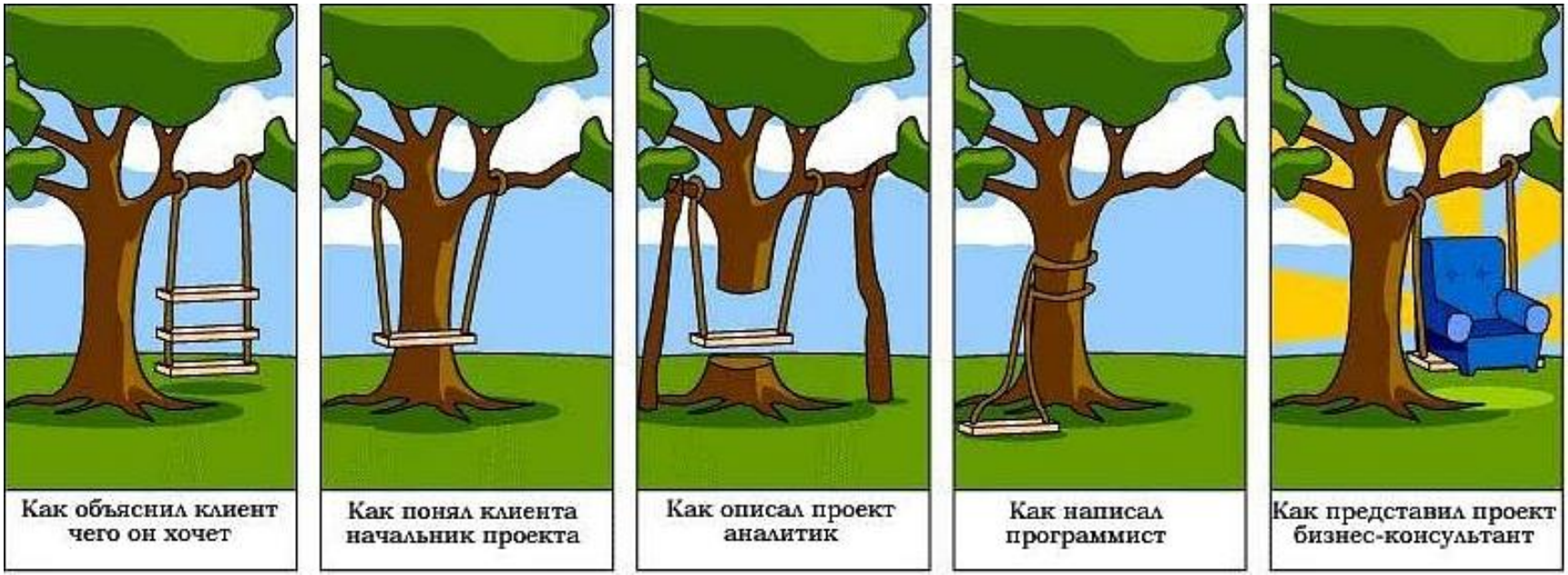
Информатика

Разработка ПО. Проектирование

Типичный программный проект



Типичный программный проект

Разработка ПО

- ПО – больше чем программы
  - сложная динамическая система
  - согласованная с многими интерфейсами
  - развивающаяся во времени
  - изменяющаяся/эволюционирующая
- Разработка ПО ≠ программирование

Разработка ПО

- Деятельность по созданию нового ПО
  - прототипирование
  - программирование
  - документирование
  - тестирование
  - поддержка
  - …
- Часть программной инженерии

Программная инженерия

Термин Software Engineering (SE) предложен Ф.Л. Бауэром в 1968г.

