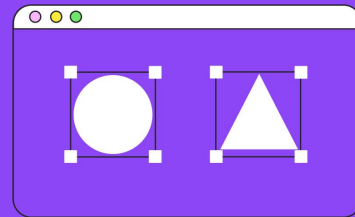


Структурное проектирование

Урок 3

На лекции мы изучим инструментарий структурного проектирования, позволяющий построить детализированную модель систему, позволяющую перейти к её реализации





Знакомство и содержание урока



Игорь Зуев

Руководитель ИТ-проектов

В Международном аэропорту Шереметьево







- ✨ 10 лет в ИТ (управление проектами)
- ✨ 3 года в преподавание (автор многочисленных обучающих программ по Project & Product Management)
- ✨ Группа компаний “Лукойл”, Проектный офис при Президенте одной из стран СНГ, компании из сферы строительства и нефтегаза
- ✨ ИТ-проекты по автоматизации финансовых систем
- ✨ ИТ-проекты в рамках строительства ГПЗ и магистрального газопровода



План курса



Что будет на уроке сегодня

-  Определим роль и место структурного проектирования
-  Узнаем как наличие структуры повышают качество модели системы
-  Изучим инструменты структурного описания
-  Познакомимся с классами и объектами
-  Изучим идеологические основы объектно-ориентированного программирования
-  Узнаем об актуальных принципах разработки о которых должен знать аналитик





Роль и место структурного проектирования



Вопрос

Какие понятия структуры Вам знакомы?





Ответ

В **философии**, Структура — совокупность связей между частями объекта

В **физике**, Структура — группа уровней энергии и спектральных линий, различающихся из-за квантовых взаимодействий



Ответ

В **химии**, структура - представляет собой пространственное упорядочение атомов и связей в молекуле

В **математике**, структура — какой-либо новый объект, вводимый на некотором множестве, свойство элементов множества



Ответ

В **менеджменте**, структура - это пакет официальных документов, отражающих иерархию и состав организации, а также функции, права и обязанности ее основных элементов

В **информатике**, структура — формат организации, управления и хранения данных, который обеспечивает эффективный доступ и модификацию



Структура системы - совокупность устойчивых связей между элементами системы, которые обеспечивают целостность системы и тождественность самой себе





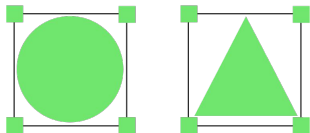
Основы объектно-ориентированного программирования

Результат структурного проектирования

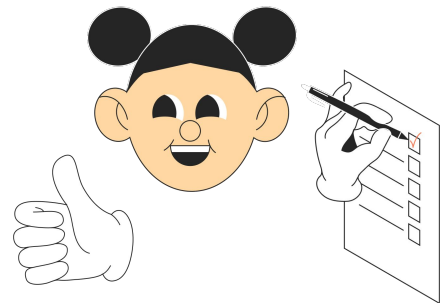
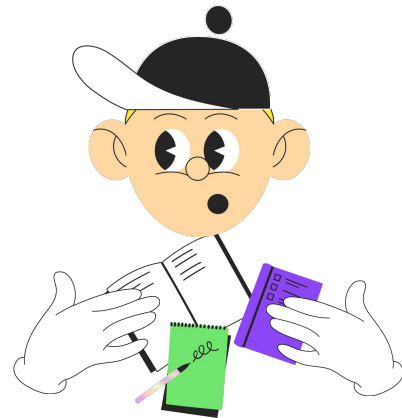
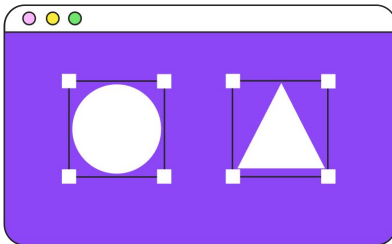
Результатом структурного анализа информационной системы выступает её объектная модель

Объектная модель

Помогает оптимизировать модель системы на низком уровне абстракции



Однозначно определяет реализацию системы





Оптимизация с применением
структурного моделирования
позволяет упростить описание
системы при сохранении её
целостности и функциональности



Вопрос

Знакомо ли Вам понятие объектно-ориентированное
программирование?





