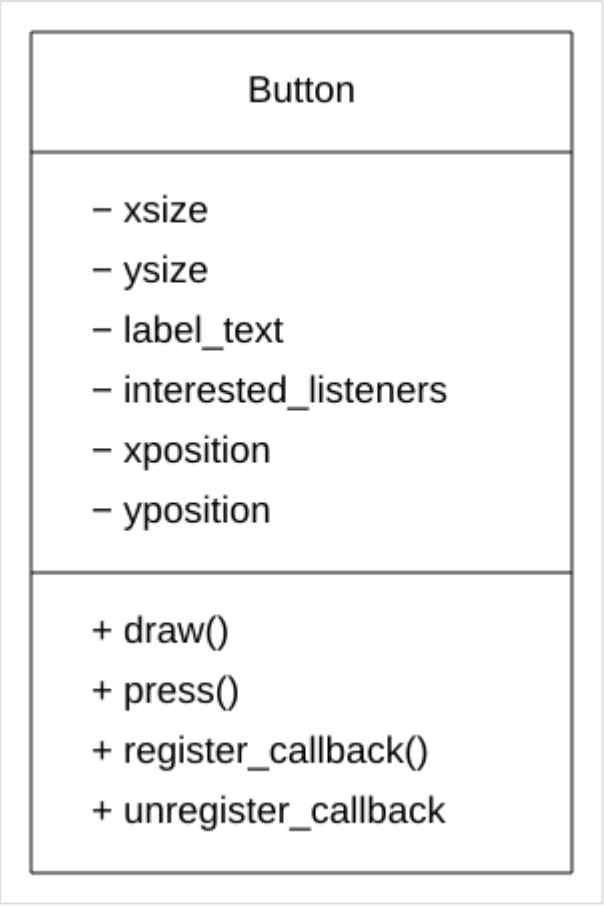
Объектно-ориентированное программирование

Объектно-ориентированное программирование (**ООП**) — это парадигма программирования , основанная на концепции *объектов* . [1] Объекты могут содержать данные (называемые полями , атрибутами или свойствами) и иметь действия, которые они могут выполнять (называемые процедурами или методами и реализованные в коде). В ООП компьютерные программы разрабатываются путем создания их из объектов, которые взаимодействуют друг с другом. [2][3]

Многие из наиболее широко используемых языков программирования (такие как C++ , Java , [4] и Python) поддерживают объектно-ориентированное программирование в большей или меньшей степени, как правило, как часть нескольких парадигм в сочетании с другими, такими как императивное программирование и декларативное программирование .

Значимые объектно-ориентированные языки включают Ada , ActionScript , C++ , Common Lisp , C# , Dart , Eiffel , Fortran 2003 , Haxe , Java , [4] JavaScript , Kotlin , Logo , MATLAB , Objective-C , Object Pascal , Perl , PHP , Python , R , Raku , Ruby , Scala , SIMSCRIPT , Simula , Smalltalk , Swift , Vala и Visual Basic.NET .



Нотация UML для класса. Этот класс Button имеет переменные для данных и функции . С помощью наследования можно создать подкласс как подмножество класса Button. Объекты являются экземплярами класса.

История

Идея «объектов» в программировании зародилась в группе искусственного интеллекта в Массачусетском технологическом институте в конце 1950-х и начале 1960-х годов. Здесь «объект» относился к атомам LISP с идентифицированными свойствами (атрибутами). [5][6] Другим ранним примером был Sketchpad, созданный Иваном Сазерлендом в Массачусетском технологическом институте в 1960–1961 годах. В глоссарии своего технического отчета Сазерленд определил такие термины, как «объект» и «экземпляр» (с концепцией класса, охватываемой «мастером» или «определением»), хотя и специализированной для графического взаимодействия. [7] Позже, в 1968 году, AED-0, версия языка программирования ALGOL от Массачусетского технологического

института, связала структуры данных («плексы») и процедуры, предвосхищая то, что позже было названо «сообщениями», «методами» и «функциями-членами». [8] [9] Такие темы, как абстракция данных и модульное программирование, были обычными темами для обсуждения в то время.