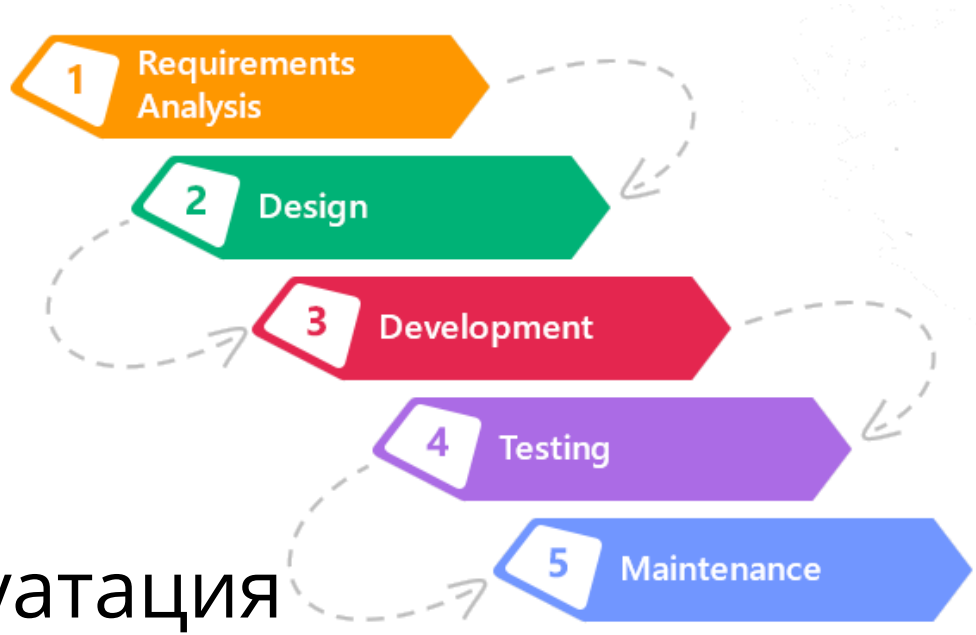
- **Определение ISO/IEC 2382-1**
  - Систематическое применение научных и технических знаний, методов и опыта для разработки, реализации, тестирования и документирования программного обеспечения.
- **Определение Иана Соммервиля** [кн. Software engineering]
  - инженерная дисциплина, охватывающая все аспекты создания ПО от начальной стадии разработки системных требований через создание ПО до его использования

SE и информатика

- Информатика
  - подходы к программированию
    - теория и методы построения вычислительных и программных систем
- Программная инженерия
  - решение проблем производства ПО
  - основывается на фундаменте информатики

Области SE

- Требования
- Проектирование
- Конструирование
- Тестирование
- Поддержка и эксплуатация



- еще конфигурация, управление, процесс, методы и инструменты, качество

Жизненный цикл

Software development life cycle описывает фазы развития программного проекта

Существует несколько моделей SDLC (методологий)

- каскадная,
- спираль,
- гибкая разработка,
- быстрое прототипирование,
- инкрементальная…

Проблемы при разработке ПО

- Отсутствие четкой спецификации – подробного описания системы, полностью определяющего цели и функциональные возможности
- Недостаток прозрачности при недостаточном планировании структуры (архитектуры)
- Недостаток контроля над процессом
- Недостаток мониторинга
- Неконтролируемые изменения

Спецификации

Все называют по-своему

- постановка задачи
- требования пользователя
- техническое задание
- функциональная спецификация
- архитектура системы

Говоря на своем языке специалисты зачастую не понимают друг друга

Решение

- Проблему можно решить при наличии единого, унифицированного средства создания спецификаций, достаточно простого и понятного для всех заинтересованных лиц

UML

- Unified Modeling Language
- единый универсальный стандарт для визуального описания спецификаций
- язык объектно-ориентированного моделирования (визуальный)
- позволяет строить модель программных систем

