Разработчик C++

Принципы проектирования ПО

otus.ru

Проверить, идет ли запись

Меня хорошо видно && слышно?

Ставим “+”, если все хорошо
“-”, если есть проблемы

Тема вебинара

Принципы проектирования ПО

Курс: C++ Developer. Professional

Пальчуковский Евгений

Разработчик ПО

Развиваю технологии финансовых услуг с помощью C++

Telegram: @palchukovsky
Email: eugene@palchukovsky.com

Правила вебинара

Активно
участвуем

Off-topic обсуждаем
в учебной группе в
Telegram

Задаем вопрос
в чат или голосом

Вопросы вижу в чате,
могу ответить не сразу

Условные обозначения

Индивидуально

Время, необходимое
на активность

Пишем в чат

Говорим голосом

Документ

Ответьте себе или
задайте вопрос

C++ Core Guidelines

A: Architectural ideas

https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#a-architectural-ideas

C++ Core Guidelines

A: Architectural ideas

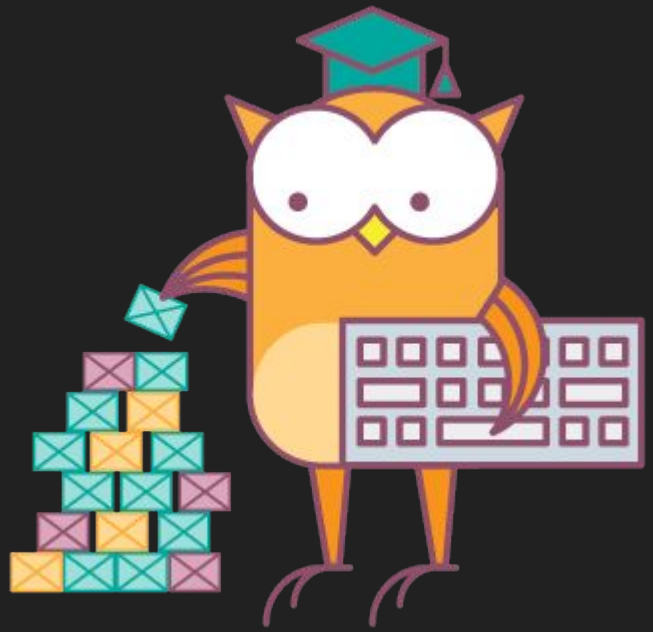
https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#a-architectural-ideas

A.1: Separate stable code from less stable code

Isolating less stable code facilitates its unit testing, interface improvement, refactoring, and eventual deprecation.

Принципы проектирования ПО

- **DRY**: повторениям — нет!
- **SoC:** ответственность
- **YAGNI**: ненужное — не нужно
- **KISS**: простое проще сложного
- **SOLID**



Зачем?

Каждый middle-специалист должен самостоятельно уметь

принимать архитектурные решения!

Зачем?

Пример задачи для junior-а:

На главную страницу сайта
добавить зелёную кнопку
авторизации с двумя полями ввода
«Логин» и «Пароль», а также форму
восстановления пароля, доступную
по ссылке «Забыли пароль».

Зачем?

Пример задачи для junior-а:

На главную страницу сайта добавить зелёную кнопку авторизации с двумя полями ввода «Логин» и «Пароль», а также форму восстановления пароля, доступную по ссылке «Забыли пароль».

Что известно из постановки задачи?

- Где
- Есть кнопка
- Кнопка - зеленая
- Какие нужны поля
- Функционал восстановления доступа
- Имена полей и форм

Зачем?

Пример задачи для middle-а:

Реализовать схему авторизации пользователя.

Зачем?

Пример задачи для middle-а:

Реализовать схему авторизации пользователя.

Что известно из постановки задачи?

Что это?

Архитектура ПО:

совокупность важнейших решений об организации программной системы: структурных элементах, их взаимодействии.

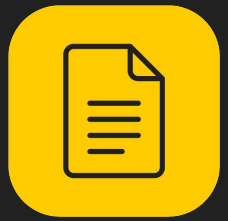
Что это?

Архитектура ПО:

совокупность важнейших решений об организации программной системы: структурных элементах, их взаимодействии.

На практике: способ организации кода

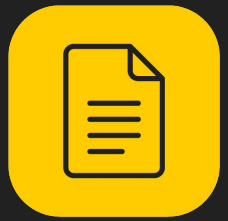
Что это?



Хорошая архитектура:

это **всего лишь** способ организации исходников (файлов, классов, функций, переменных, и т.д.), при котором стоимость разработки и поддержки (внесения изменений) **минимальна.**

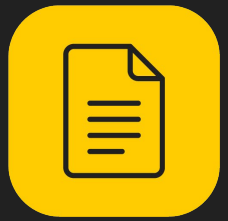
Что это?



Хорошая архитектура:

- эффективность

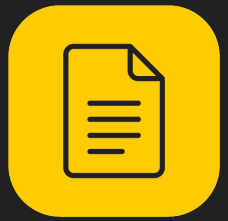
Что это?



Хорошая архитектура:

- эффективность
- тестируемость

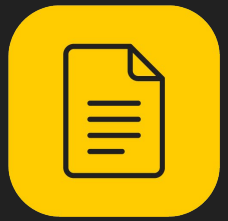
Что это?



Хорошая архитектура:

- эффективность
- тестируемость
- сопровождаемость

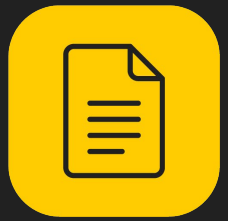
Что это?



Хорошая архитектура:

- эффективность
- тестируемость
- сопровождаемость
- гибкость

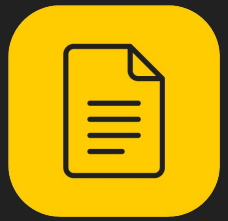
Что это?



Хорошая архитектура:

- эффективность
- тестируемость
- сопровождаемость
- гибкость
- расширяемость

Что это?



