ООП

+ Класс: Описывает модель объекта, его свойства и поведение. Говоря языком программиста, класс — такой тип данных, который создается для описания сложных объектов.

+ Объект: Хранит конкретные значения свойств и информацию о принадлежности к классу. Может выполнять методы.

+ Атрибут: Свойство, присущее объекту. Класс объекта определяет, какие атрибуты есть у объекта. Конкретные значения атрибутов — характеристика уже не класса, а конкретного экземпляра этого класса, то есть объекта.

+ Метод: Действие (функция внутри класса), которое объект может выполнять над самим собой или другими объектами.

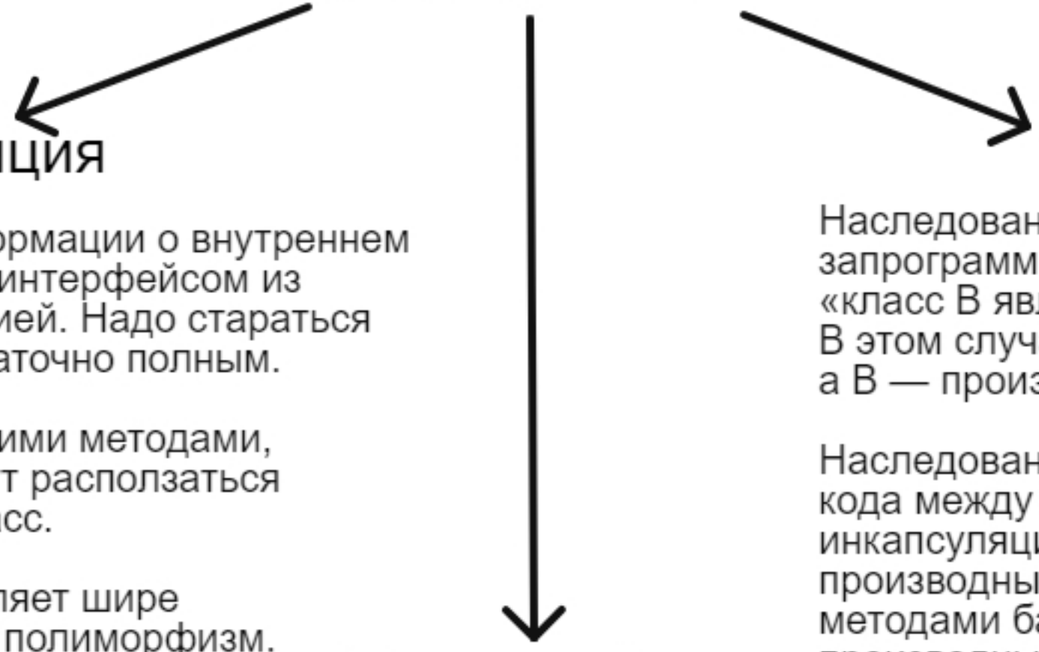
+ Экземпляр класса (Instance): Отдельный объект-представитель определенного класса.

+ Наследование (Inheritance): Передача аттрибутов и методов родительского класса дочерним классам.

+ Перегрузка функций (Function overloading): Изменение работы метода, унаследованного дочерним классом от родительского класса.

+ Перегрузка операторов (Operator overloading): Определение работы операторов с экземплярами данного класса.

3 кита ООП



Инкапсуляция

Такая технология сокрытия информации о внутреннем устройстве объекта за внешним интерфейсом из методов называется инкапсуляцией. Надо стараться делать интерфейс методов достаточно полным.

Тогда вы будете пользоваться этими методами, а изменения в атрибутах не будут расползаться по коду, использующему ваш класс.

Кроме того, инкапсуляция позволяет шире использовать такое понятие, как полиморфизм.

Наследование

Наследование — механизм, позволяющий запрограммировать отношение вида «класс B является частным случаем класса A». В этом случае класс A называется базовым классом, а B — производным классом.

Наследование является способом переиспользования кода между классами без необходимости нарушения инкапсуляции. Это достигается за счет того, что производный класс может пользоваться атрибутами и методами базового класса (иными словами, производный класс наследует атрибуты и методы базового класса).

Полиморфизм

Свойство кода работать с разными типами данных.

