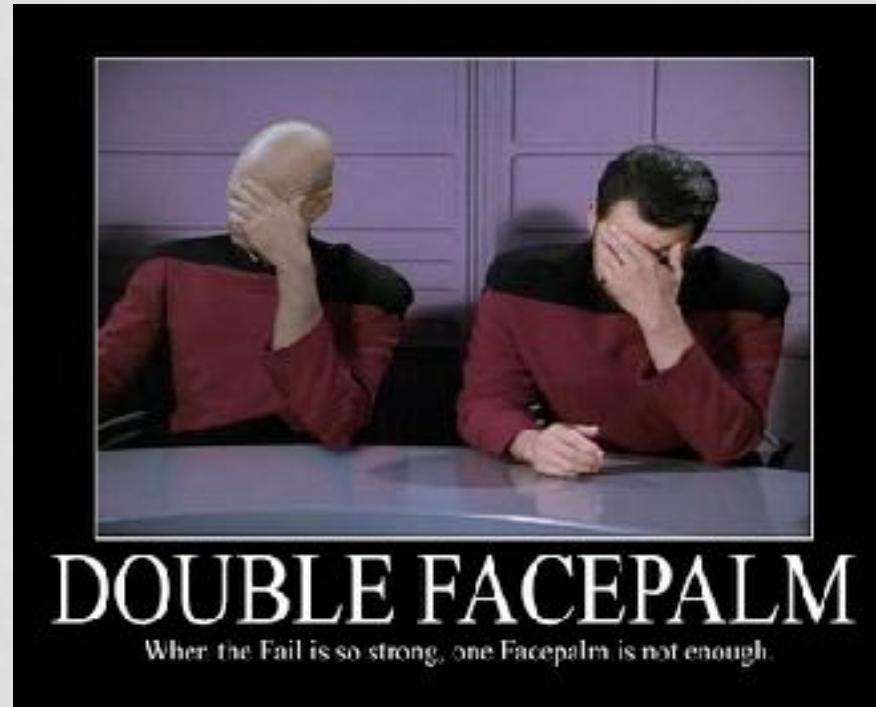
#ifndef _DEBUG
#define presume( c )
#else
#define presume( c ) if(!(c)) { abort(); }
#endif
```

Чуть более умная реализация ещё выводит файл, строку и ошибку  
р.с. #include <assert.h> решает многие проблемы

# ОШИБКИ В АЛГОРИТМАХ

- Почему presume != assert только для debug'а?

**presume(DoSomeBusinessLogic())**



# COMPILE TIME ASSERT

- ТОЛЬКО ДЛЯ ГУРУ
- Для тех, кто знает, что некоторые вещи можно проверить на этапе компиляции
- `compileTimeAssert( expr )`

```
enum TColors {  
    TC_Red,  
    TC_Blue,  
    TC_Green,  
    TC_Count  
};
```

```
compileTimeAssert( TC_Count == 3 );
```



# ОШИБКИ ПОЛЬЗОВАТЕЛЬСКОГО ВВОДА



# ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ

```
int someFoo() {  
    ...  
    if( a < 0 ) {  
        return SUPER_MEGA_ERROR;  
    }  
}
```

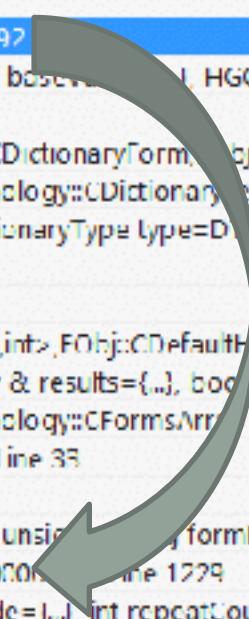
# ИСКЛЮЧЕНИЯ

- Обработка ошибок
  - функция или метод не может на своём уровне обработать ошибку
  - код ошибки
  - неизвестно, может ли вызывающая функция обработать ошибку
- Исключение
  - бросить // throw
  - «пролетает» сквозь стек вызовов
  - поймать // catch
  - «отмотать стек», корректно вызвав деструкторы объектов на стеке

# ИСКЛЮЧЕНИЯ

Call Stack

Name
MorphologyD.dll!Morphology::CMorphologicalLanguageModel::GetAClass(HCLASS handle,...) Line 592
MorphologyD.dll!Morphology::CDictionaryFormCreator::addFormsFromGClass(const CGrammarValue & baseValue, HCLASS hGC, CGrammarValue & result, bool isReflected=false) Line 101
MorphologyD.dll!Morphology::CDictionaryFormCreator::makeForms() Line 105
MorphologyD.dll!Morphology::CDictionaryFormCreator::MakeForms(TObj::CObjectArray< Morphology::CDictionaryForm> & obj) Line 105
MorphologyD.dll!CDictionaryFormMorphoAnalyser::checkNestOrGetPrefixes(const FObj::CArray< Morphology::CDictionaryForm> & searchResults, const FObj::CObjectArray< Morphology::CDictionaryForm> & prefixes) Line 105
MorphologyD.dll!CDictionaryFormMorphoAnalyser::processDictionaryForGetPrefixes(Morphology::TDictionaryType type=DictionaryType::Core1) Line 105
MorphologyD.dll!CDictionaryFormMorphoAnalyser::GetPrefixes() Line 116
MorphologyD.dll!CLanguageMorphoAnalyser::GetPrefixes() Line 92
MorphologyD.dll!CCompoundFormMorphoAnalyser::buildGLID(FObj::CHashTable< FObj::CDoubleKey< int,int>, FObj::CDefaultHash< FObj::CDoubleKey< int,int> > & hashTable, const Morphology::CFormsArray & results={...}, bool modelCheck=false) Line 105
MorphologyD.dll!CCompoundFormMorphoAnalyser::compositeRulesAnalysis(Morphology::CFormsArray & results={...}, bool modelCheck=false) Line 105
MorphologyD.dll!CCompoundFormMorphoAnalyser::process(bool forCompoundLexemes=false, Morphology::CFormsArray & results={...}) Line 105
MorphologyD.dll!CCompoundFormMorphoAnalyser::Process(Morphology::CFormsArray & results={...}) Line 33
MorphologyD.dll!CLanguageMorphoAnalyser::Process(bool analyseCompoundLexemes=false) Line 78
MorphologyD.dll!Morphology::CLanguageDictionaries::AnalyseWord(const wchar_t * word=0x032ae3cc, unsigned int wordFlags=0x00000000, unsigned int formFlags=0x00000000) Line 1229
MorphoTest.exe!CTestDialog::AnalyseWord(const FObj::CUnicodeString & str={...}, int repeatCount=0x00000001) Line 1229
MorphoTest.exe!CTestDialog::callMethod(int methodId=0x00000009, const FObj::CUnicodeString & unicode={...}, int rpcCallCount=0x00000001) Line 736
MorphoTest.exe!CTestDialog::callMethod(int methodId=0x00000009) Line 736
MorphoTest.exe!CTestDialog::OnTest() Line 734
Awl.dll!AWL::CCmdTarget::DispatchCmdMsg(AWL::CCmdMsgParams & params={...}, void (void) * pfn=0x004443cb, unsigned int nSig=1, unsigned int nFwd=0) Line 214
Awl.dll!AWL::CCmdTarget::handleCommand(AWL::CCmdMsgParams & params={...}, bool isReflected=false) Line 214



# ИСКЛЮЧЕНИЯ В С++

```
try {  
    throw new A();  
} catch( B* b ) {  
    throw b;  
} catch( A* a ) {  
    throw;  
} catch( ... ) {  
    throw new C();  
}
```

ТЫСЯЧА ЧЕРТЕЙ



photon

# НЕМНОГО О ПОЛИМОРФИЗМЕ

```
class A {};
class B : public A {};

try {
    try {
        throw new
            B();
    } catch( A* a ) {
        throw a;
    }
} catch( B* ) {
    std::cout << "B
catched";
}
```