Хорошая архитектура:

- эффективность
- тестируемость
- сопровождаемость
- гибкость
- расширяемость
- переиспользование

Принцип DRY

don’t repeat yourself - не повторяйся

- Устойчивость к изменениям
- Устойчивость к багам
- Тестируемость
- Подходящие абстракции

Я по два раза не повторяю,
по два раза не повторяю!

Не только “не copy-paste кода”, но и идею

Принцип DRY

don’t repeat yourself - не повторяйся

Не только код!

Клиент

Id _кл	Фамилия	Имя	Отчество
15	Иванов	Иван	Иванович
16	Петров	Петр	Петрович
17	Николаев	Николай	Николаевич

Товар

Id_тов	Название
1	Шкаф
2	Стул
3	Стол

Заказ

Id_зак	Клиент	Товар	Дата	Количество
1	15	1	15.09.2003	1
2	17	1	17.09.2003	2
3	15	2	20.09.2003	12

Я по два раза не повторяю,
по два раза не повторяю!

Принцип DRY

don’t repeat yourself - не повторяйся

- DRY <-> WET
- "write everything twice“
- "we enjoy typing“
- "waste everyone's time“

Я по два раза не повторяю,
по два раза не повторяю!

Принцип SoC
Separation of Concerns

Разделение ответственностей

- Разделяй и властвуй
- Так проще
- Можно делегировать
- Расширяемость и устойчивость к изменениям
- Отказоустойчивость
- Тестируемость

Принцип SoC
Separation of Concerns

Примеры

- MVC –model view controller
- SOA –service-oriented architecture
- Микросервисы
- CI/CD

Принцип YAGNI

You aren’t gonna need it

Вам это не понадобится, если:

- Но это же классная штука!
- Но это точно пригодится!
- Я уверен, это будет полезно!
- Смотри, как я умею! (CVDD)
- Ну раз уж я все равно сделал…

Принцип YAGNI

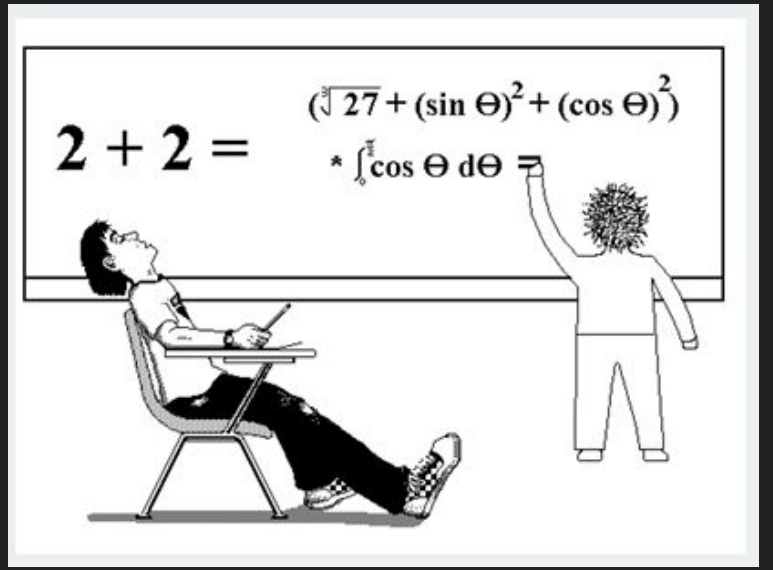
You aren’t gonna need it

Вам это не понадобится, если:

- Тратится время
- Усложняется код
- Снежный ком из супер-фич
- Ненужное становится мёртвым кодом
- … который становится хламом, когда оказывается нужным

Принцип KISS
Keep it simple, stupid

Чем проще, тем лучше.



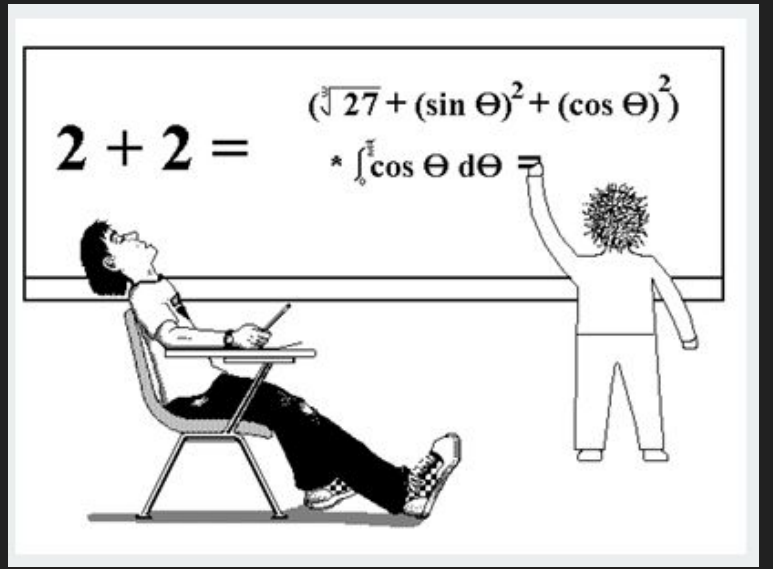
M16 - Сотни движущихся деталей

скрепленных десятками болтов и винтов.

Винтовка Мосина - три движущихся детали, два винта.

Принцип KISS
Keep it simple, stupid

Чем проще, тем лучше.



- Критерий сложности/простоты?
- А судьи кто?
- А как же творчество?
- Мне нужен повод для гордости.