Тем временем в Норвегии Simula разрабатывалась в 1961–1967 годах. [8] Simula представила основные объектно-ориентированные идеи, такие как классы , наследование и динамическое связывание . [10] Simula в основном использовалась исследователями, занимающимися физическим моделированием , например, движением судов и их содержимого через грузовые порты. [10] Simula обычно считается первым языком с основными функциями и структурой объектно-ориентированного языка. [11]

> Я представлял себе объекты как биологические клетки и/или отдельные компьютеры в сети, способные общаться только с помощью сообщений (поэтому обмен сообщениями появился в самом начале — потребовалось некоторое время, чтобы понять, как осуществлять обмен сообщениями на языке программирования достаточно эффективно, чтобы быть полезным).
>
> Алан Кей, [1]

Под влиянием MIT и Simula, Алан Кей начал разрабатывать свои собственные идеи в ноябре 1966 года. Он продолжил работу над созданием Smalltalk , влиятельного объектно-ориентированного языка программирования. К 1967 году Кей уже использовал термин «объектно-ориентированное программирование» в разговоре. [1] Хотя его иногда называют «отцом» объектно-ориентированного программирования, [12] Кей сказал, что его идеи отличаются от того, как обычно понимают объектно-ориентированное программирование, и намекнул, что истеблишмент компьютерной науки не принял его идею. [1] В меморандуме MIT 1976 года, соавтором которого была Барбара Лисков, Simula 67 , CLU и Alphard указаны как объектно-ориентированные языки, но Smalltalk не упоминается. [13]

В 1970-х годах в Xerox PARC Аланом Кеем , Дэном Ингаллсом и Адель Голдберг была разработана первая версия языка программирования Smalltalk . Smalltalk-72 был примечателен использованием объектов на уровне языка и графической средой разработки. [14] Smalltalk был полностью динамической системой, позволяющей пользователям создавать и изменять классы по мере работы. [15] Большая часть теории ООП была разработана в контексте Smalltalk, например множественное наследование. [16]

В конце 1970-х и 1980-х годах объектно-ориентированное программирование приобрело известность. Объектно-ориентированный Lisp Flavors разрабатывался с 1979 года, представляя множественное наследование и миксины . [17] В августе 1981 года журнал Byte Magazine выделил Smalltalk и ООП, представив эти идеи широкой аудитории. [18] LOOPS, объектная система для Interlisp -D, находилась под влиянием Smalltalk и Flavors, и статья о ней была опубликована в 1982 году. [19] В 1986 году первая *конференция по объектно-ориентированному программированию, системам, языкам и приложениям* (OOPSLA) посетила 1000 человек. Эта конференция ознаменовала начало усилий по консолидации объектных систем Lisp, что в конечном итоге привело к созданию Common Lisp Object System . В 1980-х годах было предпринято несколько попыток разработать архитектуры процессоров , которые включали бы аппаратную поддержку объектов в памяти , но они не увенчались успехом. Примеры включают Intel iAPX 432 и Linn Smart Rekursiv .

В середине 1980-х годов появились новые объектно-ориентированные языки, такие как Objective-C , C++ и Eiffel . Objective-C был разработан Брэдом Коксом , который использовал Smalltalk в ITT Inc. Бьярне Страуструп создал C++ на основе своего опыта использования Simula для своей докторской

диссертации. [14] Бертран Мейер создал первый проект языка Eiffel в 1985 году, который был сосредоточен на качестве программного обеспечения с использованием подхода «проектирование по контракту» . [20]

В 1990-х годах объектно-ориентированное программирование стало основным способом программирования, особенно с появлением большего количества языков, поддерживающих его. К ним относятся Visual FoxPro 3.0, [21] [22] C++ , [23] и Delphi . ООП стало еще более популярным с появлением графических пользовательских интерфейсов , которые использовали объекты для кнопок, меню и других элементов. Одним из известных примеров является фреймворк Cocoa от Apple, используемый в Mac OS X и написанный на Objective-C . Инструментарии ООП также повысили популярность событийно-управляемого программирования .