Ответ

Объектно-ориентированное программирование (сокращенно ООП) стало нормой промышленной разработки информационных систем, а структурный анализ результатом которого выступает построение объектной модели (англ. ORM – object relative model) стало нормой для большего числа проектов в различных сферах применения



Классы и объекты в структурном проектировании



Вопрос

Что такое понятие класс, объект и чем они
отличаются?



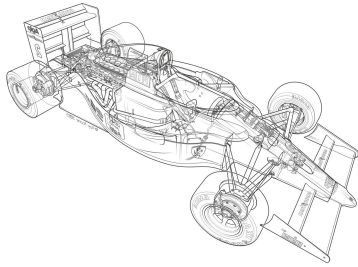


Ответ

Класс — это описание каким должен быть предмет.

Объект - конкретный предмет, выступающий
экземпляром класса.

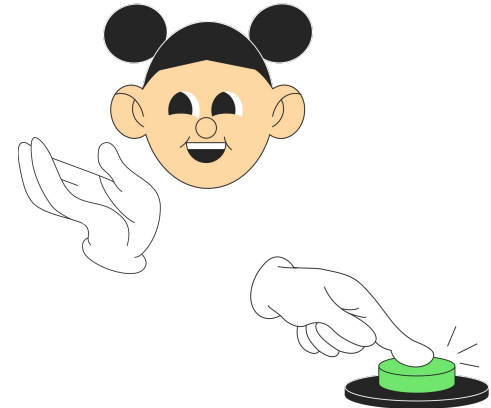
Понятие класса и объекта



Класс — это описание того, какими свойствами и поведением будет обладать объект



Объект — это экземпляр с собственным состоянием свойств в каждый момент времени, поведение которого совпадает с описанием класса





Когда мы говорим «свойства и поведение», то с точки зрения объектно-ориентированного программирования мы имеем ввиду переменные и функции - методы.