Принцип LoD
Law of Demeter

Говори, что делать, а не спрашивай

- Класс – объединение данных и операций с ними
- Каждый раз, когда Вы получаете доступ к внутренним данным класса, в мире умирает ООП-котёнок
- Инкапсуляция данных и логики их обработки

Принцип TDA
Tell-Don’t-Ask

Law of Demeter

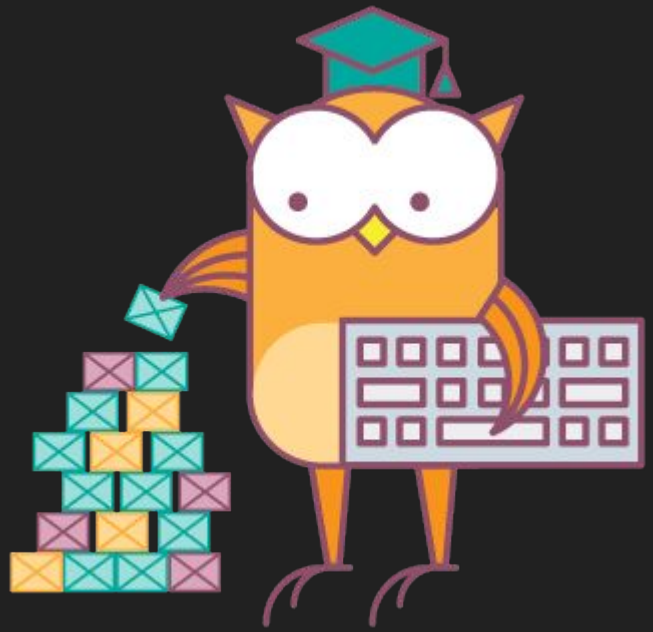
- Модули связаны только с непосредственными «соседями»
- Модули работают только со связанными модулями
- Модули обращаются только к непосредственным «соседям»

❌ a.b.Method()

a.MethodWithB() ✅

Принципы проектирования ПО

- **DRY**: повторениям — нет!
- **SoC:** ответственность
- **YAGNI**: ненужное — не нужно
- **KISS**: простое проще сложного
- **SOLID**



SOLID

акроним Майкла Фэзерса
для принципов Роберта Мартина...

- **S - Single Responsibility** - единственность ответственности
- **O - Open Closed** - открыт для расширения, закрыт для модификации
- **L - Liskov Substitution** - принципы подстановки Лисков
- **I - Interface Segregation** разделение интерфейса
- **D - Dependency Inversion** - инверсия зависимостей

SOLID

Single Responsibility - единственность ответственности

Принцип SoC

Separation of Concerns

Разделение ответственностей

- Разделяй и властвуй
- Так проще
- Можно делегировать
- Расширяемость и устойчивость к изменениям
- Отказоустойчивость
- Тестируемость

OTUS | ОНЛАЙН ОБРАЗОВАНИЕ

SOLID

O - Open Closed

открыт для расширения, закрыт для модификации

SOLID

O - Open Closed - открыт для расширения, закрыт для модификации

```
01.
02. void doSomething(const MyCustomView& view)
03. {
04.     view.showAllData();
05. }
06.
07. void doSomething(const IView& view)
08. {
09.     view.showAllData();
10. }
11.
```

SOLID

O - Open Closed - открыт для расширения, закрыт для модификации

```
01.
02. void doSomething(const MyCustomView& view)
03. {
04.     view.showAllData();
05. }
06.
07. void doSomething(const IView& view)
08. {
09.     view.showAllData();
10. }
11.
```

Прибив гвоздями конкретный view, спровоцировать модификацию

Используем через интерфейс IView.

Бесконечно расширяем, используя полиморфизм, расширяя IView.

doSomthing - продолжает работать.

SOLID

L - Liskov Substitution - принципы подстановки Барбары Лисков

```
01.
02. class Square : public Rectangle
03. {
04.     void setWidth(int value) override;
05. };
06.
07. void func(const Rectangle& rect)
08. {
09.     view.setWidth(10);
10. }
11.
```

Код, который использует **базовый** тип, должен иметь возможность **работать** с **наследниками** этого типа, **не зная об этом**.

SOLID

L - Liskov Substitution - принципы подстановки Барбары Лисков

```
01.
02. class Square : public Rectangle
03. {
04.     void setWidth(int value) override;
05. };
06.
07. void func(const Rectangle& rect)
08. {
09.     view.setWidth(10);
10. }
11.
```

Квадрат от прямоугольника? o_O

Какой будет результат?

SOLID

I - Interface Segregation разделение интерфейса

Лучше **несколько разных** интерфейсов, чем **один универсальный** с сотней методов на все случаи жизни (YAGNI)

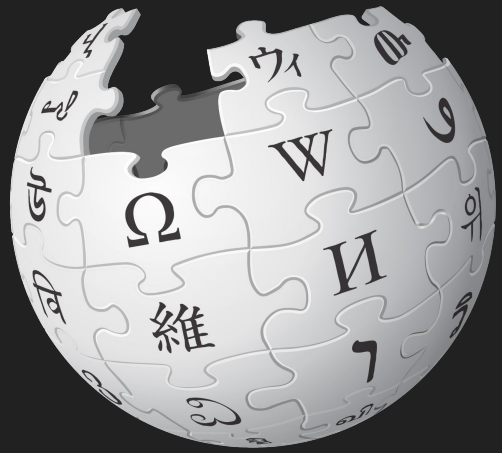
SOLID

D - Dependency Inversion - инверсия зависимостей

Формулировка:

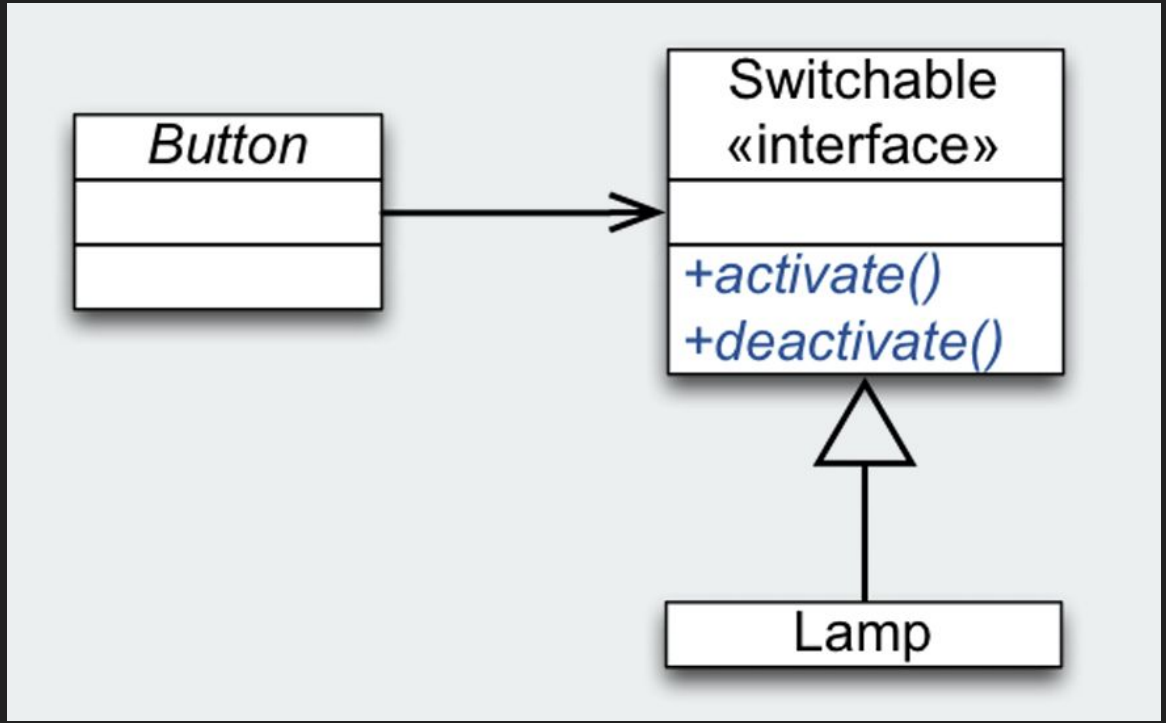
- Модули верхних уровней не должны зависеть от модулей нижних уровней.
  Оба типа модулей должны зависеть от абстракций.

- Абстракции не должны зависеть от деталей.
  Детали должны зависеть от абстракций.