В ETH Zürich Никлаус Вирт и его коллеги создали новые подходы к ООП. Modula-2 (1978) и Oberon (1987) включали отличительный подход к объектной ориентации, классам и проверке типов через границы модулей. Наследование не очевидно в дизайне Вирта , поскольку его номенклатура смотрит в противоположном направлении: это называется расширением типа, и точка зрения направлена от родителя к наследнику.

Многие языки программирования, существовавшие до ООП, добавили объектно-ориентированные возможности, включая Ada , BASIC , Fortran , Pascal и COBOL . Иногда это вызывало проблемы совместимости и поддержки, поскольку эти языки изначально не были разработаны с учетом ООП.

В новом тысячелетии появились новые языки, такие как Python и Ruby , которые сочетают объектно-ориентированный и процедурный стили. Наиболее коммерчески важными «чистыми» объектно-ориентированными языками по-прежнему остаются Java , разработанный Sun Microsystems , а также C# и Visual Basic.NET (VB.NET), оба разработанные для платформы .NET от Microsoft . Эти языки демонстрируют преимущества ООП, создавая абстракции от реализации. Платформа .NET поддерживает кросс-языковое наследование, позволяя программам использовать объекты из нескольких языков вместе.

Функции

Объектно-ориентированное программирование фокусируется на работе с объектами, но не все языки ООП имеют все функции, связанные с ООП. Ниже приведены некоторые общие функции языков, которые считаются сильными в ООП или поддерживают его вместе с другими стилями программирования. Также отмечены важные исключения. [24] [25] [26] [27] Кристофер Дж. Дейт указал, что сравнивать ООП с другими стилями, такими как реляционное программирование , сложно, поскольку нет четкого, согласованного определения ООП. [28]

Императивное программирование

Возможности императивного и структурного программирования присутствуют в ООП-языках, а также встречаются в не-ООП-языках.

- Переменные содержат различные типы данных , такие как целые числа , строки , списки и хэш-таблицы . Некоторые типы данных являются встроенными, а другие являются результатом объединения переменных с использованием указателей памяти .
- Процедуры – также известные как функции, методы, подпрограммы или подпрограммы – принимают входные данные, генерируют выходные данные и работают с данными.

> Современные языки включают в себя структурированные программные конструкции, такие как циклы и условные операторы .

Поддержка модульного программирования позволяет программистам организовывать связанные процедуры в файлы и модули. Это упрощает управление программами. Каждый модуль имеет свое собственное пространство имен , поэтому элементы в одном модуле не будут конфликтовать с элементами в другом.

Объектно-ориентированное программирование (ООП) было создано для того, чтобы сделать код более простым для повторного использования и поддержки . [29] Однако оно не было разработано для того, чтобы четко показывать поток инструкций программы — это было оставлено компилятору. Поскольку компьютеры начали использовать больше параллельной обработки и многопоточности , стало более важным понимать и контролировать поток инструкций. Это трудно сделать с ООП. [30] [31] [32] [33]

Объекты

Объект — это тип структуры данных , которая состоит из двух основных частей: полей и методов . Поля также могут быть известны как члены, атрибуты или свойства и содержат информацию в форме переменных состояния . Методы — это действия, подпрограммы или процедуры, определяющие поведение объекта в коде. Объекты обычно хранятся в памяти , и во многих языках программирования они работают как указатели , которые напрямую ссылаются на непрерывный блок, содержащий данные экземпляров объекта.

Объекты могут содержать другие объекты. Это называется композицией объектов . Например, объект Employee может иметь внутри себя объект Address вместе с другой информацией, такой как «first_name» и «position». Этот тип структур показывает отношения «has-a», например «у сотрудника есть адрес».

Некоторые считают, что ООП уделяет слишком много внимания использованию объектов, а не алгоритмам и структурам данных . [34] [35] Например, программист Роб Пайк отметил, что ООП может заставить программистов больше думать об иерархии типов, чем о композиции. [36] Он назвал объектно-ориентированное программирование « римскими цифрами вычислений». [37] Рич Хики , создатель Clojure , описал ООП как чрезмерно упрощенное, особенно когда речь идет о представлении реальных вещей, которые меняются со временем. [35] Александр Степанов сказал, что ООП пытается втиснуть все в один тип, что может быть ограничивающим. Он утверждал, что иногда нам нужны многосортные алгебры — семейства интерфейсов, которые охватывают несколько типов, например, в обобщенном программировании . Степанов также сказал, что называние всего «объектом» не добавляет много понимания. [34]