Основные понятия и принципы объектно-ориентированного программирования



Основные понятия и принципы объектно-ориентированного программирования



Конструкции объектно-ориентированного программирования

Если заглянуть в программный код, то мы увидим несколько типовых конструкций

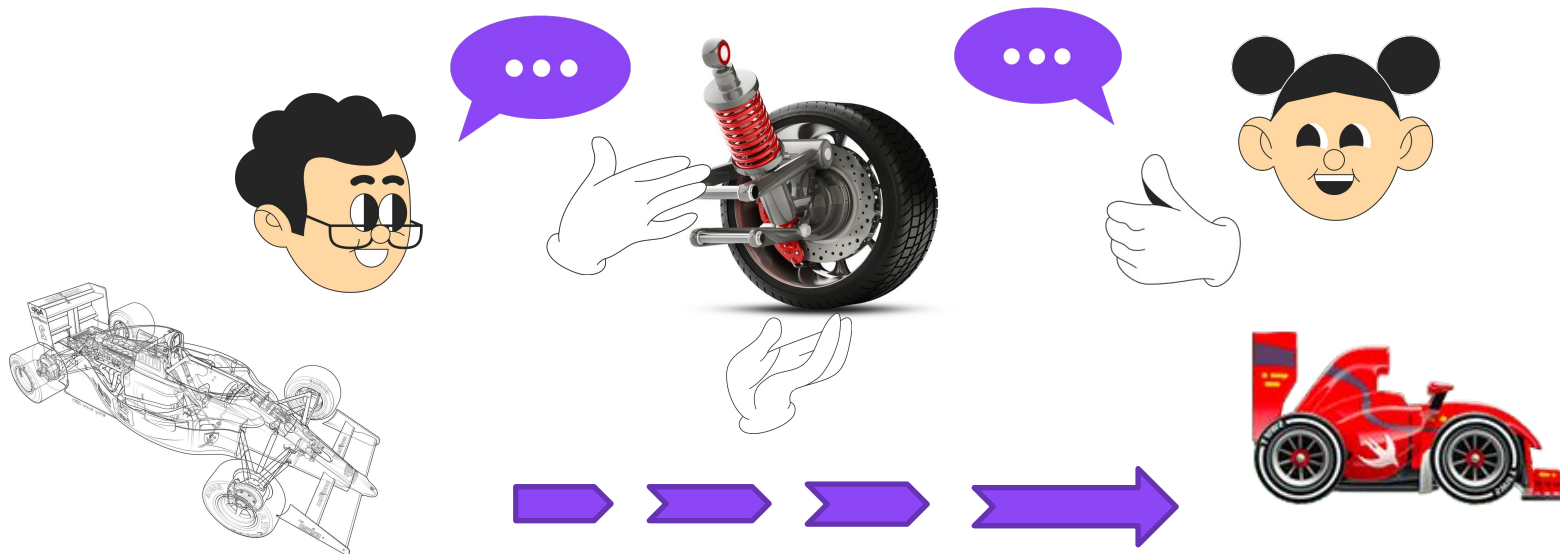


- 💡 **new** — это ключевое слово, которое необходимо использовать для создания нового экземпляра какого-либо класса. В этот момент создается объект и вызывается конструктор
- 💡 **this** (иногда **self**) — это специальная локальная конструкция переменная (внутри методов), которая позволяет объекту обращаться из своих методов к собственным атрибутам. Так обращение
- 💡 **constructor** (**construct**, **init** или совпадает с именем класса) — это специальный метод, который автоматически вызывается при создании каждого экземпляра объекта. Конструктор может принимать любые аргументы, как и любой другой метод. В каждом языке конструктор обозначается своим именем.



Конструкции объектно-ориентированного программирования

Например, для рассматриваемого класса автомобиль должен быть спроектирован и исполнен **метод установки колес** на штатные места с контролем давления в шинах, иначе использовать автомобиль не получится