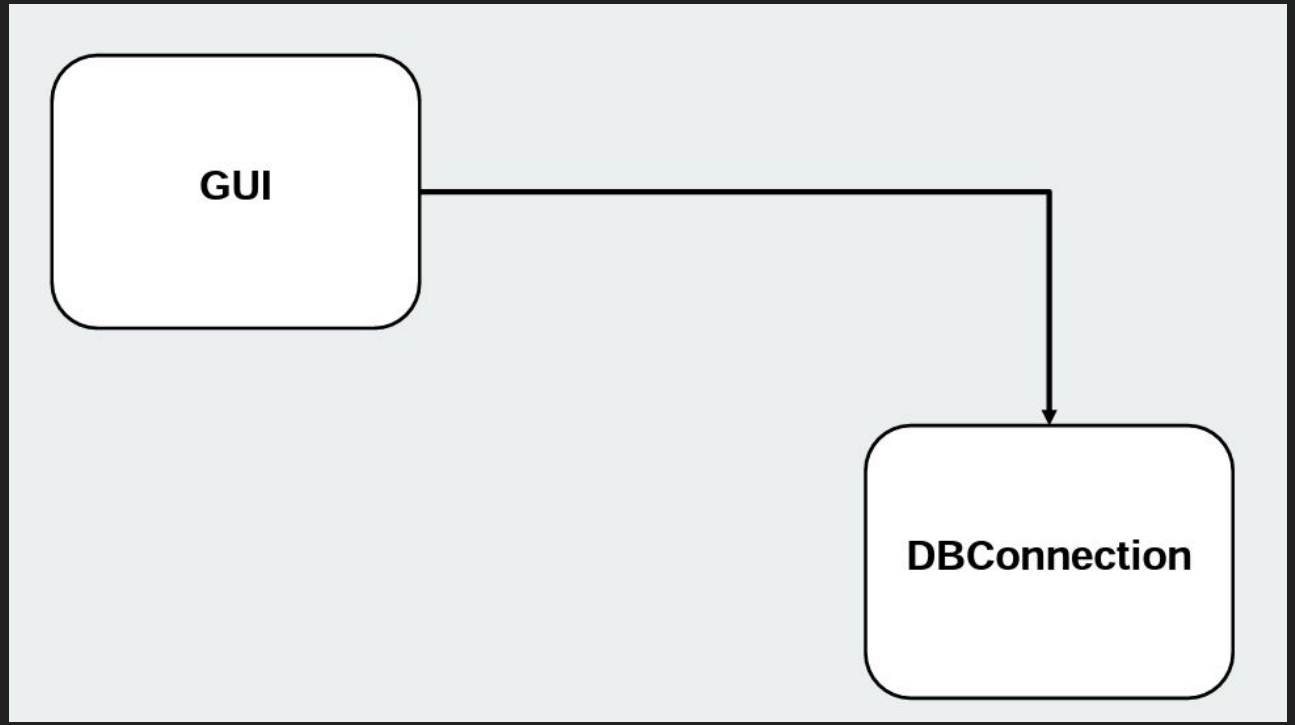
SOLID

D - Dependency Inversion - инверсия зависимостей



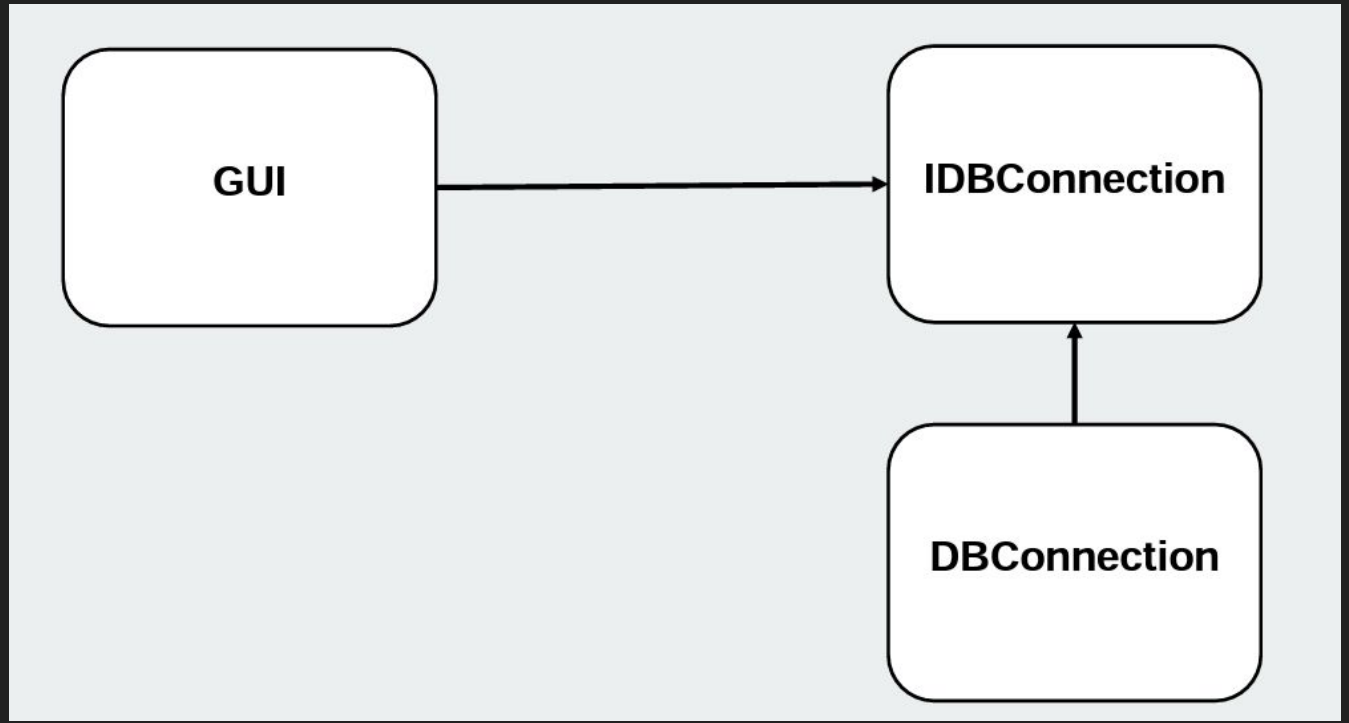
SOLID

D - Dependency Inversion - инверсия зависимостей



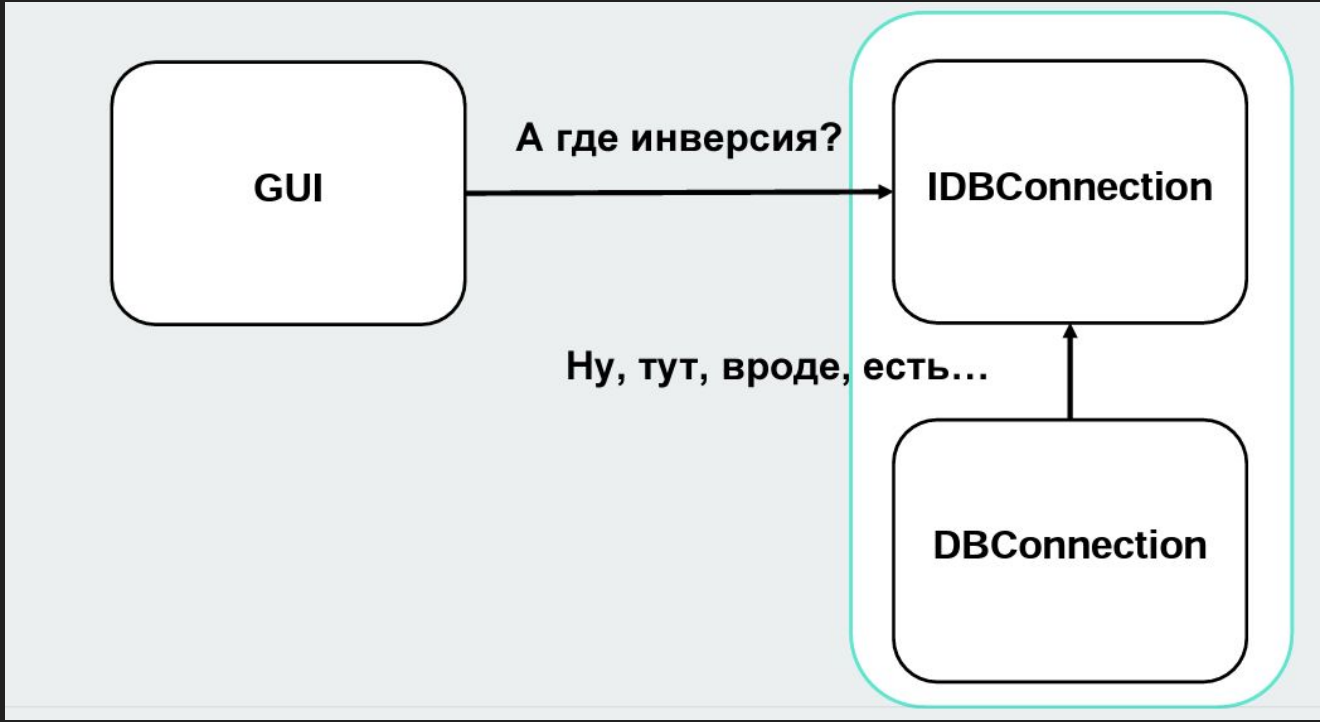
SOLID

D - Dependency Inversion - инверсия зависимостей



