

Программирование на современном C++

Объектно-ориентированное проектирование. SOLID.



Отметьтесь на портале!

- Посещение необязательное, но тем, кто пришёл, следует отмечаться на портале в начале каждого занятия
- Это позволяет нам анализировать, какие занятия были более или менее интересны студентам, и менять курс в лучшую сторону
- Также это даст возможность вам оставить обратную связь по занятию после его завершения


пн, 1 ноября	вт, 2 ноября	ср, 3 ноября	чт, 4 ноября	пт, 5 ноября	сб, 6 ноября
Нет занятий	<div>18:00 Углубленный C/C++ (w... п</div> <div>Обработка исключительных ситуаций. Шаблоны классов и методов. Обобщенное и безопасное программирование</div> <div>А. Халайджи</div> <div>18:00 Углубленный C/C++ (ML) п</div> <div>Обработка исключительных ситуаций. Шаблоны классов и методов. Обобщенное и безопасное программирование</div> <div>А. Халайджи</div>	Нет занятий	Нет занятий	Нет занятий	Нет занятий

Ссылка на Zoom РК сегодня в 18:00

Безопасность интернет-приложений (третий семестр)

Подключиться к конференции Zoom
<https://maillru.zoom.us/j/98586081818?pwd=eWxwSDdCbIhQNnNwQU5ibIFvU2dTZz09>

Идентификатор конференции: 985 8608 1818
Код доступа: 785885

 Алексей Набережный 55 минут назад

★ 0 💬 0 ↓ 0 ↑

Запись прошлой лекции (+ что почитать перед РК)

Безопасность интернет-приложений (третий семестр)

Углублённый C/C++

Лекция 5

📍 Онлайн - ML

Отметьтесь, что вы пришли на занятие. Так вы улучшите свою посещаемость и вас увидит преподаватель в своём "Журнале посещений".

Отметиться

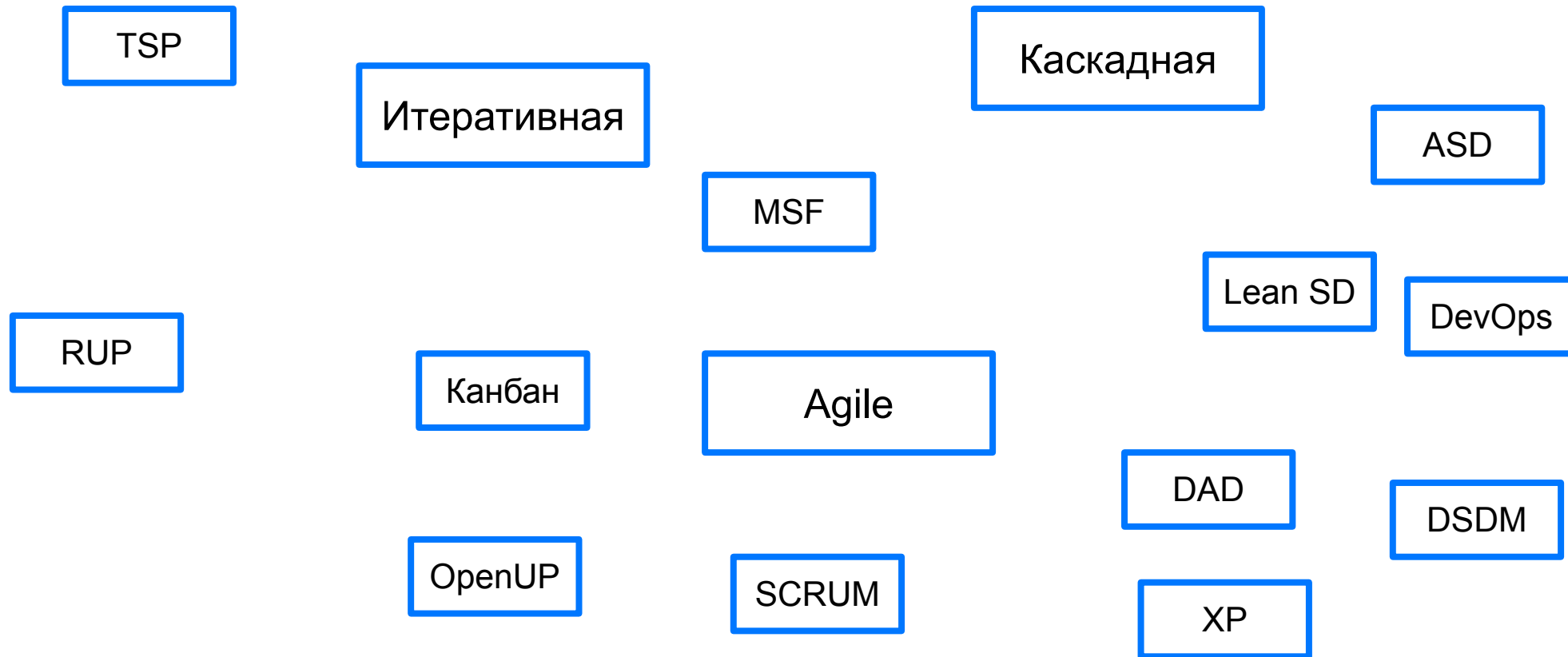
Оставьте отзыв о занятии и мы сможем улучшить учебный процесс.