Конструкции объектно-ориентированного программирования

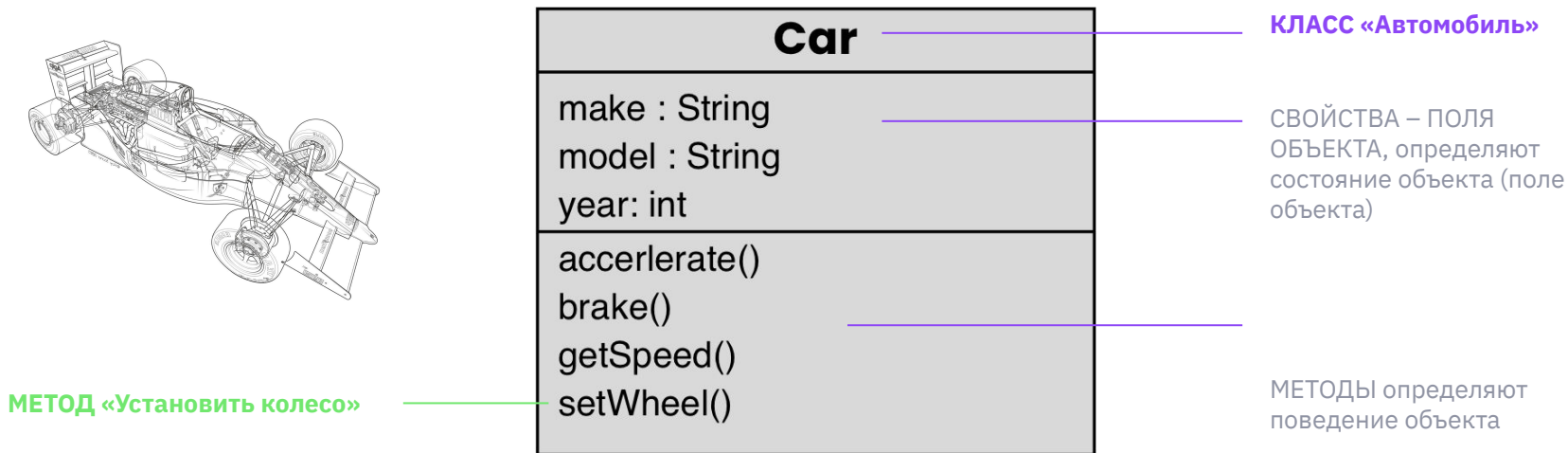


Диаграмма классов UML (от англ. "Class diagram") предназначена для представления внутренней структуры программы в виде классов и связей между ними.



SOLID — это пять основных принципов проектирования в объектно-ориентированном программировании

Единственной ответственности (Single responsibility)

одна функция должна выполнять только одно законченное действие

Открытости / закрытости (Open-closed)

программные сущности должны быть открыты для расширения, но закрыты для модификации

Подстановки Барбары Лисков (Liskov substitution)

Объекты в программе могут быть заменены их наследниками без изменения свойств программы

разделения интерфейса (Interface segregation)

Много специализированных интерфейсов лучше, чем один универсальный

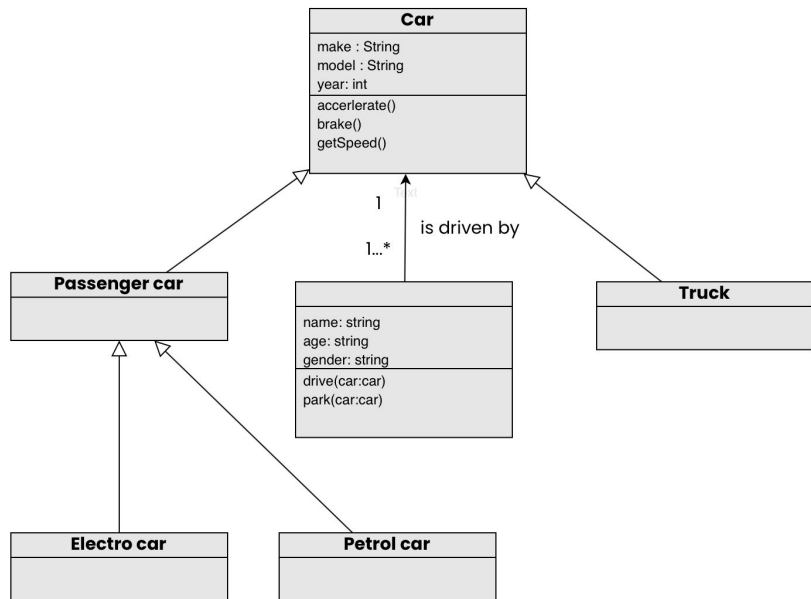
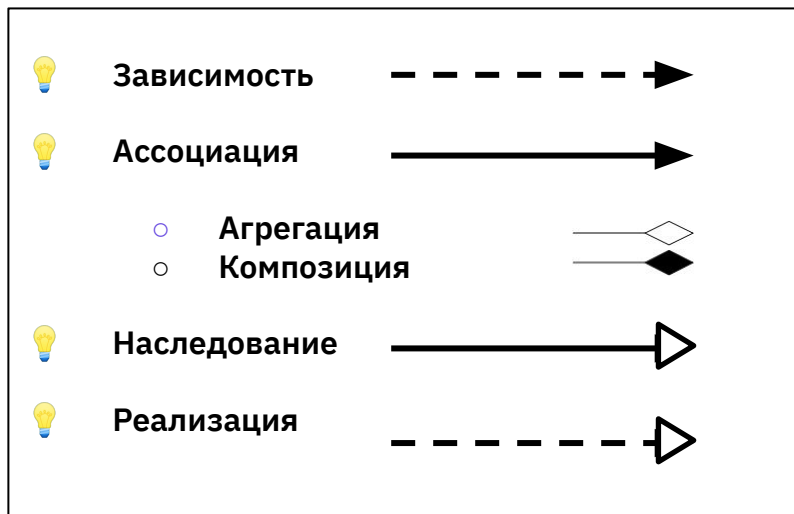
инверсии зависимостей (Dependency inversion)