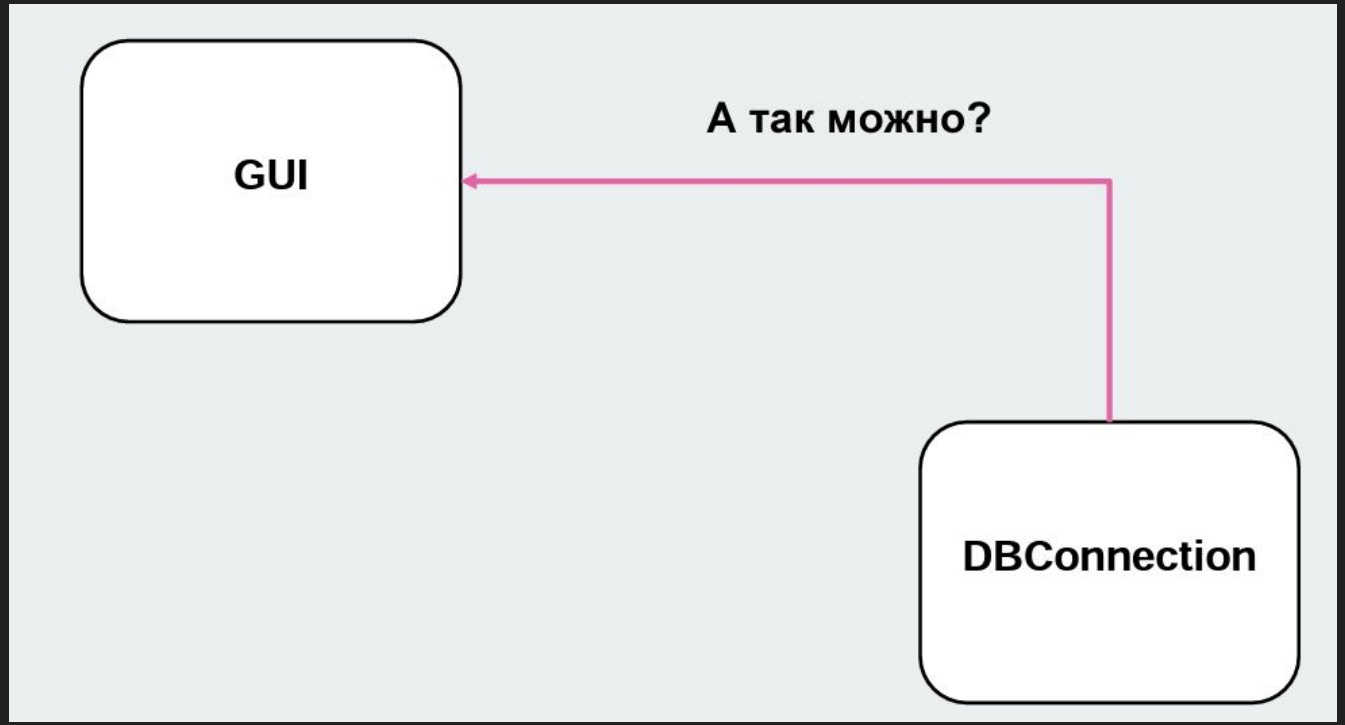
SOLID

D - Dependency Inversion - инверсия зависимостей



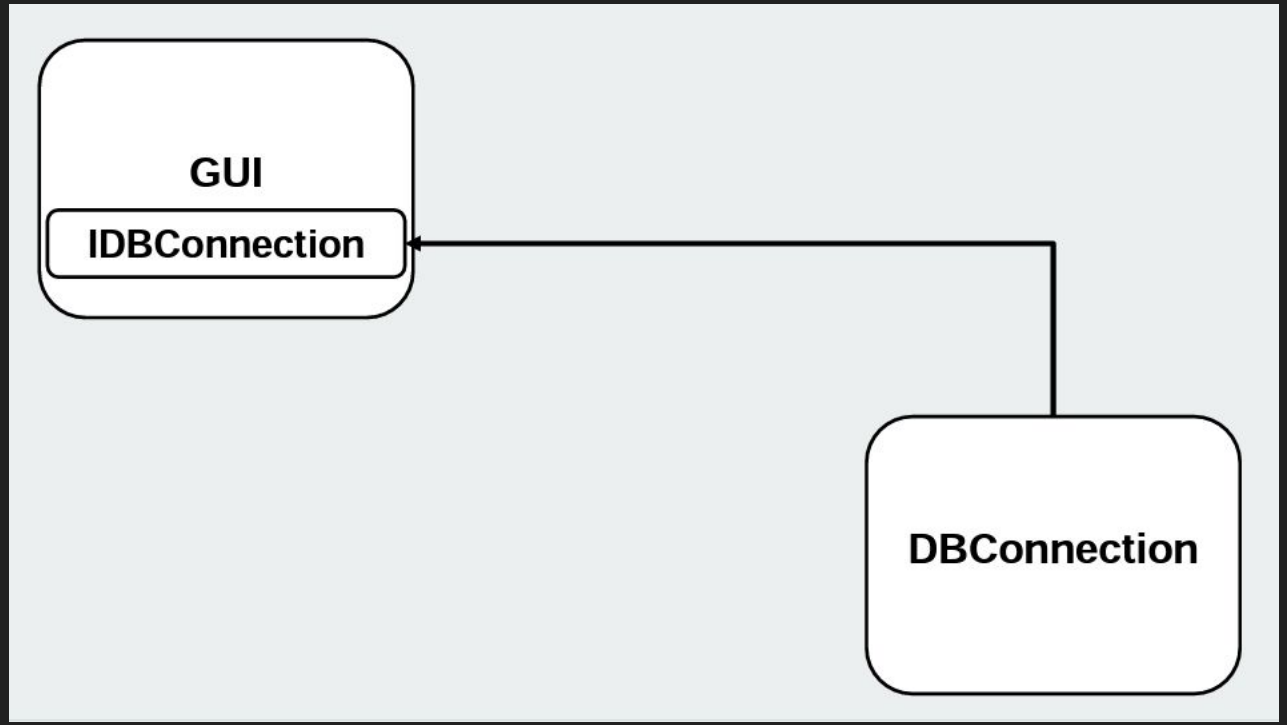
SOLID

D - Dependency Inversion - инверсия зависимостей



SOLID

D - Dependency Inversion - инверсия зависимостей