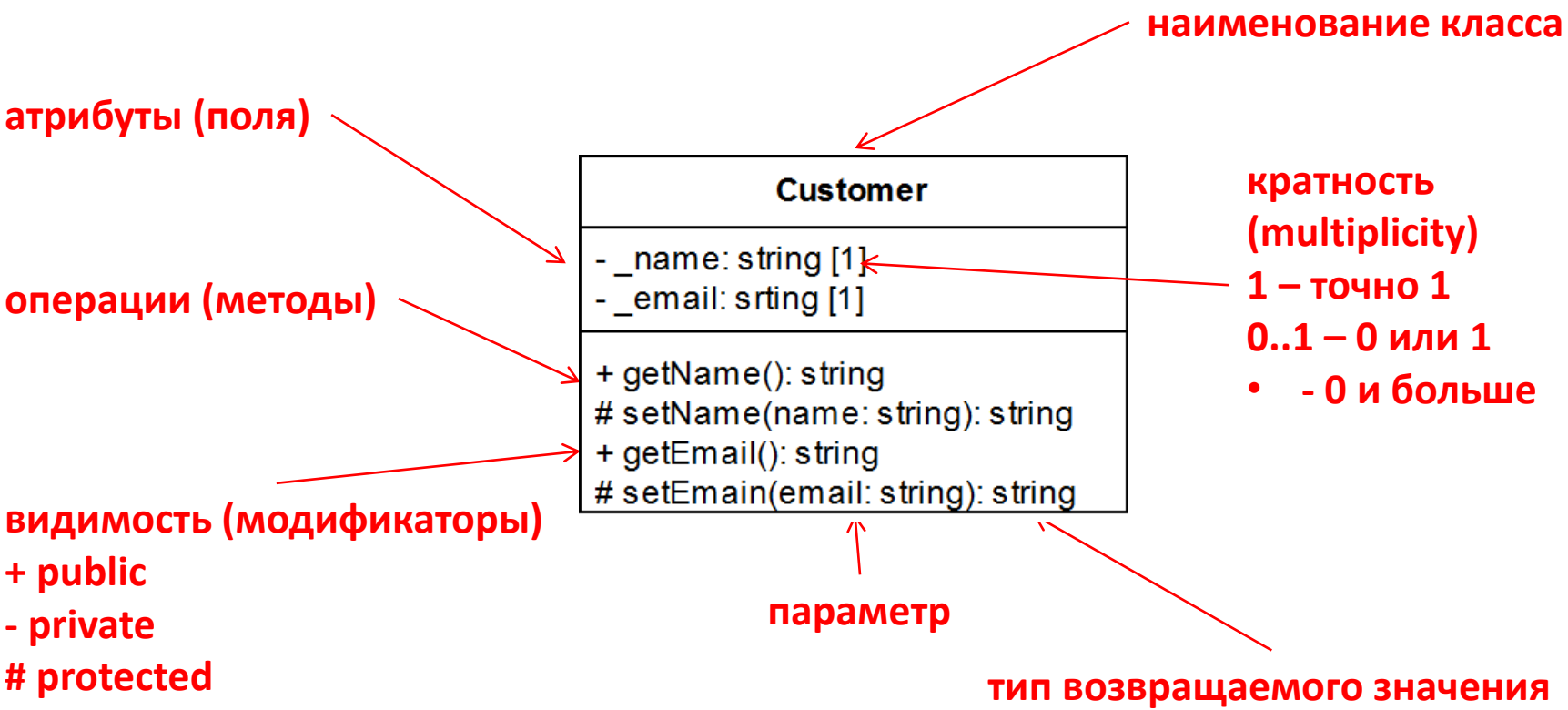
Появление UML

- ООП требовало удобного инструмента для моделирования
- 1991. Описания «Трех амиго» легли в основу языка
  Rumbaugh, Jacobson, Booch: The Unified Modeling Language Reference Manual
  - Грейди Буч (Grady Booch)
  - Джим Румбах (Jim Rumbaugh)
  - Айвар Якобсон (Ivar Jacobson)
- В 1995 UML был впервые продемонстрирован
- В 1997 стандартизировался

UML: диаграммы

- **Логическая структура**
  - диаграмма классов
  - структурная, взаимодействия, компонентов
  - вариантов использования (use-case)
- **Поведение**
  - активности
  - состояний
  - последовательности, коммуникации
- **Физическая структура**
  - внедрения

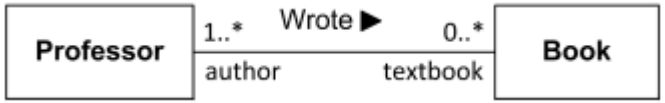
Класс


наименование класса
атрибуты (поля)
Customer
- _name: string [1]
- _email: srting [1]
операции (методы)
+ getName(): string
# setName(name: string): string
+ getEmail(): string
# setEmain(email: string): string
кратность (multiplicity)
1 – точно 1
0..1 – 0 или 1
• - 0 и больше
видимость (модификаторы)
+ public
- private
protected
параметр
тип возвращаемого значения

Ассоциация

- Структурная связь между классами (обычно ссылка)
- Бывает бинарной и n-арной
- Может иметь имя и навигацию
- Для классов могут быть указаны роли и кратность