Оставить отзыв

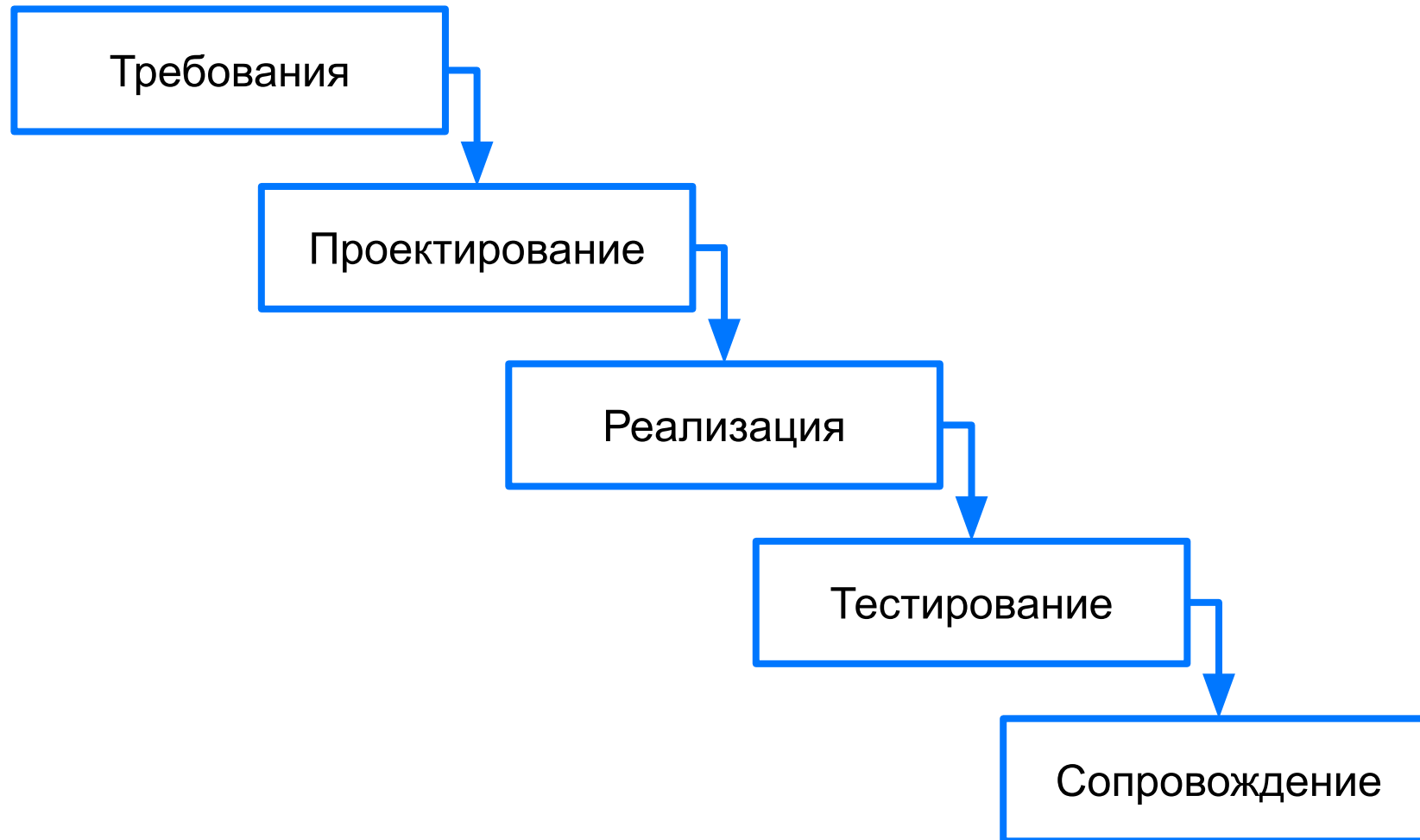
План лекции

- Методологии разработки
- ОО проектирование
- Перерыв
- SOLID

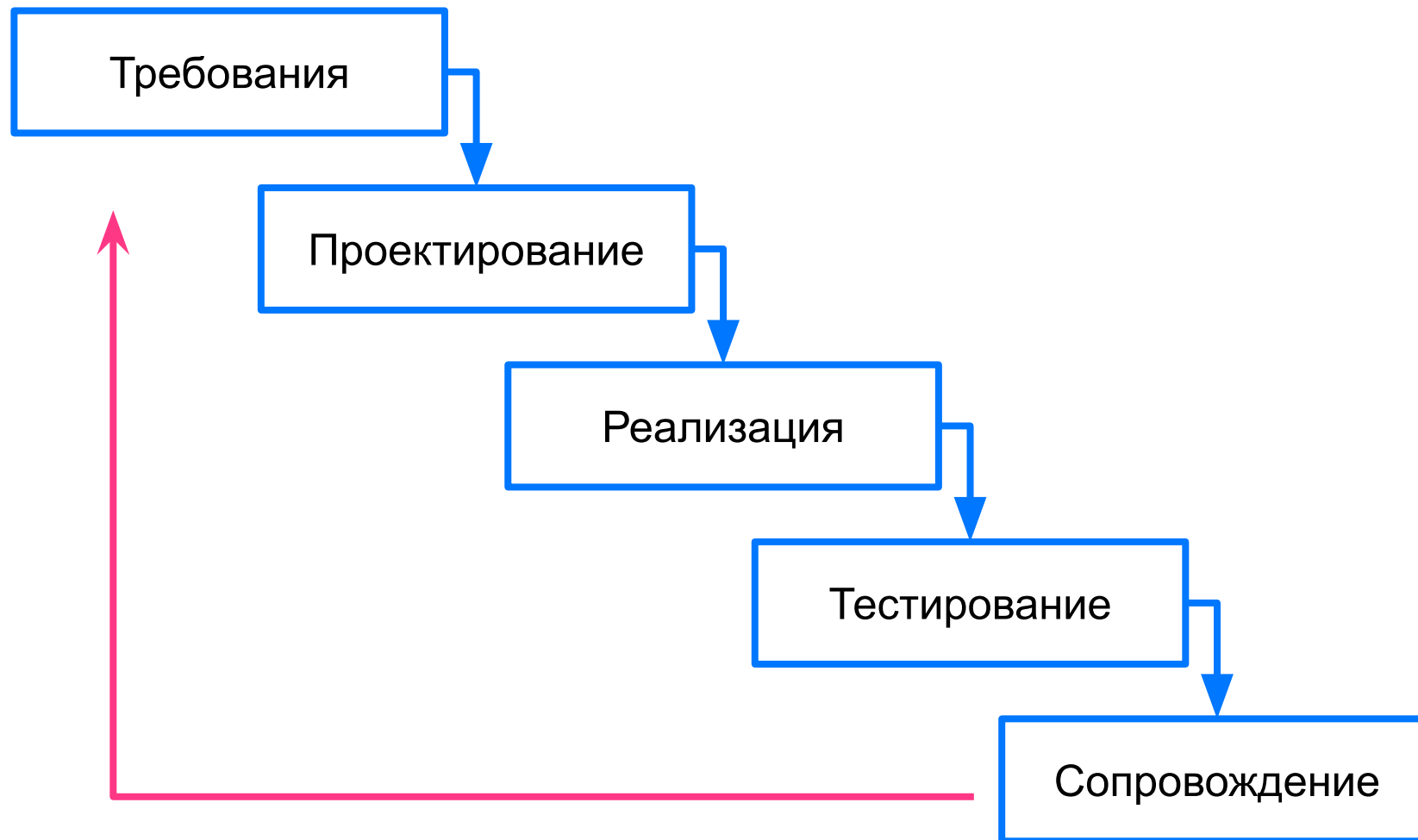
Методологии разработки ПО



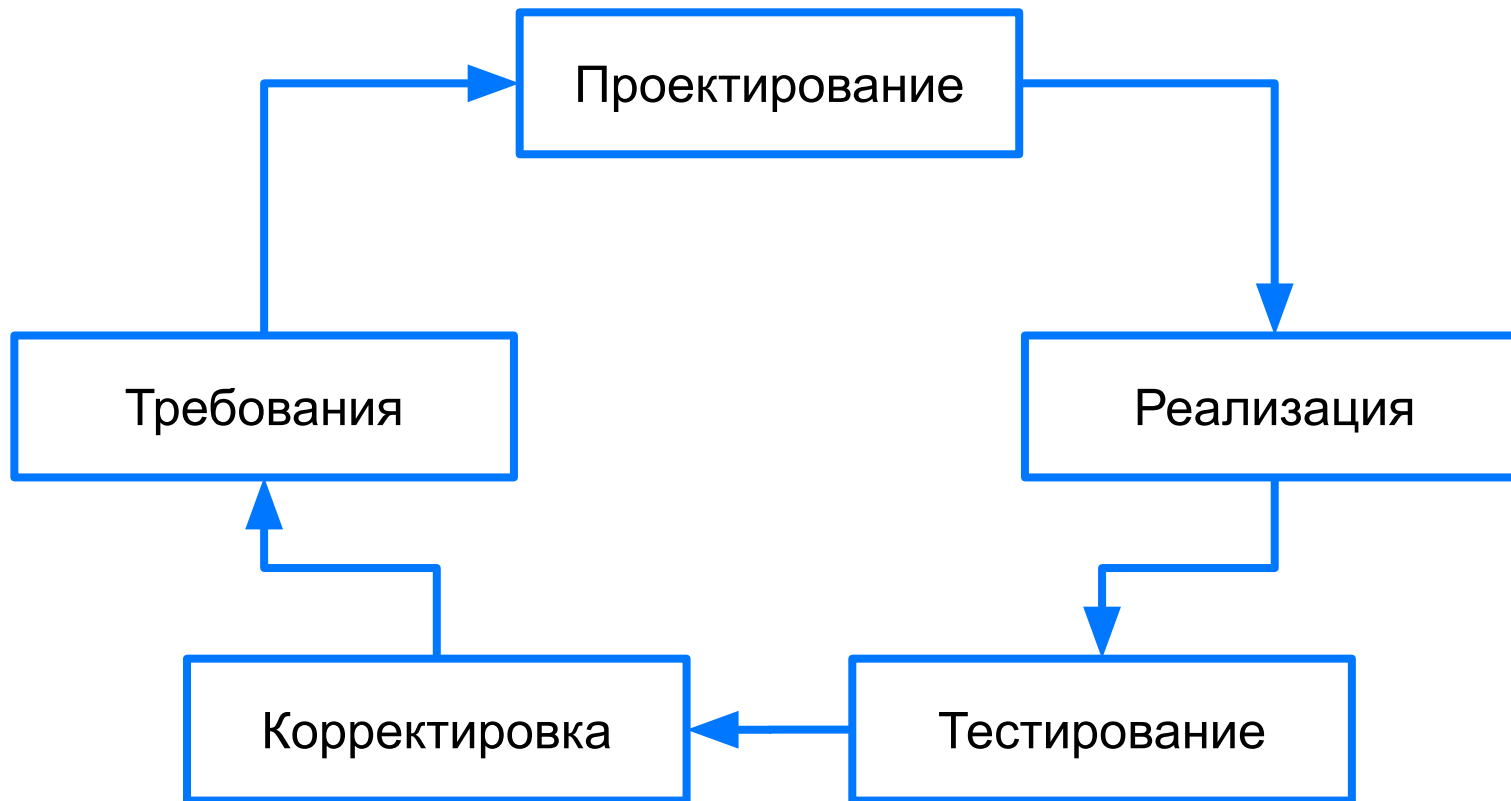
Каскадная модель



Каскадная методология



Итеративная методология



Основные этапы

- Анализ требований
- Проектирование бизнес-модели
- Реализация
- Тестирование
- Сопровождение

Основные этапы

- **Анализ требований**
- **Проектирование бизнес-модели**
- Реализация
- Тестирование
- Сопровождение

Анализ требований

- Бизнес-требования — определяют назначение ПО.
- Пользовательские требования — определяют набор пользовательских задач, которые должна решать программа.

Анализ требований

Use-case - прецедент/сценарий использования.

Повествовательные истории об использовании системы, которые используются для формулировки требований.

Анализ требований

Use-case - прецедент/сценарий использования.

Повествовательные истории об использовании системы, которые используются для формулировки требований.

Пользователь открывает сайт системы и загружает туда изображение. Система отображает пользователю похожие изображения.

Анализ требований

Use-case - прецедент/сценарий использования.

Повествовательные истории об использовании системы, которые используются для формулировки требований.

Пользователь открывает сайт системы и загружает туда изображение. Система отображает пользователю похожие изображения.

Пользователь открывает сайт системы и загружает туда изображение неподдерживаемого формата. Система отображает пользователю ошибку.

Анализ требований

User story - пользовательские истории. Краткие описания требований к системе.

Анализ требований

User story - пользовательские истории. Краткие описания требований к системе. Обычно формулируются как одно предложение

Анализ требований

User story - пользовательские истории. Краткие описания требований к системе. Обычно формулируются как одно предложение

Как пользователь я хочу иметь возможность сохранять статьи для чтения позже.

Анализ требований

User story - пользовательские истории. Краткие описания требований к системе. Обычно формулируются как одно предложение

Как пользователь я хочу иметь возможность сохранять статьи для чтения позже.

Как пользователь я хочу иметь возможность смотреть события за месяц.

Анализ требований

User story - пользовательские истории. Краткие описания требований к системе. Обычно формулируются как одно предложение

Как пользователь я хочу иметь возможность сохранять статьи для чтения позже.

Как пользователь я хочу иметь возможность смотреть события за месяц.

Как пользователь я хочу иметь возможность сохранять прогресс игры.

Анализ требований

User story - пользовательские истории. Краткие описания требований к системе. Обычно формулируются как одно предложение

Как пользователь я хочу иметь возможность сохранять статьи для чтения позже.

Как пользователь я хочу иметь возможность смотреть события за месяц.

Как пользователь я хочу иметь возможность сохранять прогресс игры.

Как администратор я хочу иметь возможность банить статьи.

UML

UML (англ. Unified Modeling Language — унифицированный язык моделирования) — язык графического описания для объектного моделирования в области разработки программного обеспечения, для моделирования бизнес-процессов, системного проектирования и отображения организационных структур.