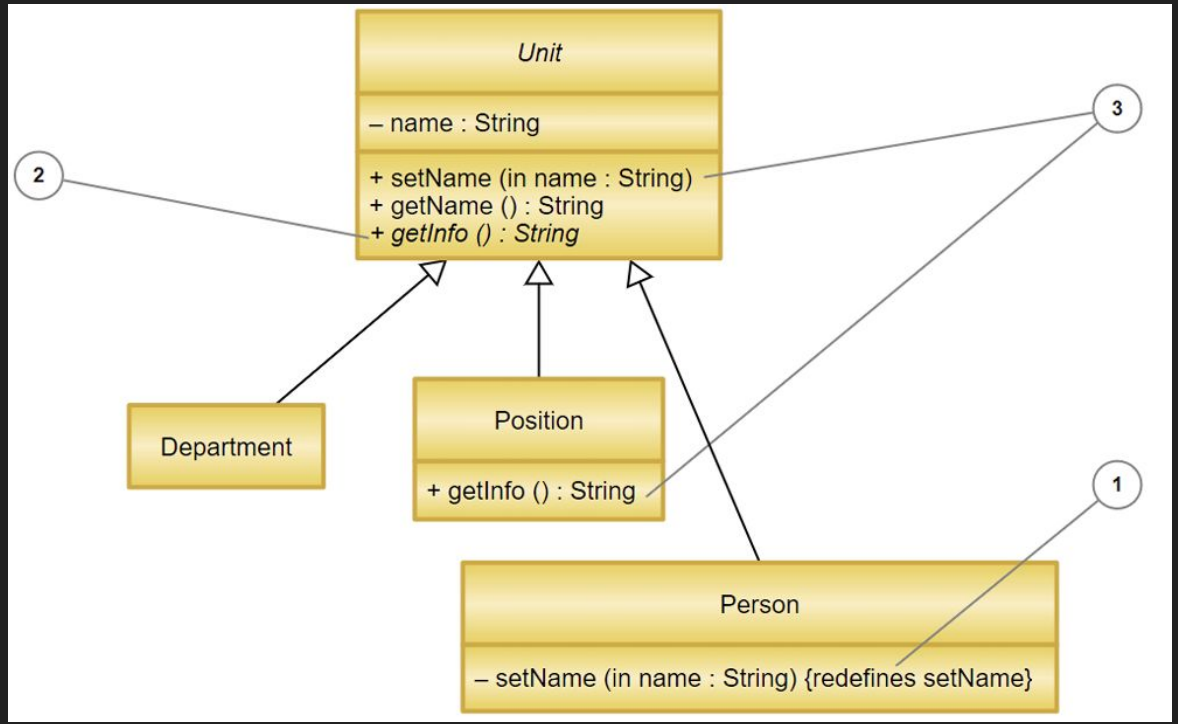


UML

Unified Modelling Language
Унифицированный язык моделирования

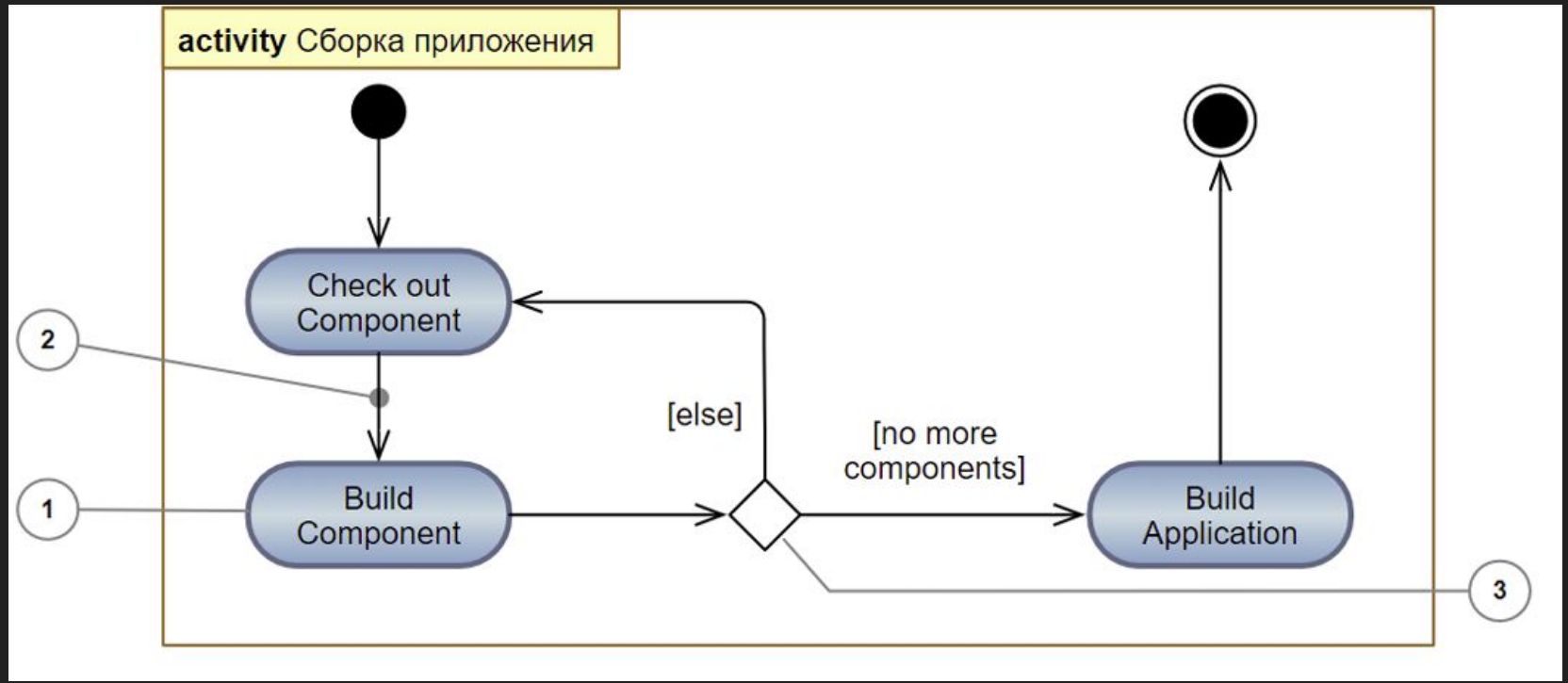
- Унифицированный
- Кодогенерация – программисты не нужны!
- А нам-то зачем?
- Графическое документирование (doxygen)
- Пояснения к ТЗ
- Поболтать в курилке