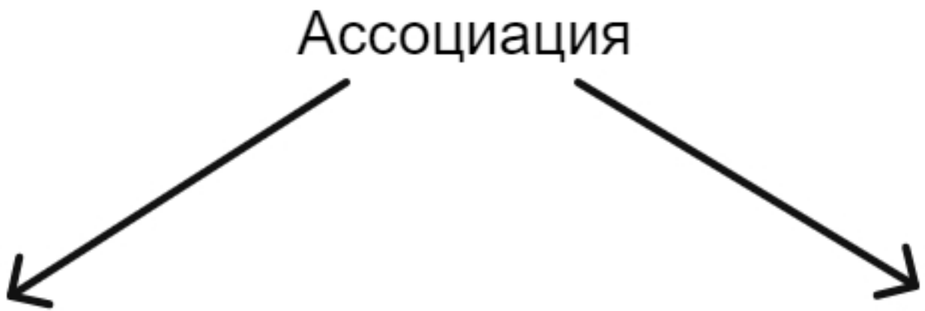
Например, оператор + является полиморфным.

ООП

Один класс включает в себя другой класс в качестве одного из полей.

Ассоциация описывается словом «имеет». Автомобиль имеет двигатель. Вполне естественно, что он не будет являться наследником двигателя (хотя такая архитектура тоже возможна в некоторых ситуациях).

Выделяют два частных случая ассоциации: композицию и агрегацию.



Композиция

Когда двигатель не существует отдельно от автомобиля. Он создается при создании автомобиля и полностью управляется автомобилем. В типичном примере, экземпляр двигателя будет создаваться в конструкторе автомобиля.

Агрегация

Когда экземпляр двигателя создается где-то в другом месте кода, и передается в конструктор автомобиля в качестве параметра.

ООП

в Python возможны следующие варианты доступа к данным:

attribute (без одного или двух подчеркиваний вначале) – публичное свойство (public);

_attribute (с одним подчеркиванием) – режим доступа protected (служит для обращения внутри класса и во всех его дочерних классах)

__attribute (с двумя подчеркиваниями) – режим доступа private (служит для обращения только внутри класса).

Шаблоны проектирования

Паттерн проектирования — это часто встречающееся решение определённой проблемы при проектировании архитектуры программ.

Зачем нужны Паттерны?

+ Проверенные решения.
  Вы тратите меньше времени, используя готовые решения,
  вместо повторного изобретения велосипеда.
  До некоторых решений вы смогли бы додуматься и сами,
  но многие могут быть для вас открытием.

+ Стандартизация кода.
  Вы делаете меньше просчётов при проектировании,
  используя типовые унифицированные решения,
  так как все скрытые проблемы в них уже давно найдены.

+ Общий программистский словарь.
  Вы произносите название паттерна, вместо того,
  чтобы час объяснять другим программистам,
  какой крутой дизайн вы придумали и
  какие классы для этого нужны.

Шаблоны проектирования

Классификация паттернов

Паттерны отличаются по:

+ уровню сложности,
+ детализации и
+ охвата проектируемой системы.

Идиомы - самые низкоуровневые и простые паттерны. Они не универсальны, поскольку применимы только в рамках одного языка программирования.

Архитектурные паттерны - самые универсальные. Их можно реализовать практически на любом языке. Они нужны для проектирования всей программы, а не отдельных её элементов.

3 основные группы паттернов:

+ Порождающие паттерны
  беспокоятся о гибком создании объектов
  без внесения в программу лишних зависимостей.

+ Структурные паттерны
  показывают различные способы построения связей между объектами.

+ Поведенческие паттерны
  заботятся об эффективной коммуникации между объектами.

Шаблоны проектирования

Порождающие

Отвечают за удобное и безопасное создание новых объектов или даже целых семейств объектов.

Фабричный метод — определяет общий интерфейс для создания объектов в суперклассе, позволяя подклассам изменять тип создаваемых объектов.

Абстрактная фабрика — позволяет создавать семейства связанных объектов, не привязываясь к конкретным классам создаваемых объектов.

Строитель — создаёт сложные объекты пошагово. Строитель даёт возможность использовать один и тот же код строительства для получения разных представлений объектов. Позволяет собирать объекты пошагово, вызывая только те шаги, которые вам нужны. А значит, больше не нужно пытаться «запихнуть» в конструктор все возможные опции продукта.

Прототип — позволяет копировать объекты, не вдаваясь в подробности их реализации.

Одиночка — гарантирует, что у класса есть только один экземпляр, и предоставляет к нему глобальную точку доступа.