Зависимости должны строиться относительно абстракций, а не деталей



SOLID — это пять основных принципов проектирования в объектно-ориентированном программировании

Строительные блоки модели предметной области позволяют построить информативную модель без серьезных трудозатрат





Наследование - это механизм системы, который позволяет наследовать одними классами свойства и поведение других классов





Переопределение - это нюанс реализации наследования, позволяющий не дополнять поведение родительского класса, а модифицировать





Множественное наследование -
это такой вариант наследования,
когда свойства и поведение
наследуются от нескольких
классов.





Полиморфизм - это свойство системы, позволяющее иметь множество реализаций одного интерфейса.





Принцип полиморфизма позволяет добавлять новые реализации не меняя порядок взаимодействия с потребителем.



Инкапсуляция – принцип обеспечения контроля доступа к полям и методам объекта.





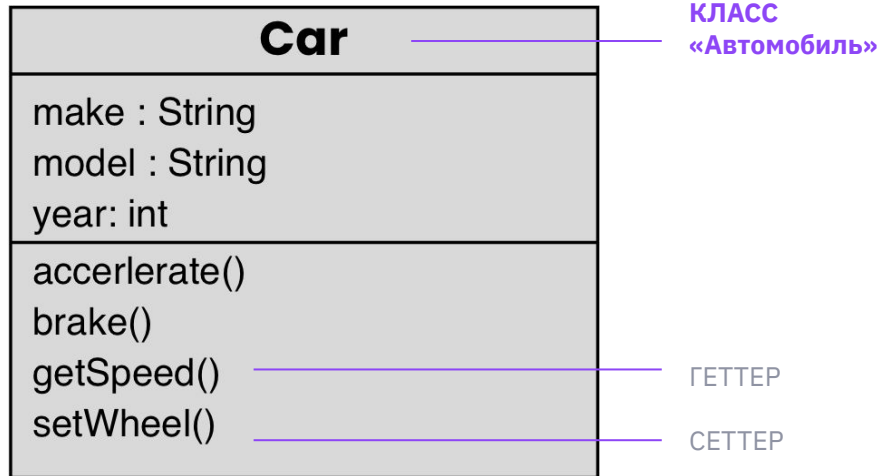
Конструкции объектно-ориентированного программирования

Во многих языках частью инкапсуляции является сокрытие данных. Для этого существуют модификаторы доступа:

`public` — к атрибуту может получить доступ любой желающий

`private` — к атрибуту могут обращаться только методы данного класса

`protected` — то же, что и `private`, только доступ получают и наследники класса, в том числе





Геттеры и сеттеры (ацессоры, англ. Accessors) — это методы, задача которых контролировать доступ к полям.





В некоторых языках
программирования ацессоры
принято маскировать под свойства,
что делает доступ прозрачным для
внешнего кода



Интерфейсы – декларативный контракт обязывающий всех участников поддерживать определенный протокол взаимодействия.





Интерфейсы

Интерфейс очень похож на абстрактный класс, но является не классом, а просто пустышкой с перечислением абстрактных методов (без имплементации).

Интерфейс имеет множественность реализаций.

У интерфейса двустороннее применение:

- По одну сторону интерфейса — классы, имплементирующие данный интерфейс.
- По другую сторону — потребители, которые используют этот интерфейс в качестве описания типа данных, с которым они (потребители) работают.





Другие актуальные принципы
разработки о которых должен знать
аналитик



KISS (акроним для «Keep it simple, stupid» — «Делай проще, тупее») — принцип проектирования, принятый в ВМС США в 1960 году.