Моделирование реального мира и взаимоотношений

Иногда объекты представляют собой реальные вещи и процессы в цифровой форме. [38] Например, графическая программа может иметь такие объекты, как «круг», «квадрат» и «меню». Система онлайн-покупок может иметь такие объекты, как «корзина», «клиент» и «продукт». Никлаус Вирт сказал: «Эта парадигма [ООП] точно отражает структуру систем в реальном мире и поэтому хорошо подходит для моделирования сложных систем со сложным поведением». [39]

Однако чаще объекты представляют собой абстрактные сущности, такие как открытый файл или конвертер единиц. Не все согласны с тем, что ООП позволяет легко точно копировать реальный мир или что это вообще необходимо. Боб Мартин предполагает, что поскольку классы являются

программным обеспечением, их отношения не соответствуют отношениям реального мира, которые они представляют. [40] Бертран Мейер в книге *«Конструирование объектно-ориентированного программного обеспечения»* утверждает , что программа — это не модель мира, а модель некоторой части мира; «Реальность — это дважды удаленная кузина». [41] Стив Йегге отметил, что естественным языкам не хватает подхода ООП, который строго расставляет приоритеты между *вещами* (объектами/ существительными) и *действиями* (методами/ глаголами), в отличие от функционального программирования , которое делает наоборот. [42] Иногда это может сделать решения ООП более сложными, чем те, которые написаны в процедурном программировании. [43]

Наследование

Большинство языков ООП допускают повторное использование и расширение кода посредством « наследования ». Это наследование может использовать либо « классы », либо « прототипы », которые имеют некоторые различия, но используют похожие термины для таких идей, как «объект» и «экземпляр».

На основе классов

В программировании на основе классов , наиболее распространенном типе ООП, каждый объект является экземпляром определенного *класса* . Класс определяет формат данных, например переменные (например, имя, возраст) и методы (действия, которые может выполнять объект). Каждый экземпляр класса имеет тот же набор переменных и методов. Объекты создаются с помощью специального метода в классе, известного как конструктор .

Вот несколько ключевых терминов ООП на основе классов:

- Переменные класса – принадлежат *самому классу* , поэтому все объекты в классе имеют одну копию.
- Переменные экземпляра – принадлежат отдельным *объектам* ; каждый объект имеет свою собственную версию этих переменных.
- Переменные-члены — относятся как к переменным класса, так и к переменным экземпляра, которые определены конкретным классом.
- Методы класса — связаны с *самим классом* и могут использовать только переменные класса.
- Методы экземпляра – принадлежат *отдельным объектам* и могут использовать как переменные экземпляра, так и переменные класса.

Классы могут наследовать от других классов, создавая иерархию «подклассов». Например, класс «Employee» может наследовать от класса «Person». Это означает, что объект Employee будет иметь все переменные от Person (например, переменные имени) плюс любые новые переменные (например, должность и зарплата). Аналогично, подкласс может расширять интерфейс новыми методами. Большинство языков также позволяют подклассу переопределять *методы* , определенные суперклассами. Некоторые языки поддерживают множественное наследование , когда класс может наследовать от более чем одного класса, а другие языки аналогично поддерживают миксины или черты . Например, миксин с именем UnicodeConversionMixin может добавить метод unicode_to_ascii() как к классу FileReader, так и к классу WebPageScraper.

Некоторые классы являются абстрактными , то есть они не могут быть напрямую инстанциированы в объекты; они предназначены только для наследования в другие классы. Другие классы являются *служебными* классами, которые содержат только переменные и методы класса и не предназначены для инстанцирования или подклассификации. [44]

На основе прототипа

В прототипном программировании нет никаких классов. Вместо этого каждый объект связан с другим объектом, называемым его *прототипом* или *родителем* . В Self объект может иметь несколько или ни одного родителя, [45] но в самом популярном прототипном языке, Javascript, каждый объект имеет ровно одну ссылку *на прототип* , вплоть до базового типа Object, прототип которого равен null.