UML Диаграмма прецедентов

Пример

Проектирование

- Моделирование предметной области
- Определение взаимодействия

Моделирование предметной области

- Пример

Определение взаимодействия

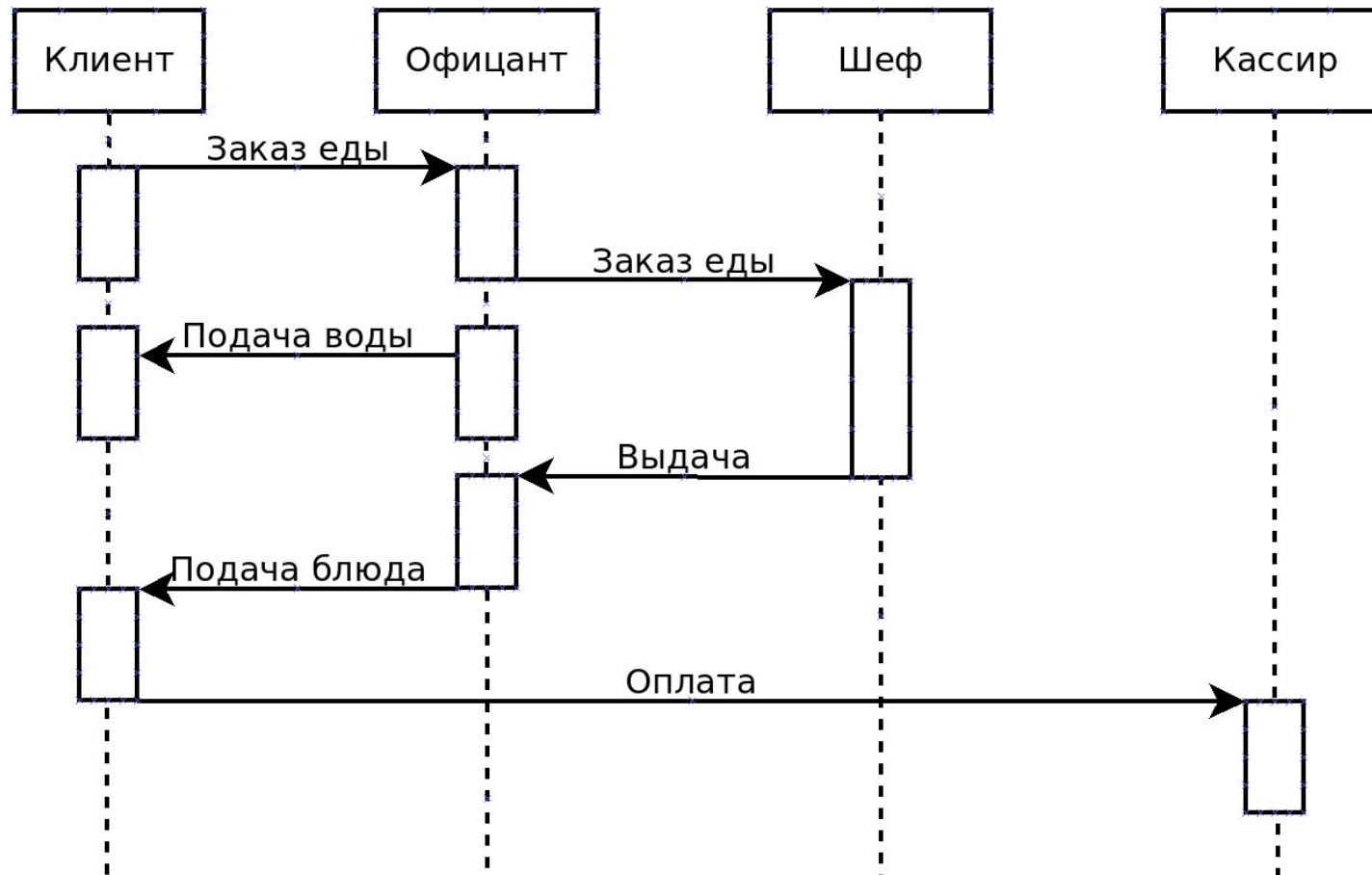
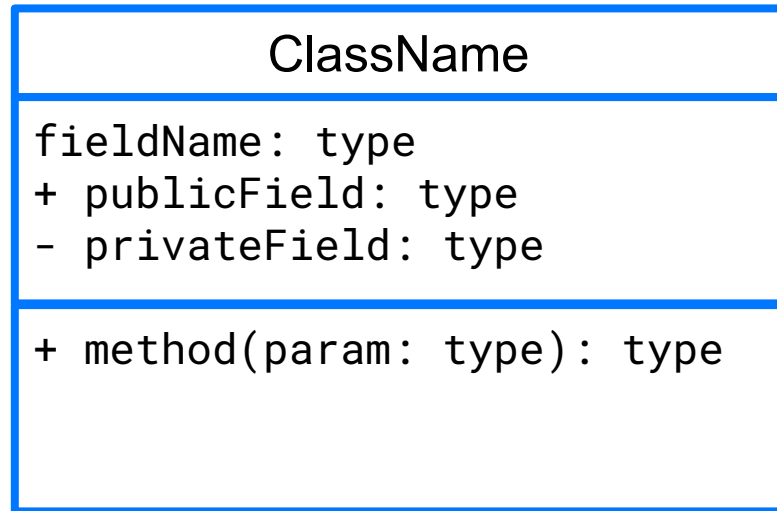
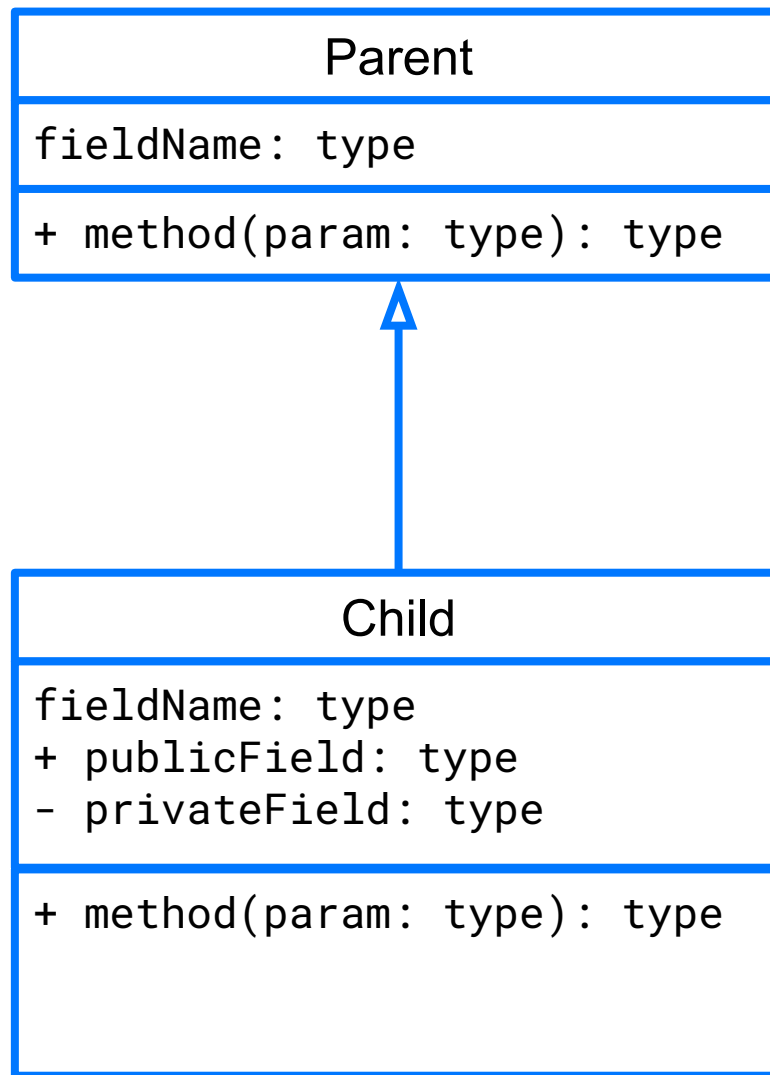