Ассоциация



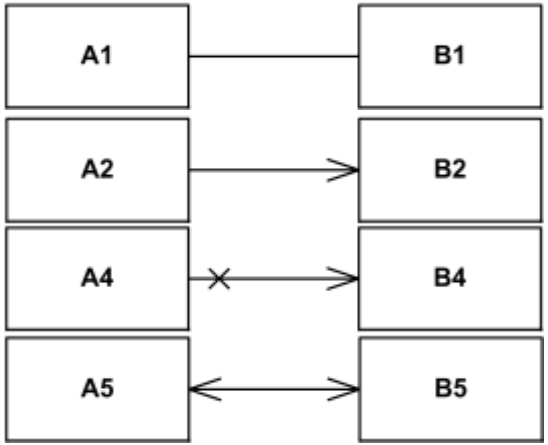
- Профессор в *роли* автора ассоциируется с учебником типа Книга
  - треугольник указывает как читать ассоциацию
- Класс может участвовать в разных ассоциациях в разных ролях
- Класс может иметь ассоциацию с самим собой

Навигация в ассоциации

Навигация указывает достижимость класса из другого

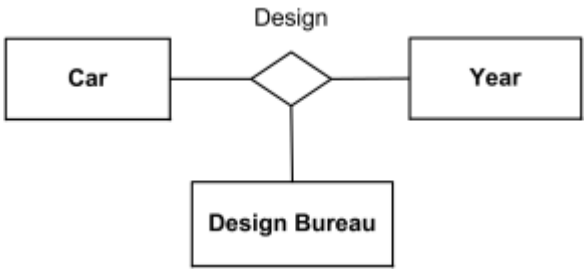
Может быть неопределена, разрешена и запрещена

- Неопределенная
- B2 достижим из A2
- A4 не достижим из B4
- A5 и B5 достижимы



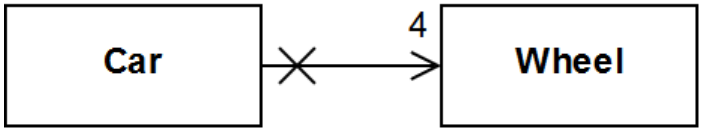
n-арная ассоциация

- Указывает на структурную связь между 3 и более классами
- Обозначается ромбом



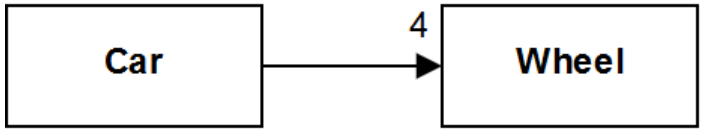
- пример тернарной ассоциации

Пример ассоциации



- Явно указано что навигация запрещена
- Колеса достижимы из машины, но не наоборот
- Машина имеет ссылки на 4 колеса

Пример ассоциации



- Что, если у машины кратность *?
- Тогда одно колесо может быть прикреплено к нескольким машинам
- Если у машины кратность 1, то одно колесо может быть только на одной машине

Выводы по ассоциации

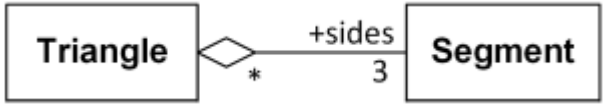
- Ассоциации – это то, что «склеивает» систему
- Ассоциация – связь
- Ассоциация описывает отношение между объектами во время исполнения

Агрегация

- Бинарная ассоциация
- Связывает «целое» и «часть» (части)
- Только один конец ассоциации может быть агрегацией (обозначается ромбом)
- «Части» могут одновременно содержаться в нескольких «целых»

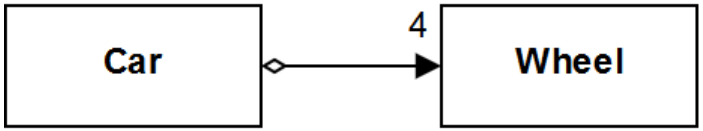
Агрегация

- Бинарная ассоциация



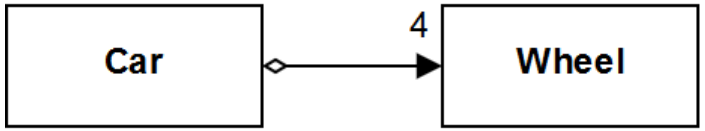
- Связывает «целое» и «часть» (части)
- Только один конец ассоциации может быть агрегацией (обозначается ромбом)
- «Части» могут одновременно содержаться в нескольких «целых»
- Не один сложных объект не может быть частью самого себя

Пример



- Что означает агрегация в случае с машиной и колесами?
- Чем отличается от ассоциации?