Прототип выступает в качестве модели для новых объектов. Например, если у вас есть объект *фрукт* , вы можете создать два объекта *яблоко* и *апельсин на его основе. Класса фрукт* нет , но они разделяют черты прототипа *фрукта* . Языки на основе прототипов также позволяют объектам иметь свои собственные уникальные свойства, поэтому объект *яблоко* может иметь атрибут *sugar_content* , в то время как объекты *апельсин* или *фрукт* не имеют.

Нет наследства

Некоторые языки, такие как Go , вообще не используют наследование. [46] Вместо этого они поощряют « композицию вместо наследования », где объекты строятся с использованием более мелких частей вместо родительско-дочерних отношений. Например, вместо наследования от класса Person класс Employee может просто содержать объект Person. Это позволяет классу Employee контролировать, какую часть Person он раскрывает другим частям программы. Делегирование — еще одна языковая функция, которая может использоваться как альтернатива наследованию.

Программисты по-разному относятся к наследованию. Бьярне Страуструп, автор C++, заявил, что ООП можно реализовать без наследования. [47] Роб Пайк критиковал наследование за создание сложных иерархий вместо более простых решений. [48]

Наследование и поведенческое подтипирование

Люди часто думают, что если один класс наследует от другого, это означает, что подкласс " является " более конкретной версией исходного класса. Это предполагает, что семантика программы заключается в том, что объекты из подкласса всегда могут заменить объекты из исходного класса без проблем. Эта концепция известна как поведенческое подтипирование , а точнее принцип подстановки Лисков .

Однако это часто не соответствует действительности, особенно в языках программирования, которые допускают изменяемые объекты, объекты, которые изменяются после их создания. Фактически, полиморфизм подтипов , обеспечиваемый проверкой типов в языках ООП, не может гарантировать поведенческое подтипирование в большинстве, если не во всех контекстах. Например, общеизвестно, что проблема круга-эллипса сложна для обработки с использованием концепции наследования ООП. Поведенческое подтипирование в общем случае неразрешимо, поэтому его нельзя легко реализовать компилятором. Из-за этого программисты должны тщательно проектировать иерархии классов, чтобы избежать ошибок, которые сам язык программирования не может отловить.

Динамическая отправка

Когда метод вызывается для объекта, сам объект — а не внешний код — решает, какой конкретный код запустить. Этот процесс, называемый динамической диспетчеризацией , обычно происходит во время выполнения путем проверки таблицы, связанной с объектом, для поиска правильного метода. В этом контексте вызов метода также известен как передача сообщений , то есть имя метода и его

входные данные подобны сообщению, отправленному объекту для выполнения действия. Если выбор метода зависит от более чем одного типа объекта (например, других объектов, переданных в качестве параметров), это называется множественной диспетчеризацией .

Динамическая диспетчеризация работает совместно с наследованием: если у объекта нет запрошенного метода, он обращается к своему родительскому классу (делегирование) и продолжает движение вверх по цепочке, пока не найдет метод или не достигнет вершины.

Абстракция и инкапсуляция данных

Абстракция данных — это способ организации кода таким образом, что только определенные части данных видны связанным функциям (скрытие данных). Это помогает предотвратить ошибки и упрощает управление программой. Поскольку абстракция данных работает хорошо, многие стили программирования, такие как объектно-ориентированное программирование и функциональное программирование, используют ее в качестве ключевого принципа. Инкапсуляция — еще одна важная идея в программировании. Она означает сохранение внутренних деталей объекта скрытыми от внешнего кода. Это упрощает изменение того, как объект работает внутри, не затрагивая другие части программы, например, при рефакторинге кода . Инкапсуляция также помогает сохранять связанный код вместе (разъединение), что упрощает его понимание программистами.