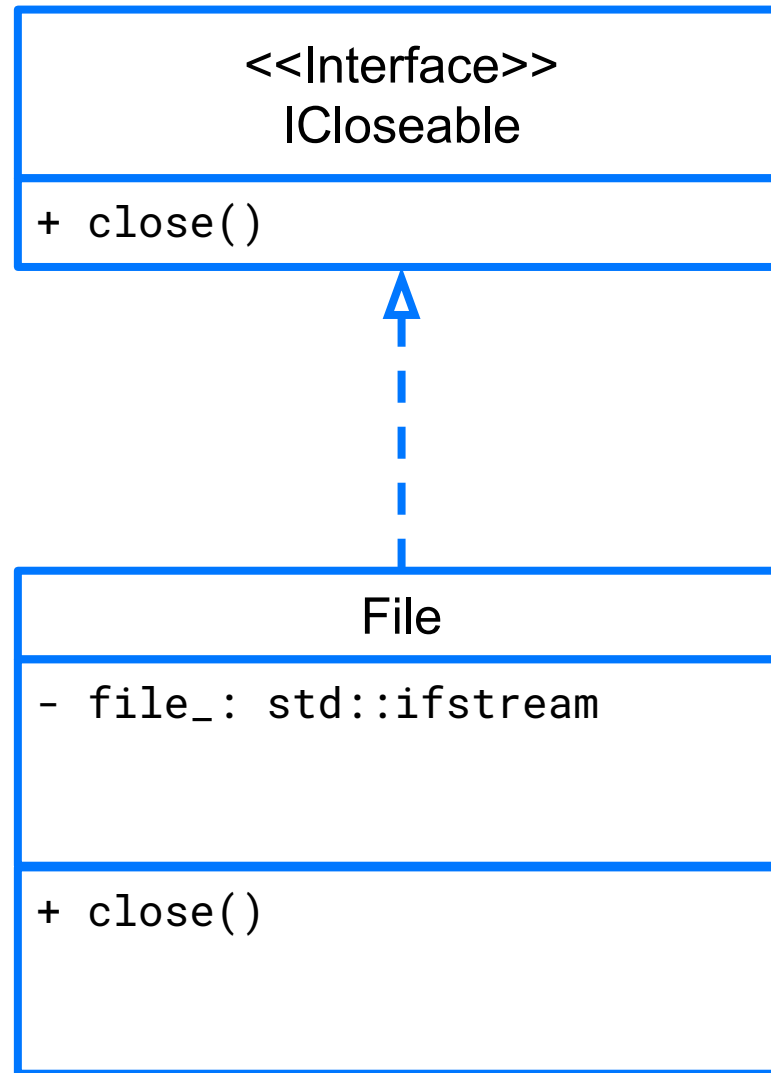
Диаграмма классов



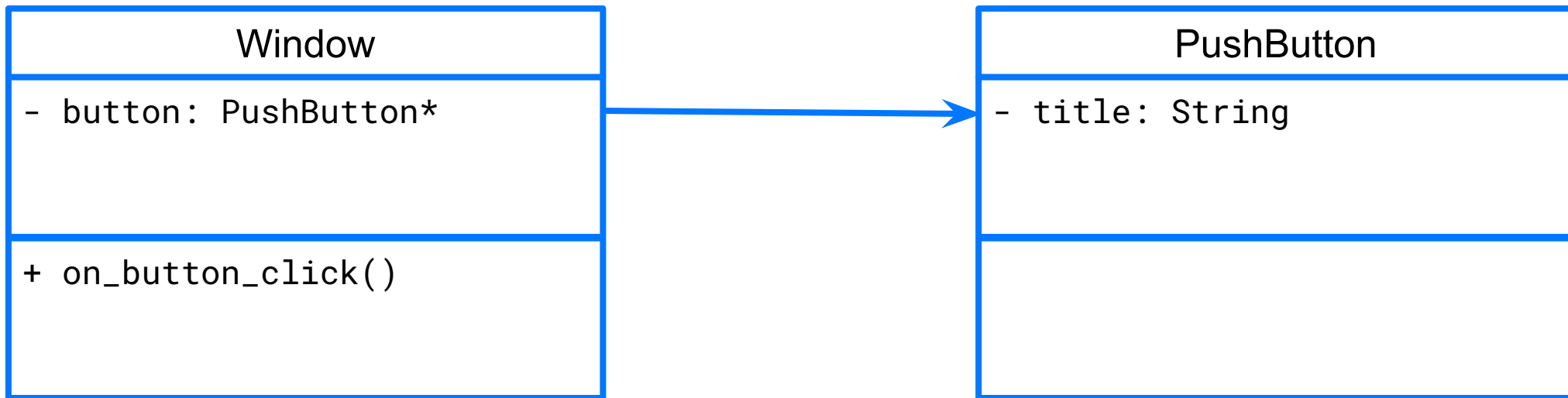
Наследование



Реализация



Ассоциация



Агрегация/Композиция

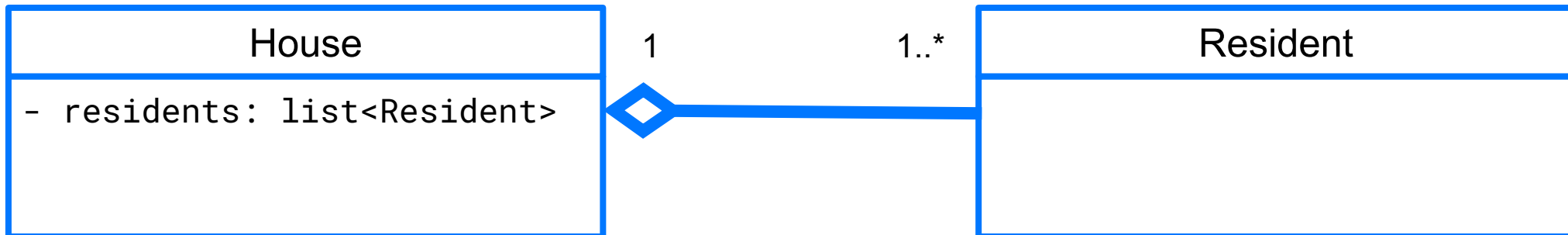
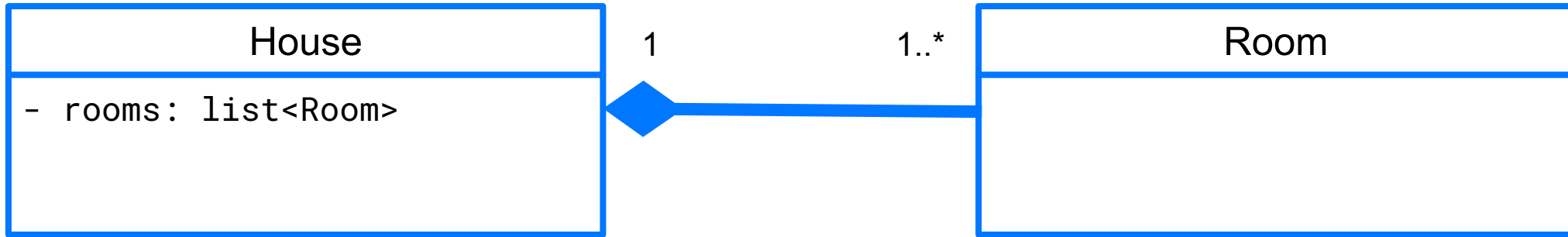


Диаграмма классов

Пример

Послесловие

- Программирование - это принятие решений

Домашнее задание 5

- Сформулированные требования (use case / userstory)
- UML диаграмма классов
- Желательно сформулировать основные сценарии

Перерыв

Принципы ОО проектирования SOLID

Принципы способствующие созданию таких систем, которые легко поддерживать и расширять.

ИНАЧЕ ГОВОРЯ,
СОПРОВОЖДАТЬ

- S - Single-responsibility principle
- O - Open-closed principle
- L - Liskov substitution principle
- I - Interface segregation principle
- D - Dependency Inversion Principle



Роберт Мартин
(Автор Agile)

Принцип единственной ответственности

S - Single responsibility principle

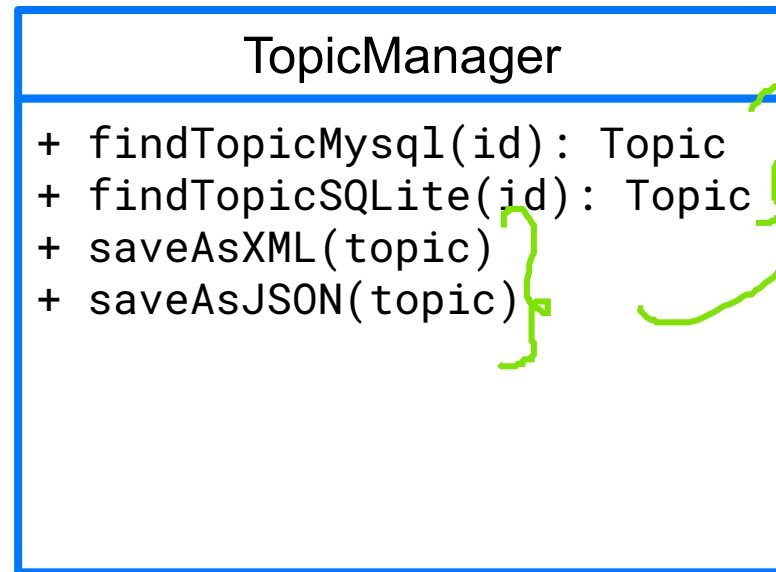
Класс/метод должен иметь только одну причину для изменения.

примеры

Принцип единственной ответственности

S - Single responsibility principle

Класс/метод должен иметь только одну причину для изменения.



СЛОЖНО ТЕСТИРОВАТЬ, ВНОСИТЬ ИЗМЕНЕНИЯ,
ИСПОЛЬЗОВАТЬ В ДРУГИХ ПРОЕКТАХ

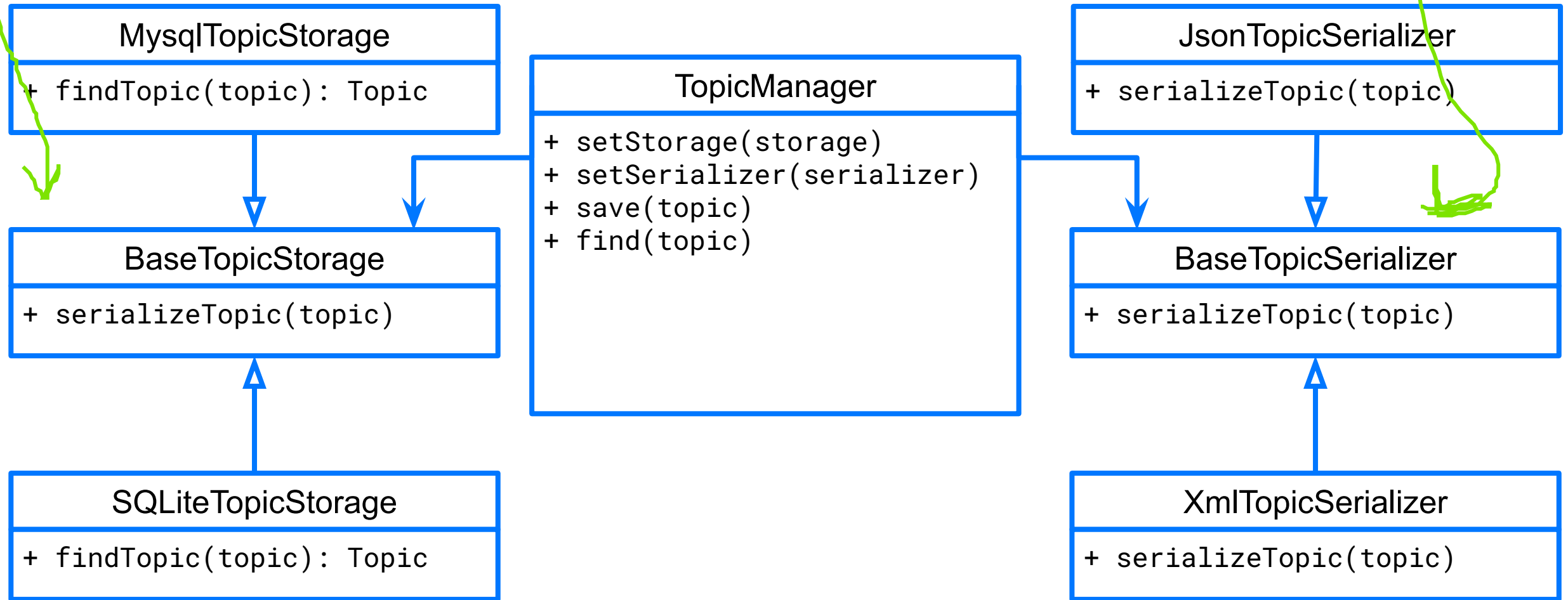
НИЗКАЯ ГИБКОСТЬ

при тестировании

STUB

MOCK

Принцип единственной ответственности



Принцип открытости/закрытости

O - Open-closed principle

Принцип открытости/закрытости

Программные сущности должны быть открыты для расширения,
но не для модификации.