Пример



- Агрегация может привести к ошибкам проектирования (несогласованности)
- Используйте аккуратно

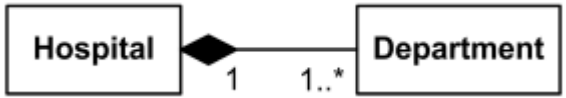
Композиция

- Бинарная ассоциация
- Сильная форма агрегации
- Обозначает отношения между «целым» и «частью» («частями»)
- «Часть» может содержаться только в одном «целом»
- Срок жизни «целого» и «части» совпадает

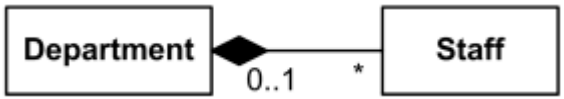
Композиция

- **Обозначается закрашенным ромбом**
- Папка может содержать много файлов. При удалении папки удаляются и файлы

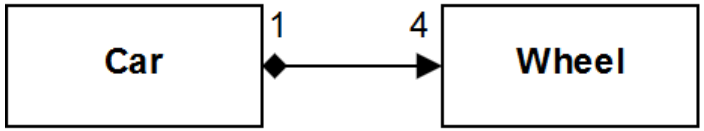
- У больницы есть несколько отделений. Каждое отделение принадлежит только одной больнице. Отделения закрываются вместе с больницей



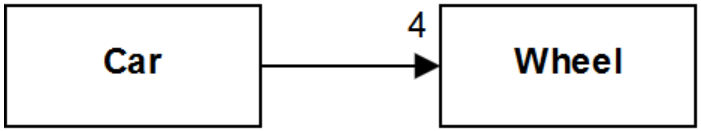
- У отдела есть персонал. Каждый сотрудник может быть членом отдела или быть «сам по себе»



Пример

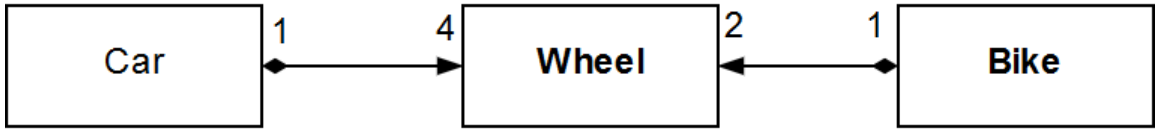


- Чем отличается от ассоциации?

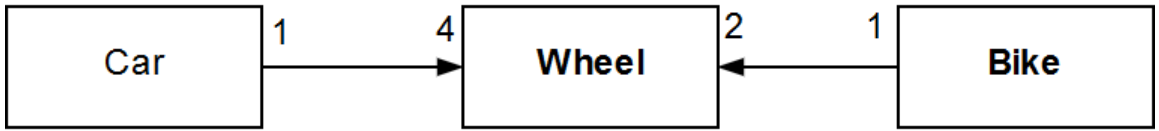


- Кратность композиции 0..1
- У экземпляра может быть только один владелец

Пример

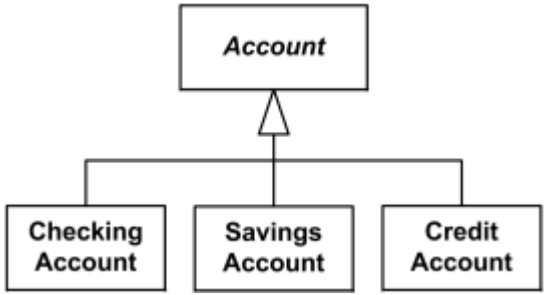


- Колеса могут быть частью машины и мотоцикла
- С композицией одно колесо не может быть частью машины и мотоцикла одновременно
- С ассоциацией такого ограничение на «одного владельца» – нет!