UML
Диаграмма классов



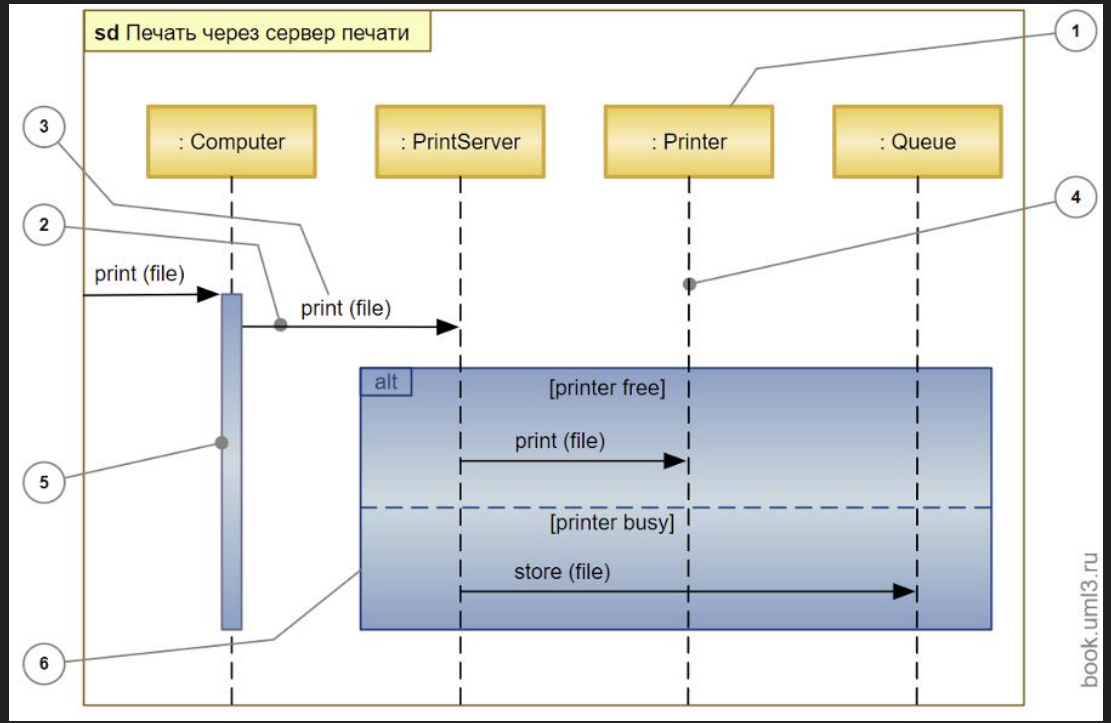
UML

Диаграмма активности



UML

Диаграмма последовательностей



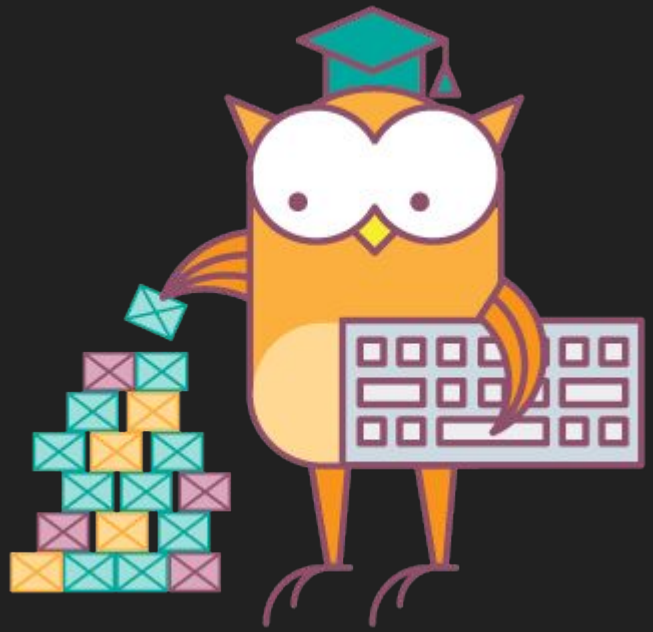
UML

Полезные материалы

- http://plantuml.com - рисовалка UML
- https://www.ozon.ru/context/detail/id/3905587/ - ГрадиБуч
- https://www.doxygen.nl/index.html - doxygen
- http://pandoc.org/ - конвертер документации

Принципы проектирования ПО

- **DRY**: повторениям — нет!
- **SoC:** ответственность
- **YAGNI**: ненужное — не нужно
- **KISS**: простое проще сложного
- **SOLID**



Домашнее задание

Следующий вебинар

15 июня 2023

GRASP

Общие принципы распределения обязанностей в ПО

- Ссылка на вебинар будет в ЛК за 15 минут
- Материалы к занятию в ЛК — можно изучать
- Обязательный материал обозначен красной лентой

Вопросы?

Ставим “+”,
если вопросы есть

Ставим “–”,
если вопросов нет

Заполните, пожалуйста, опрос о занятии по ссылке в чате