ЛУЧШЕ ДОБАВЛЯТЬ НОВЫЙ ФУНКЦИОНАЛ В СИСТЕМУ, НЕЖЕЛИ МЕНЯТЬ
СУЩЕСТВУЮЩИЙ И ТЕМ САМЫМ УВЕЛИЧИВАТЬ ВЕРОЯТНОСТЬ ТОГО, ЧТО
ЧТО-ТО НАЧНЕТ ЛОМАТЬСЯ

ПРИМЕРЫ: ЕСЛИ ПРЕДУСМОТРЕНЫ ИНТЕРФЕЙСЫ, ТО ЛУЧШЕ ДОБАВИТЬ
НОВЫЙ КЛАСС ДЛЯ РЕАЛИЗАЦИИ ЗАПРОШЕННОЙ ЛОГИКИ, НЕЖЕЛИ
МЕНЯТЬ СИГНАТУРУ ИНТЕРФЕЙСА И КУЧУ ВЫТЕКАЮЩИХ ЗАВИСИМОСТЕЙ.
МОЖНО ПОПРОБОВАТЬ ДОБАВИТЬ НОВЫЙ КЛАСС И ИСПОЛЬЗОВАТЬ ЕГО В
ШАБЛОНАХ, ЕСЛИ ТАКОЕ ПРЕДУСМОТРЕНО

ЕСЛИ МНОГО ПЕКАСОВ И ТРЕБУЕТСЯ ИЗМЕНИТЬ ФУНКЦИОНАЛ КАКОЙ-ТО
БИБЛИОТЕКИ, ВОЗМОЖНО, ЛУЧШЕ ПРОСТО СДЕЛАТЬ НОВЫЙ АНАЛОГ И
ИСПОЛЬЗОВАТЬ ЕГО В ДАЛЬНЕЙШЕЙ РАЗРАБОТКЕ, А НЕ МЕНЯТЬ СТАРЫЙ



Бертран Мейер
(Автор контрактного
программирования)

Принцип подстановки Барбары Лисков

L - Liskov substitution principle

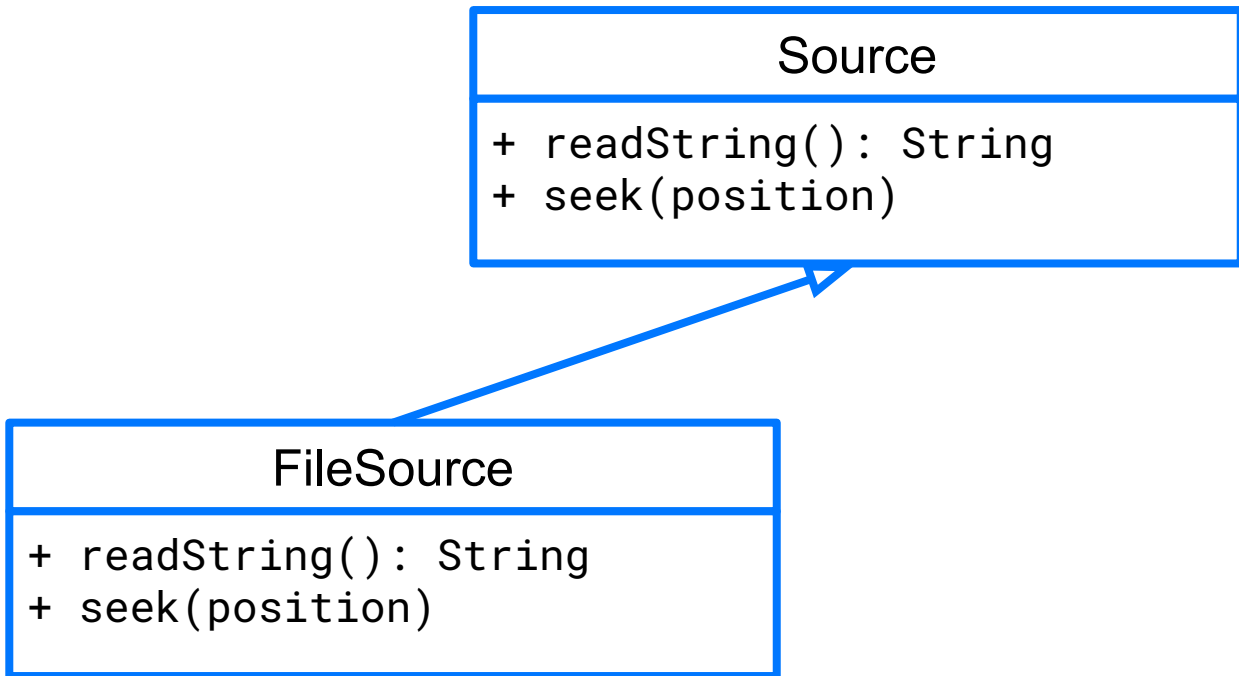
Принцип подстановки Барбары Лисков

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

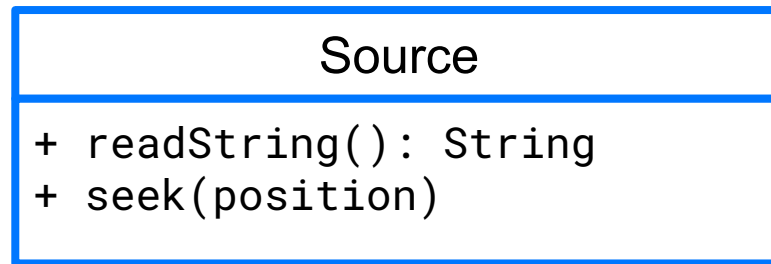


Барбара Лисков
(лауреат премии Тьюринга,
первая женщина в США,
получившей степень
доктора по информатике)

Принцип единственной ответственности

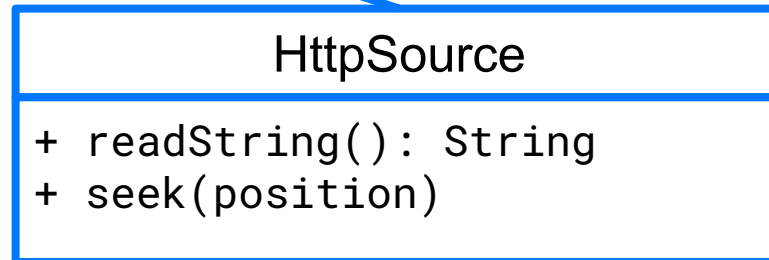
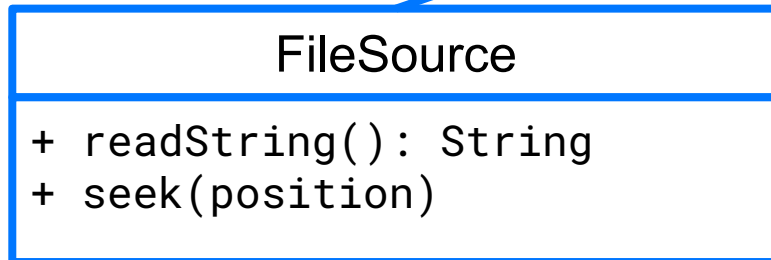


Принцип единственной ответственности



FileSource - удовлетворяет требованиям базового класса Source, HttpSource - нет(не может быть реализации seek), аналогично - с stding потоком данных и тд

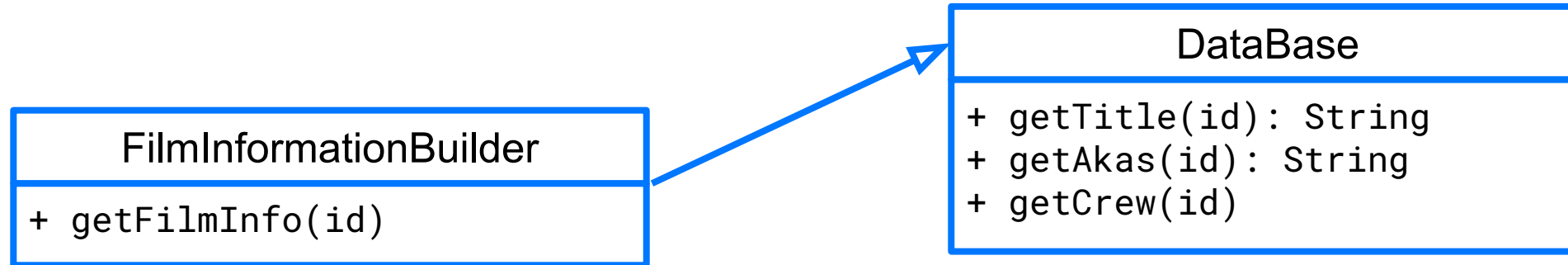
Нарушения приципа подстановки - реалзация пустого метода --> как следствие, архитектурные проблемы



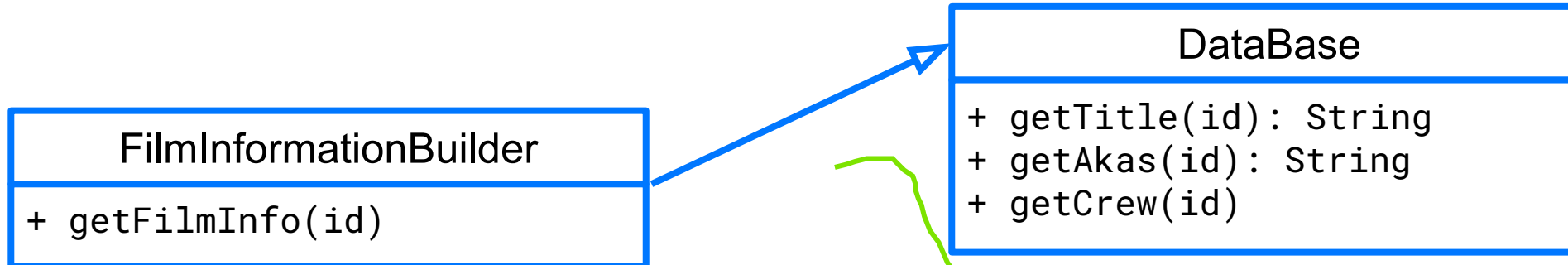
```
void seek() {
    throw std::runtime_error("I cant seek");
}
```