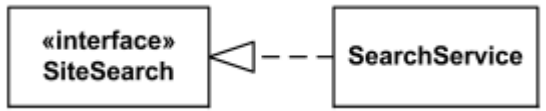
Обобщение

- Реализуется наследованием
- Указывает на отношение между базовым и производным классом (суперклассом и подклассом)



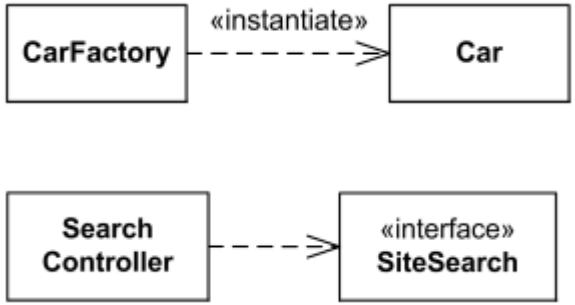
Реализация

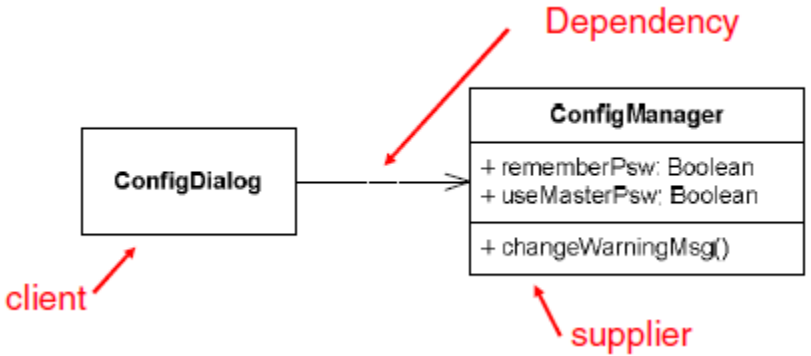
- Отношение между *спецификацией* и *реализацией*
- В чем отличие между интерфейсом и абстрактным классом?
- Может обозначаться по-разному



Зависимость

- Направленное отношение между элементами, обозначается пунктирной стрелкой
- Изменения в одном влекут изменения в другом



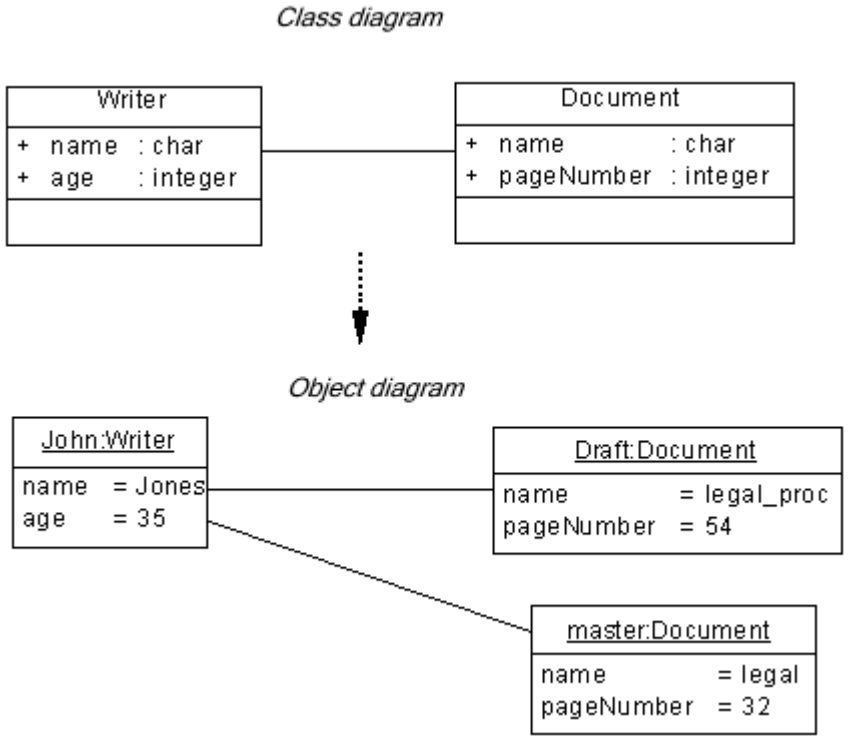


Отношения

- Ассоциация: У A есть ссылка на экземпляры B (поля)
- Агрегация: часть может существовать отдельно от целого, может приводить к ошибкам проектирования
- Композиция: часть может принадлежать одному целому или быть сама по себе
- Обобщение: реализуется наследованием, экземпляр A может использоваться там, где ожидается B
- Реализация: A реализует интерфейс B
- Зависимость: A зависит от B, если изменения в B повлекут изменения в A

Классы и объекты

- Диаграмма классов - отношения между классами
- Диаграмма объектов – отношения между конкретными экземплярами классов



Принципы проектирования

Пять основных принципов объектно-ориентированного программирования и проектирования

S.O.L.I.D.