В некоторых языках
программирования ацессоры
принято маскировать под свойства,
что делает доступ прозрачным для
внешнего кода



DRY (англ. dry — сухой, сушить) — один из основополагающих принципов разработки. Он расшифровывается как Don't repeat yourself — «не повторяйтесь».





Когда разработчик пишет
программный код, он должен
думать о том, как можно
переиспользовать тот или иной
фрагмент, но и аналитик работает
по тому же принципу



YAGNI - Расшифровывается как
You ain't gonna need it («Вам это не
понадобится») и говорит о том, что
не нужно включать в проект
реализацию излишних функций.





Создавать какую-то функциональность следует только тогда, когда она действительно нужна. Не нужно проектировать и разрабатывать впрок, так как требованиям свойственны изменения



Среды разработки

IDE (от англ. Integrated Development Environment, «интегрированная среда разработки») — это программа, в которой разработчики пишут, проверяют, тестируют и запускают код, а также ведут большие проекты.

По стоимости среды IDE делятся на:

- Открытые, то есть полностью бесплатные.
- Условно-бесплатные. Обычно их можно скачать, но за расширенные функции придётся доплачивать.
- Полностью платные, то есть требующие покупки лицензии или подписки. Такими чаще всего пользуются компании, хотя иногда разработчики покупают их и для себя, если ведут крупные личные проекты.

По универсальности среды IDE делятся на:

- Одноязычные. Поддерживают только один конкретный язык программирования и оптимизированы именно для него.
- Мультиязычные. Поддерживают, конечно, не все, но многие языки программирования.



Соглашение по разработке

Рекомендуется оформлять требования соглашения реализующих соответствующие правила хорошего стиля в виде конфигураций статических анализаторов для IDE.

Так например, соглашение описывающее качество программного кода может содержать следующие дополнительные пункты:

- в структуре проекта не должны присутствовать неиспользуемые в процессах производства технические каталоги и файлы;
- должна обеспечиваться независимость разработки, развертывания и масштабирования компонентов;
- должна быть обеспечена простота интеграции и взаимодействия компонентов;
- должна быть обеспечена устойчивость к отказам;
- обработку исключений желательно производить как можно более частными способами;
- должна быть обеспечена независимость развертывания компонентов, исполнение компонентов в отдельных потоках исполнения;
- ...

Чистая архитектура

В целом при реализации системы рекомендуется придерживаться концепций Чистой архитектуры (Clean Architecture). Функционирование и взаимодействие объектов не должно нарушать установленных архитектурных правил.

К основным принципам Clean Architecture можно отнести разделение на уровни:

- Уровень домена данных (Domain)
- Уровень приложения (Application)
- Уровень представления (Presentation)
- Уровень инфраструктуры (Infrastructure)





Чтобы избежать ошибки смешения
уровней архитектуры всегда
необходимо определять границы
между уровнями архитектуры и
придерживаться их при
проектировании и написании кода



В целом можно сказать, что
компоненты разрабатываемой
системы должны иметь низкую
связность и высокую
согласованность



Состояние гонки (англ. race condition) — ошибка проектирования при которой работа системы или приложения зависит от того, в каком порядке выполняются части кода





Соглашение по разработке прежде всего призвано защитить систему от типовых стандартных ошибок, таких как утечки памяти, порча данных, взаимные блокировки и им подобные



Погруженность системного аналитика в процессы и соглашения о разработке окупается адекватной реализацией проектируемой системы



В заключение

Мы познакомились с понятиями структурного проектирования и узнали о важных принципах проектирования, таких как SOLID, KISS, DRY, YAGNI, Clean Architecture, которые эффективно применяются на практике, узнали о важности соглашения по разработке.

В следующей лекции мы изучим аспекты интеграционного проектирования, позволяющего проектировать сложные взаимосвязанные системы