при подстановке наследника в логику, работающую с базовым классом, произойдет непредвиденное нехарактерное поведение

Принцип единственной ответственности



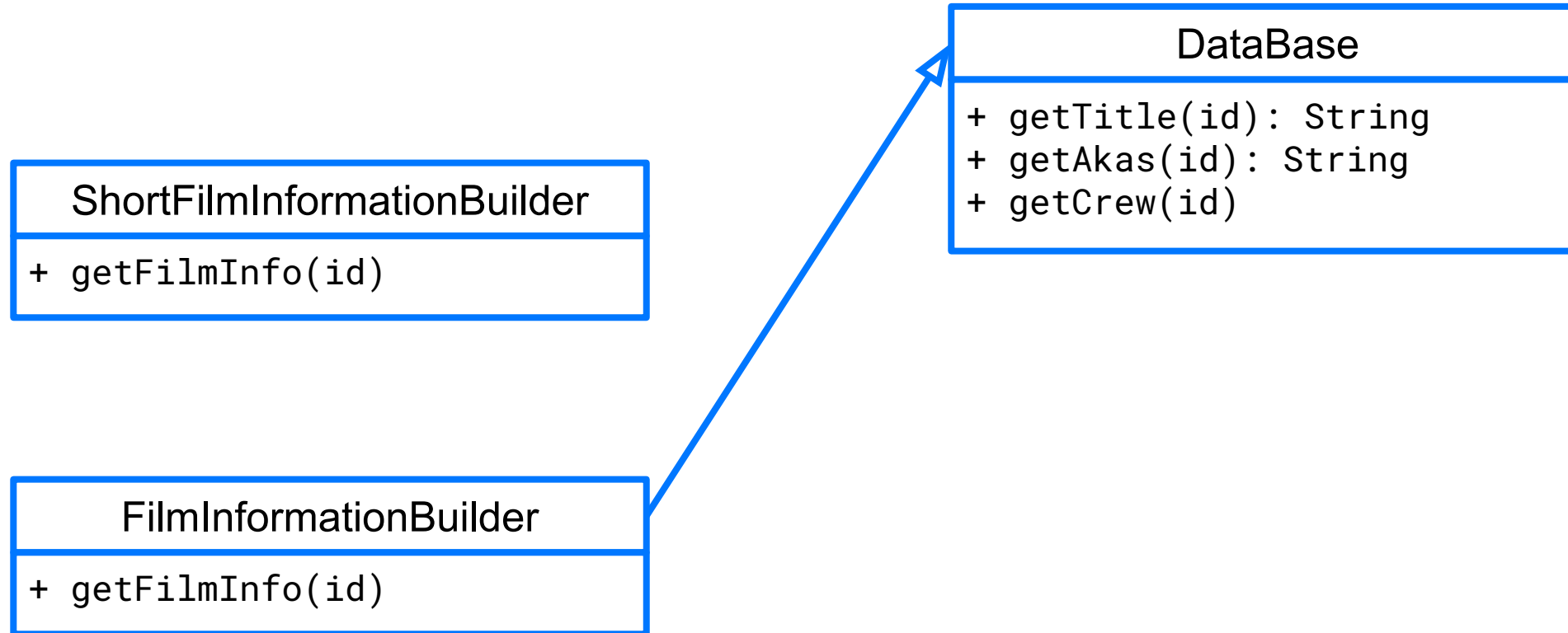
Принцип единственной ответственности



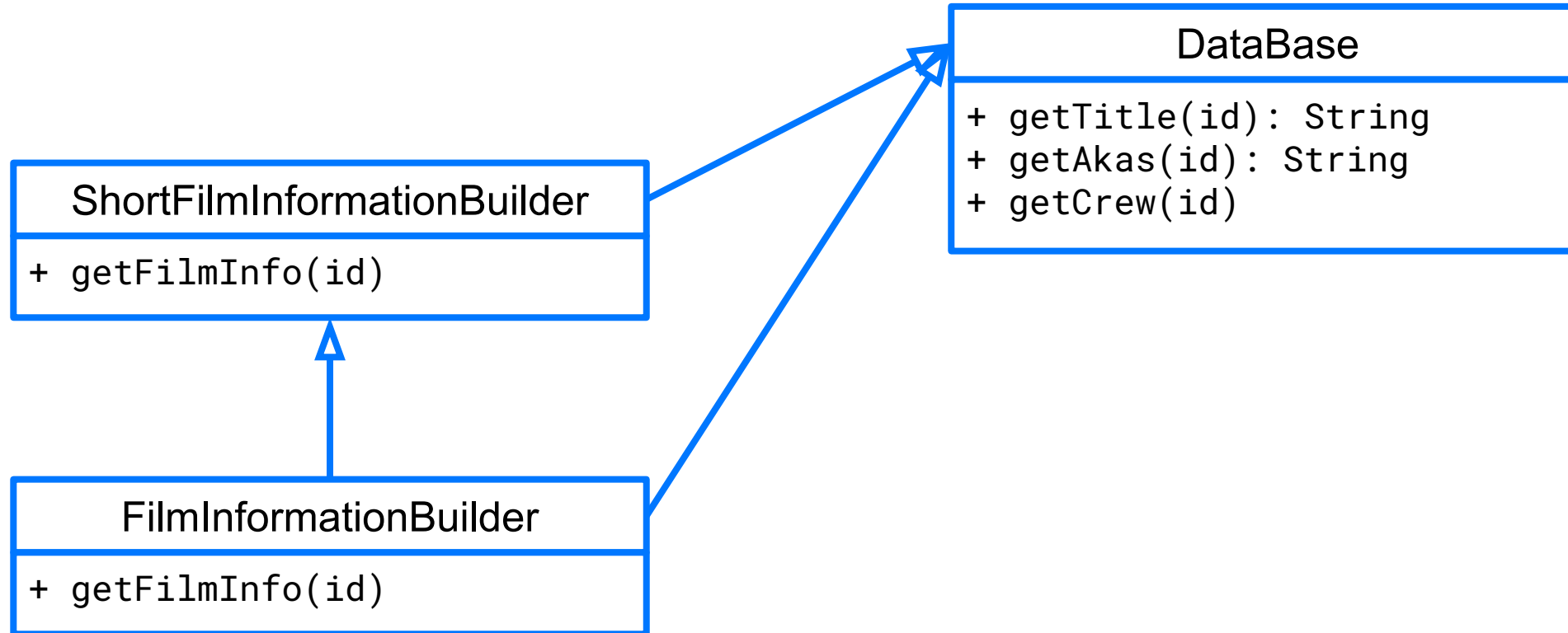
```
std::string getFilmInfo(id) {
    std::stringstream ss;
    ss << getTitle(id) << "\t";
    ss << getAkas(id, "RU") << "\t";
    return ss.str();
}
```

нарушение принципа подстановки для
облегчения переиспользования методов
класса-родителя - НЕПРАВИЛЬНО