«...акроним, введённый Майклом Фэзерсом для пяти принципов, предложенных Робертом Мартинов в начале 2000-х» - Wiki

S.O.L.I.D.

- Критерии проверки системы на изменяемость и расширяемость
- Руководства для проведения рефакторинга плохо спроектированного кода

Признаки плохой модели

Читайте про «Код с душком»: Model smell, Code Smell

Несколько примеров:

- Слишком большие/маленькие классы
- Одно изменение затрагивает несколько компонент.
- После изменений перестаёт работать то, что раньше работало
- Трудно выделить компоненты для повторного использования
- Трудно добиться корректного поведения, а выполнить некорректные действия – легко
- Неоправданная сложность, зависимость от модулей, не дающих непосредственной выгоды
- Трудно разобраться в проекте, анализировать его

Расшифровка акронима

- S – single responsibility
- O – open/closed
- L – Liskov substitution
- I – interface segregation
- D – dependency inversion

S: Single responsibility

Принцип единственной ответственности

Не должно быть больше одной причины для изменения класса

Если у класса больше одной ответственности, он будет меняться часто, дизайн будет хрупким, изменения - непредсказуемыми

Пример: валидация данных

Если осуществлять проверку данных в самом классе, то может возникнуть потребность в изменении класса при использовании его экземпляров в разных клиентах

Решение: делегировать валидацию стороннему объекту, чтобы основной объект не зависел от реализации валидатора

Предельные случаи (ошибки)

- Крайний пример несоблюдения принципа – God object
  - – божественный объект – знает и умеет всё
- Крайний пример соблюдения принципа - необоснованная сложность, размазывание логики

O: Open/closed

Программные сущности должны быть открыты для расширения (изменения), но закрыты для изменения

Модель должна быть устойчива к изменениям

Закрытость – стабильность интерфейсов

Открытость – можно изменять поведение без изменения исходного кода класса (пересборки)

Примеры ошибок:

- Использование конкретных объектов без выделения абстракций
- Привязка к абстракции, но выстраивание логики в зависимости от конкретных типов (is, typeof)

Решение: выделять абстракции, привязываться к ним, не требовать дополнительной информации о конкретных типах

Крайние случаи (ошибки)

- Крайний случай несоблюдения – отсутствие абстракций
- Крайний случай соблюдения – чрезмерное число уровней абстракции
  - «фабрика фабрик»

L: Liskov substitution

Принцип подстановки/замещения Барбары Лисков

Вместо базового типа всегда должна быть возможность подставить его подтип

Реализация «правильного» наследования

Требовать меньше, гарантировать больше

Примеры ошибок

- Ошибочное наследование
  - Неочевидное изменение поведения, несогласующееся с иерархией
  - Изменение интерфейса при наследовании
- Анти-пример – непонятное наследование (или слишком много или совсем нет)

I: Interface segregation

Клиенты не должны зависеть от методов, которыми не пользуются

Идея – предоставить удобный интерфейс с точки зрения различных клиентов

Не заставлять клиенты реализовывать методы, которые им не нужны

Примеры ошибок

- Вынуждение наследников знать и делать слишком много
- «Жирный» интерфейс: все функциональные возможности в одном интерфейсе (клиенты разделены, а интерфейс – нет)
- Анти-пример: тысяча интерфейсов (интерфейс на каждый метод)