```
try {
    try {
        throw new
            B();
    } catch( A* ) {
        throw;
    }
} catch( B* ) {
    std::cout << "B
catched";
}
```

# ПРОБЛЕМЫ ИСКЛЮЧЕНИЙ

- Бросать исключения - достаточно дорого (с точки зрения времени выполнения)

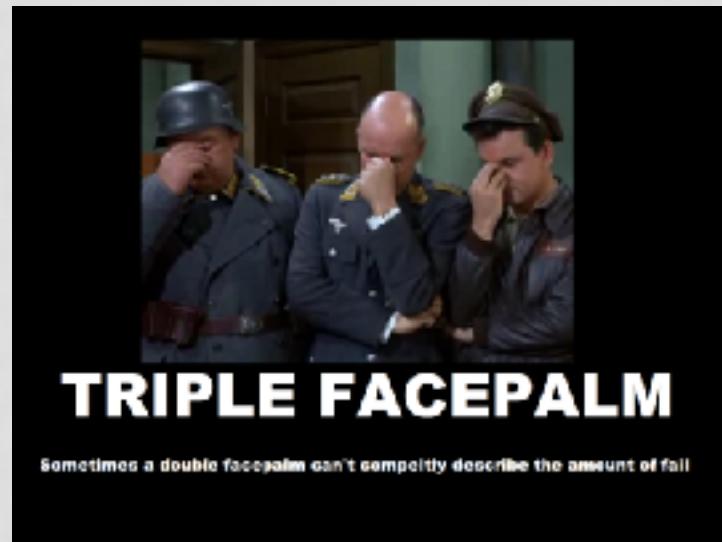
# СОВЕТЫ

- Полагайтесь на деструкторы
  - Если выделяете память явно, оберните «сырой» указатель в «умный»
- Не допускайте утечки ресурсов в конструкторе
- Не позволяйте исключениям вылетать из деструкторов
  - не до конца разрушенный объект
  - деструктор вызван во время «отмотки» стека
- Не переоценивайте спецификацию исключений в C++

# ИТОГО

- Допустимая ошибка – if
- Ошибка в алгоритме – assert, presume
- Ошибка в пользовательском вводе – исключения

**И, пожалуйста, думайте об обработке ошибок!**



СПАСИБО ЗА ВНИМАНИЕ





# УПРАВЛЕНИЕ РЕСУРСАМИ

Промышленное программирование  
лекция 5

# PECYPC

- lock
- unlock

# ПРИМЕРЫ

- Память (в языках без GC)
- Mutex, semaphore...
- Файлы
- Состояния внутри программы
- ...

# ОБЫЧНО

```
sr = SomeResource()
```

```
sr.lock()
```

```
// Logic here
```

```
sr.unlock()
```

# ОБЫЧНО

```
sr = SomeResource()
```

```
sr.lock()
```

```
// Logic here      raise SomeException()
```

```
sr.unlock()
```

# ОБЫЧНО

```
sr = SomeResource()
```

```
sr.lock()
```

```
// Logic here      raise SomeException()
```

```
sr.unlock()
```

# УТЕЧКА РЕСУРСОВ

- Незакрытые файлы
- Утекшая память
- Нарушение инвариантов программы

# ОПЕРАЦИОННАЯ СИСТЕМА

- При аварийном завершении:
  - Закроет файлы
  - Освободит тутех
  - Освободит память

# ОДНО НО

- Но если аварийного завершения не произойдет?

# TRY...FINALLY

```
sr = SomeResource()
```

```
sr.lock()
```

```
try:
```

```
# Some Logic
```

```
finally:
```

```
sr.unlock()
```

# FINALLY

- Гарантирует выполнение «финализирующего» кода в случае возникновения exception

# ПРОБЛЕМЫ FINALLY

- Постоянно писать finally - это утомительно
- Можно забыть закрыть ресурс (например, добавили еще один)
- Не везде есть (C++)

# ПРОБЛЕМЫ FINALLY

- Некоторые языки предлагают удобные альтернативы для этих целей

# C++

- RAII – Resource Acquisition Is Initialization
- Мы создаем стековый объект, который при разрушении освободит ресурс (память)

# C++

```
class CSmartPtr {  
public:  
    CSmartPtr(int *x) : data(x) {};  
  
    ~CSmartPtr() { delete data; }  
  
    int operator*() { return *data; }  
  
    int* operator->() { return data; }  
  
private:  
    int *data;  
};
```

# C++

- НЕ НАДО ИЗОБРЕТАТЬ ВЕЛОСИПЕД!
- unique\_ptr – классический умный указатель

# PYTHON

- Менеджер контекста

# PYTHON

with open('l.txt') as f:

```
# ...
```

# PYTHON

```
class FileContext(object):

    def __init__(self, filename):

        # Open File

    def __enter__(self):

        return self

    def __exit__(self, exc_type, exc_value, exc_traceback):

        # Close file
```

# GO

```
func main() {  
    f := createFile("/tmp/defer.txt")  
    defer closeFile(f)  
    writeFile(f)  
}
```

# C#

// Context - наследуется от IDisposable

```
using(new Context()) {
```

```
#SomeLogic
```

```
}
```

# ЦЕЛЬ

- Цель таких ухищрений - чтобы вы не забыли освободить ресурсы

# ВЛАДЕНИЕ ОБЪЕКТОМ

- Когда освобождать ресурс?
- Когда он больше никому не нужен

# ВЛАДЕЛЬЦЫ

- Один владелец
- Много владельцев

# ОДИН ВЛАДЕЛЕЦ

- unique\_ptr
- и тд и тп

# МНОГО ВЛАДЕЛЬЦЕВ

- Счетчик ссылок
- shared\_ptr
- промежуточные объекты

СПАСИБО ЗА  
ВНИМАНИЕ

вопросы?