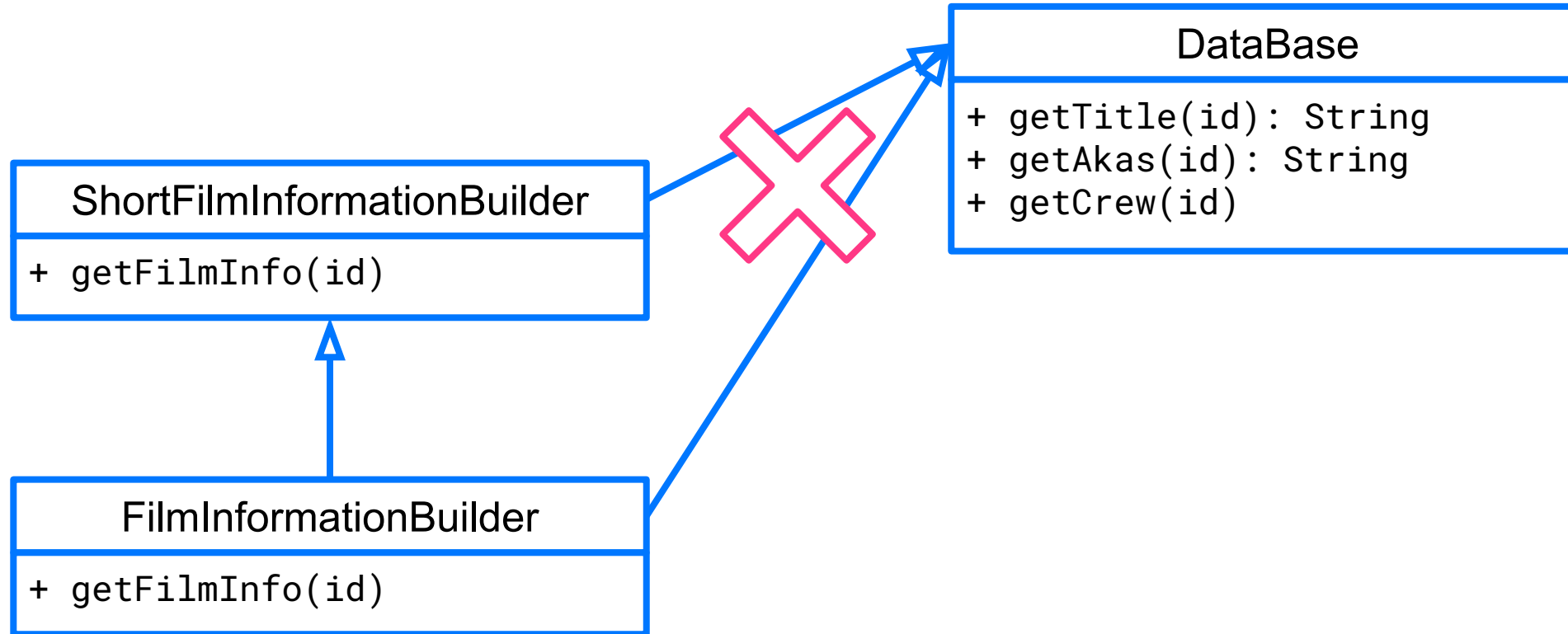
Принцип единственной ответственности



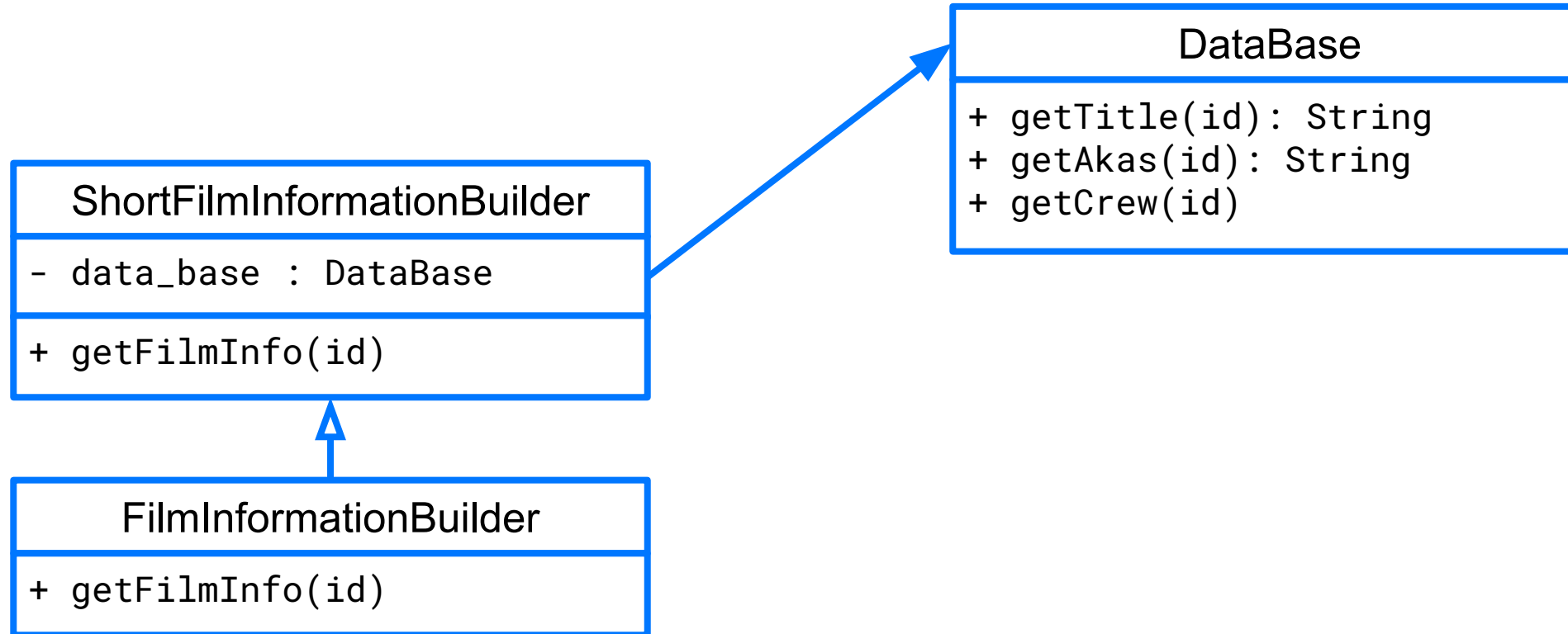
Принцип единственной ответственности



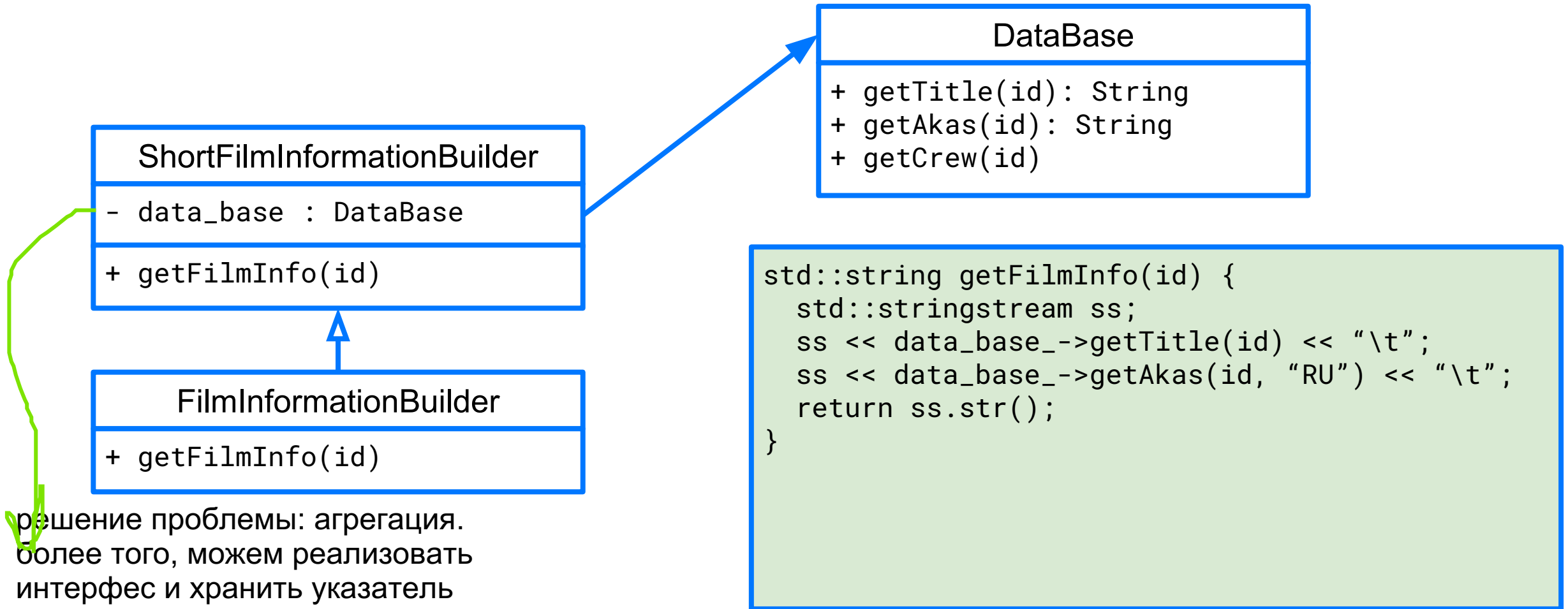
Принцип единственной ответственности



Принцип единственной ответственности



Принцип единственной ответственности



решение проблемы: агрегация.
более того, можем реализовать
интерфейс и хранить указатель
на него, тогда внедрить новую
логику будет намного проще

Принцип подстановки Барбары Лисков

L - Liskov substitution principle

Принцип подстановки Барбары Лисков

Функции, которые используют базовый тип, должны иметь возможность использовать подтипы базового типа, не зная об этом.
т.е. подстановка наследников базового класса вместо него самого не должна привести к нарушению логики

Не использовать публичное наследование только для повторного использования кода.

Агрегация лучше, чем наследование.

агрегация/ассоциация - используем сущности, например, через указатели

иными словами, наследники должны дополнять поведение базового класса, а не заменять его



Барбара Лисков
(лауреат премии Тьюринга,
первая женщина в США,
получившей степень
доктора по информатике)

Принцип разделения интерфейса

I - Interface segregation principle

Программные сущности не должны зависеть от методов, которые они не используют.

Принцип разделения интерфейса

I - Interface segregation principle

Программные сущности не должны зависеть от методов, которые они не используют.

Много маленьких интерфейсов лучше, чем один большой.

Принцип разделения интерфейса

I - Interface segregation principle

Программные сущности не должны зависеть от методов, которые они не используют.

Много маленьких интерфейсов лучше, чем один большой.

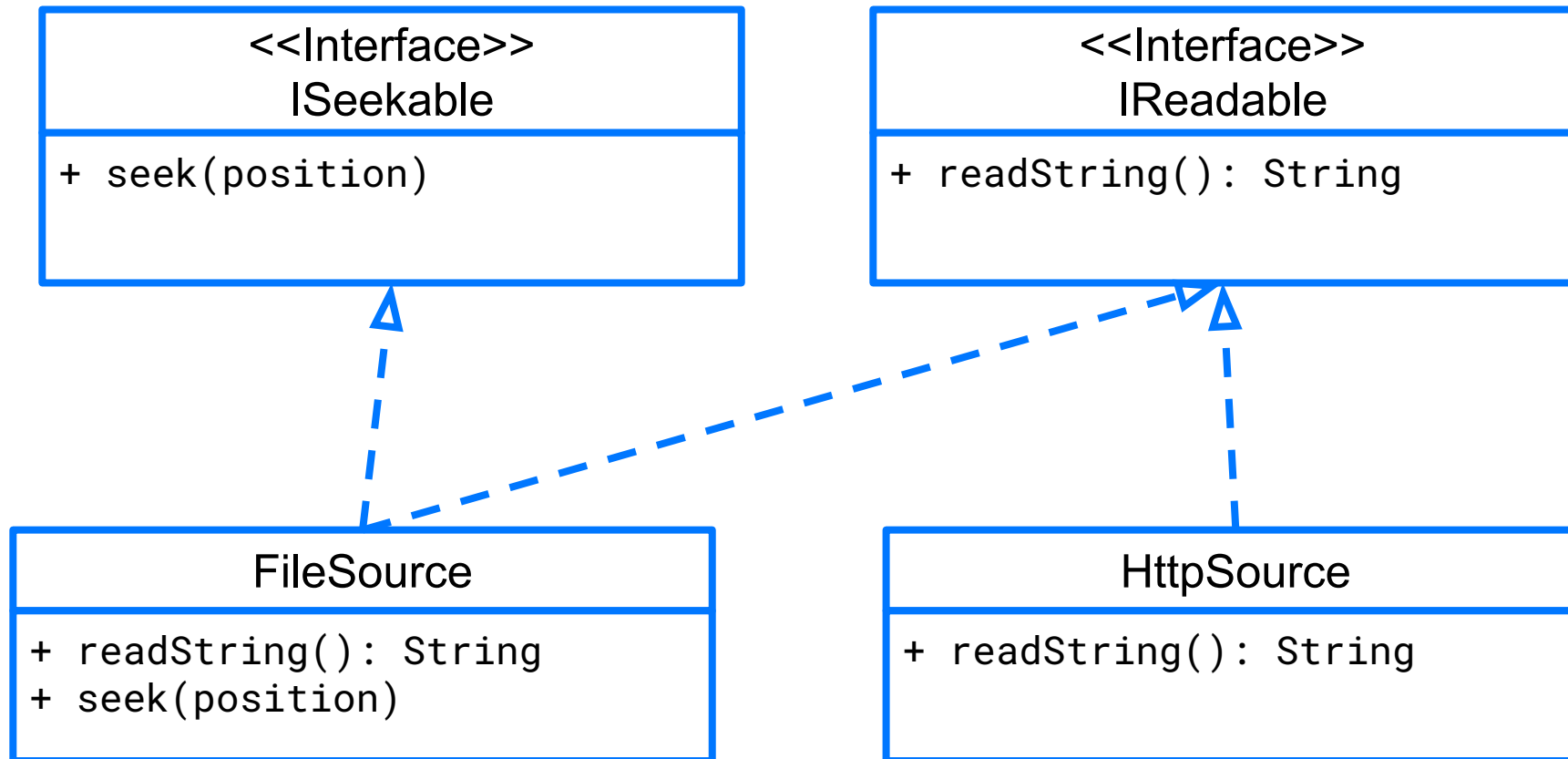
IWritable, IReadable, ISerializable.