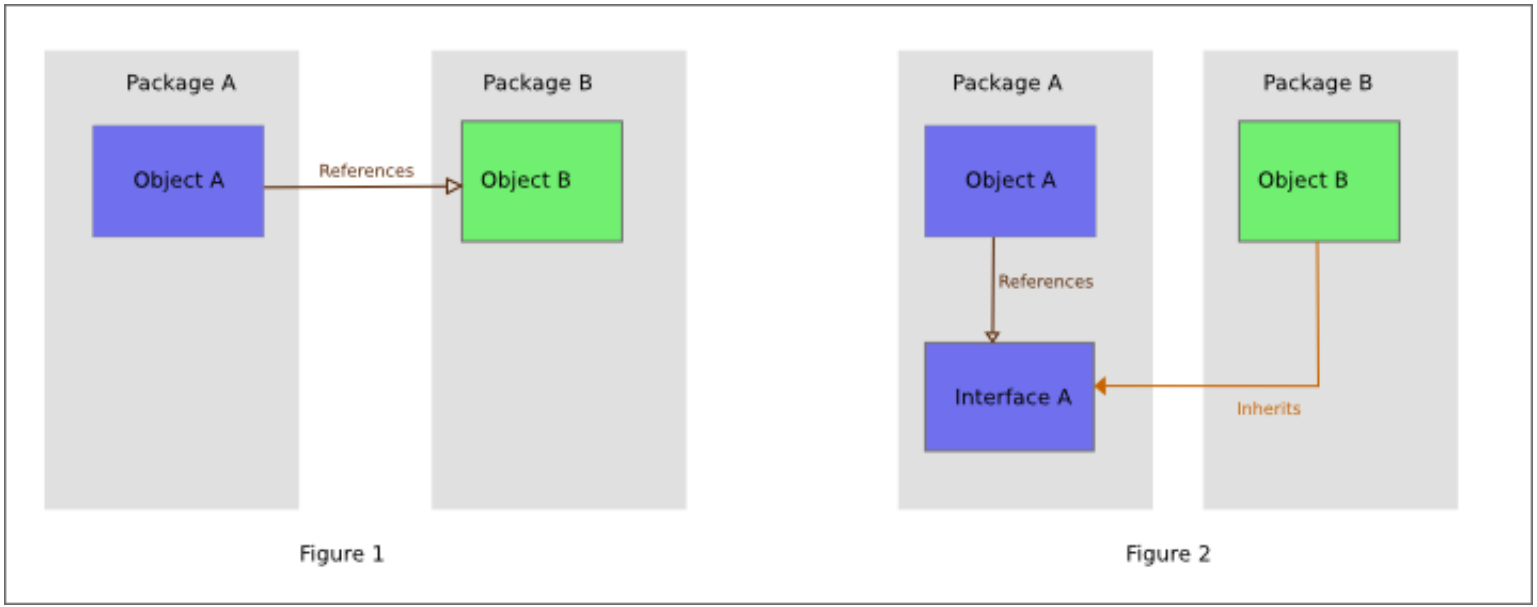
D: Dependency inversion

Принцип инверсии зависимостей

Модули верхнего уровня не должны зависеть от модулей нижнего, должны зависеть от абстракций

Идея – сделать ключевые или изменчивые зависимости явными

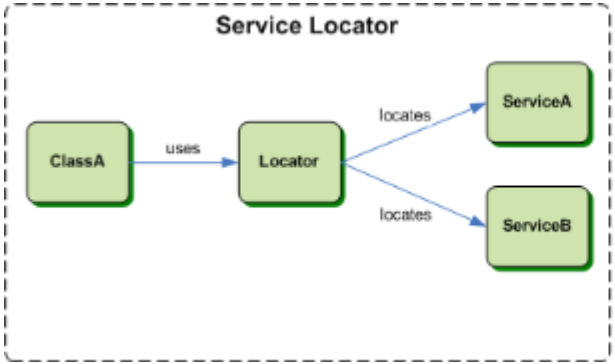
В чём инверсия

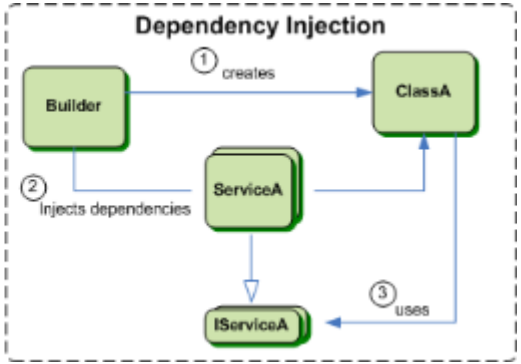


Картинка для Java, но суть - одна

Спойлер

- Инверсия завитимостей (Dependency Inversion)
- Инверсия управления (Inversion Of Control)
  - Service Locator
  - Dependency Injection (через конструктор, метод, интерфейс)





Примеры ошибок

Синглтоны, создание ключевых зависимостей в закрытых методах

Анти-пример: выделение интерфейса для каждого класса, передача всего и вся через конструкторы (невозможно понять где логика)

На следующей лекции

Обобщение опыта экспертов в решении типовых ситуаций

- Паттерны проектирования (Design Patterns)
  - Банда четырёх (Gang of Four)

Вопросы?

e-mail: marchenko@it.kfu.ru