Простая фабрика — это класс, в котором есть один метод с большим условным оператором, выбирающим создаваемый продукт. Этот метод вызывают с неким параметром, по которому определяется какой из продуктов нужно создать. У простой фабрики, обычно, нет подклассов.

Структурные

Отвечают за построение удобных в поддержке иерархий классов.

Адаптер — позволяет объектам с несовместимыми интерфейсами работать вместе.

Мост — разделяет один или несколько классов на две отдельные иерархии — абстракцию и реализацию, позволяя изменять их независимо друг от друга.

Компоновщик — позволяет сгруппировать множество объектов в древовидную структуру, а затем работать с ней так, как будто это единичный объект.

Декоратор — позволяет динамически добавлять объектам новую функциональность, оборачивая их в полезные «обёртки».

Фасад — предоставляет простой интерфейс к сложной системе классов, библиотеке или фреймворку.

Легковес — позволяет вместить бóльшее количество объектов в отведённую оперативную память. Легковес экономит память, разделяя общее состояние объектов между собой, вместо хранения одинаковых данных в каждом объекте.

Заместитель - позволяет подставлять вместо реальных объектов специальные объекты-заменители. Эти объекты перехватывают вызовы к оригинальному объекту, позволяя сделать что-то до или после передачи вызова оригиналу.

Поведенческие

Решают задачи эффективного и безопасного взаимодействия между объектами программы.

Цепочка обязанностей — позволяет передавать запросы последовательно по цепочке обработчиков. Каждый последующий обработчик решает, может ли он обработать запрос сам и стоит ли передавать запрос дальше по цепи.

Команда — превращает запросы в объекты, позволяя передавать их как аргументы при вызове методов, ставить запросы в очередь, логировать их, а также поддерживать отмену операций.

Итератор — даёт возможность последовательно обходить элементы составных объектов, не раскрывая их внутреннего представления.

Посредник — позволяет уменьшить связанность множества классов между собой, благодаря перемещению этих связей в один класс-посредник.

Снимок — позволяет сохранять и восстанавливать прошлые состояния объектов, не раскрывая подробностей их реализации.

Наблюдатель — создаёт механизм подписки, позволяющий одним объектам следить и реагировать на события, происходящие в других объектах.

Состояние — позволяет объектам менять поведение в зависимости от своего состояния. Извне создаётся впечатление, что изменился класс объекта.

Стратегия — определяет семейство схожих алгоритмов и помещает каждый из них в собственный класс, после чего алгоритмы можно взаимозаменять прямо во время исполнения программы.

Шаблонный метод — определяет скелет алгоритма, перекладывая ответственность за некоторые его шаги на подклассы. Паттерн позволяет подклассам переопределять шаги алгоритма, не меняя его общей структуры.

Посетитель — позволяет добавлять в программу новые операции, не изменяя классы объектов, над которыми эти операции могут выполняться.

Page Object Model

"Модель Page Object" – это паттерн/шаблон проектирования, который извлекает взаимодействия веб-страницы с целью улучшить их читабельность и повторное использование.

BitPaper
Share Knowledge

Непосредственная цель проектирования: собрать в одном месте все то, что будет меняться.
Поместите вещи, которые изменяются по другой причине в другое время, в другое место.
То, что не меняется вообще, поместите в третье место.

Шаблон Page Object Model изолирует несколько типов изменений, самым значительным и очевидным из которых является интерфейс между кодом ваших тестов и DOM приложения.
Тесты пользовательского интерфейса должны содержать информацию о том, как находить элементы на страницах.
Эта информация имеет тенденцию меняться, и она имеет тенденцию меняться на разных страницах.

DOM (Document Object Model
— «объектная модель документа»)
— это независящий от платформы и языка программный интерфейс, позволяющий программам и скриптам получить доступ к содержимому документов

Страницы представлены как классы с атрибутами локаторов и методами взаимодействия.
Вместо сырых вызовов WebDriver тесты вызывают методы объекта страницы.

Объекты страниц (Page Objects) собирают и инкапсулируют эти знания в одном месте.
Они говорят: «Подождите все, не беспокойтесь о том, КАК найти элемент x на странице y, я владею всеми знаниями о поиске элементов на странице Y. Я ваш эксперт по поиску на Y!».

Когда страница Y изменится, любой, кому нужно будет обновить эти знания, будет точно знать, куда идти, поскольку вся логика поиска вещей должна существовать в объекте страницы Y.
Эти знания не должны существовать где-либо еще.