а если будет один большой интерфейс, то его наследники вынуждены хранить информацию о методах, которые не подразумеваются в их работе

Принцип разделения интерфейсов говорит о том, что слишком «толстые» интерфейсы необходимо разделять на более маленькие и специфические, чтобы программные сущности маленьких интерфейсов знали только о методах, которые необходимы им в работе.

т.е. большой интерфейс накладывает большое кол-во ограничений на классы, которые от него наследуются

Принцип разделения интерфейса



Принцип инверсии зависимостей

D - Dependency Inversion Principle

Принцип инверсии зависимостей

Модули верхних уровней не должны зависеть от модулей нижних уровней.

Оба типа модулей должны зависеть от абстракций.

Принцип инверсии зависимостей

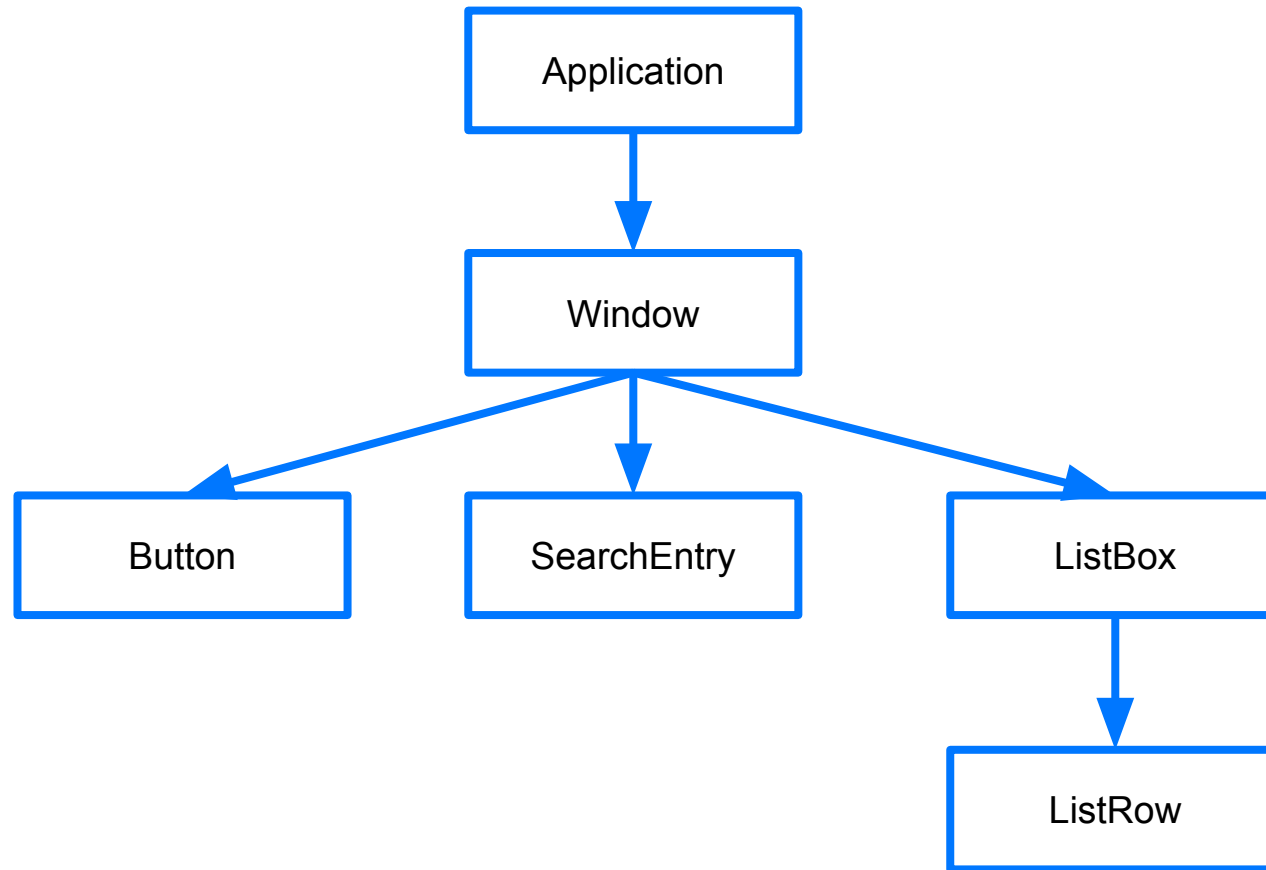
D - Dependency Inversion Principle

Принцип инверсии зависимостей

Модули верхних уровней не должны зависеть от модулей нижних уровней.

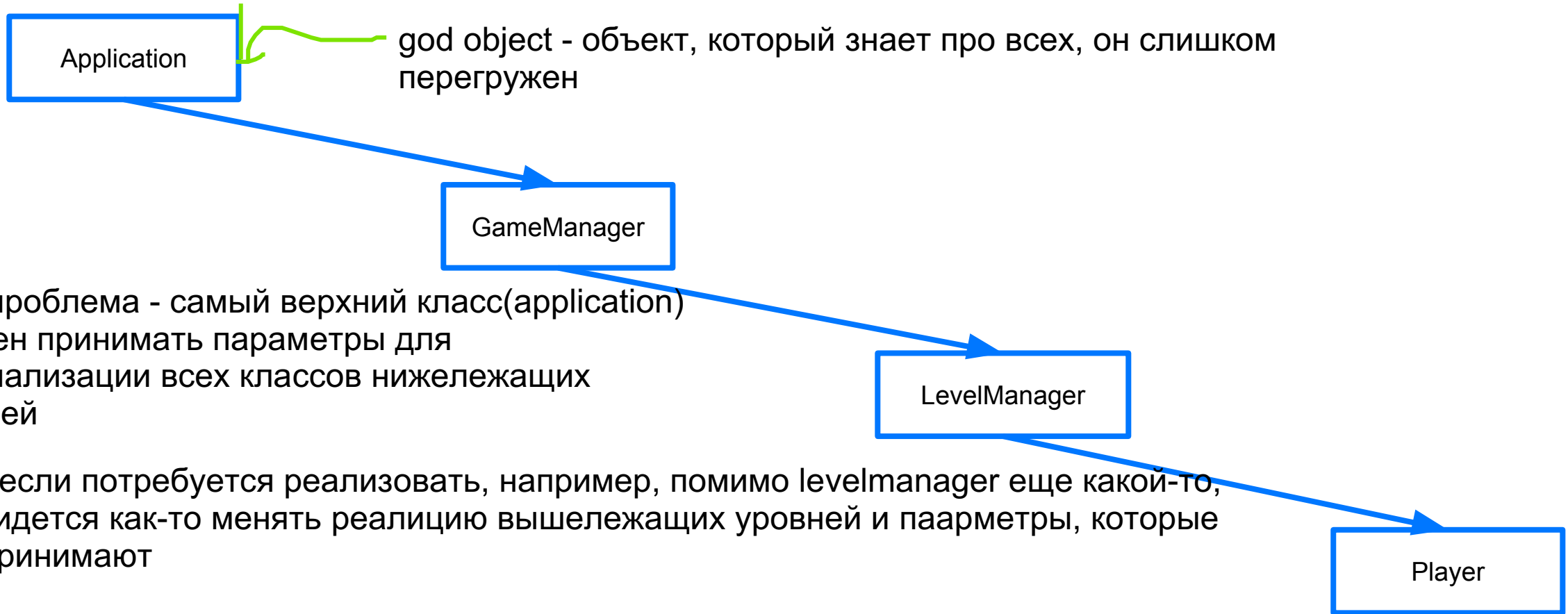
Оба типа модулей должны зависеть от абстракций.

Принцип инверсии зависимостей



допустим, application содержит gamemanager и так по цепочке

Принцип инверсии зависимостей



Принцип инверсии зависимостей



при новом подходе можем спокойно менять логику, тестировать классы(подменять их моками/стабами, удовлетворяющими требованиям интерфейсов)

новый подход: держать в верхних уровнях ассоциацию(ссылка, указатель) на интерфейс нижнего уровня
ПРИ ЭТОМ СОЗДАНИЕМ И ИНИЦИАЛИЗАЦИЕЙ СЛОЖНЫХ ЗАВИСИМОСТЕЙ КЛАССЫ ВЕРХНИХ
УРОВНЕЙ ЗАНИМАТЬСЯ НЕ ДОЛЖНЫ, ОНИ НЕ ДОЛЖНЫ ЗНАТЬ О ВОЗМОЖНЫХ ОШИБКАХ И
ИСКЛЧЕНИЯХ, ВЫБРАСЫВАЕМЫХ ПРИ НЕПРАВИЛЬНОЙ ИНИЦИАЛИЗАЦИИ И ТП, пусть этим
занимается другой класса-инициализатор, класс-ассемблер (то есть зависимости вводятся в верхний объект)⁵⁸

Литература

- Бертран Мейер.
Объектно-ориентированное конструирование
программных систем
- Крэг Ларман.
Применение UML 2.0 и шаблонов проектирования.



Спасибо за внимание!