В объектно-ориентированном программировании объекты действуют как барьер между их внутренней работой и внешним кодом. Внешний код может взаимодействовать с объектом только путем вызова определенных *публичных* методов или переменных. Если класс разрешает доступ к своим данным только через методы, а не напрямую, это называется сокрытием информации . При разработке программы часто рекомендуется сохранять данные максимально скрытыми. Это означает использование локальных переменных внутри функций, когда это возможно, затем закрытых переменных (которые может использовать только объект) и, наконец, открытых переменных (к которым может получить доступ любая часть программы) при необходимости. Скрытие данных помогает предотвратить проблемы при последующем изменении кода. [49] Некоторые языки программирования, такие как Java, управляют сокрытием информации, помечая переменные как `private`(скрытые) или `public`(доступные). [50] Другие языки, такие как Python, полагаются на соглашения об именовании, такие как начало имени закрытого метода с подчеркивания. Существуют также промежуточные уровни доступа, такие как ключевое слово Java `protected`(которое позволяет получить доступ из того же класса и его подклассов, но не объектов другого класса), а также `internal`ключевое слово в C#, Swift и Kotlin, которое ограничивает доступ к файлам в пределах одного модуля. [51]

Абстракция и сокрытие информации являются важными концепциями в программировании, особенно в объектно-ориентированных языках. [52] Программы часто создают много копий объектов, и каждая из них работает независимо. Сторонники этого подхода говорят, что он упрощает повторное использование кода и интуитивно представляет реальные ситуации. [53] Однако другие утверждают, что объектно-ориентированное программирование не улучшает читаемость или модульность. [54] [55] Эрик С. Рэймонд написал, что объектно-ориентированные языки программирования, как правило, поощряют программы с толстыми слоями, которые разрушают прозрачность. [56] Рэймонд сравнивает это с подходом, принятым в Unix и языке программирования C , в невыгодном свете . [56]

Один из принципов программирования, называемый « принципом открытости/закрытости », гласит, что классы и функции должны быть «открыты для расширения, но закрыты для модификации». Лука Карделли заявил, что языки ООП имеют «чрезвычайно плохие свойства модульности в отношении

расширения и модификации классов» и, как правило, чрезвычайно сложны. [54] Последний пункт повторяет Джо Армстронг , главный изобретатель Erlang , которого цитируют: [55]

> Проблема объектно-ориентированных языков в том, что у них есть вся эта неявная среда, которую они несут с собой. Вы хотели банан, а получили гориллу, держащую банан и все джунгли.

Лео Броди говорит, что сокрытие информации может привести к копированию одного и того же кода в нескольких местах (дублирование кода), [57] что противоречит правилу «не повторяйся» при разработке программного обеспечения. [58]

Полиморфизм

Полиморфизм — это использование одного символа для представления нескольких различных типов. [59] В объектно-ориентированном программировании полиморфизм более конкретно относится к подтипированию или полиморфизму подтипов, где функция может работать с определенным интерфейсом и, таким образом, манипулировать сущностями различных классов единообразно. [60]

Например, представьте, что у программы есть две фигуры: круг и квадрат. Обе фигуры принадлежат общему классу под названием «Shape». Каждая фигура имеет свой собственный способ рисования. Благодаря полиморфизму подтипов программе не нужно знать тип каждой фигуры, и она может просто вызвать метод «Draw» для каждой фигуры. Среда выполнения языка программирования гарантирует, что для каждой фигуры будет запущена правильная версия метода «Draw». Поскольку детали каждой фигуры обрабатываются внутри их собственных классов, это делает код проще и более организованным, обеспечивая сильное разделение задач .

Открытая рекурсия

В объектно-ориентированном программировании объекты имеют методы, которые могут изменять или использовать данные объекта. Многие языки программирования используют специальное слово, например `this`или `self`, для ссылки на текущий объект. В языках, поддерживающих открытую рекурсию , метод в объекте может вызывать другие методы в том же объекте, включая себя самого, используя это специальное слово. Это позволяет методу в одном классе вызывать другой метод, определенный позже в подклассе, функция, известная как позднее связывание .

ООП языки

Языки ООП можно сгруппировать в различные типы в зависимости от того, как они поддерживают и используют объекты:

- Чистые языки ООП: в этих языках все рассматривается как объект, даже такие базовые вещи, как числа и символы. Они разработаны для полной поддержки и обеспечения ООП. Примеры: Ruby , Scala , Smalltalk , Eiffel , Emerald , [61] JADE , Self , Raku .
- В основном языки ООП: Эти языки фокусируются на ООП, но также включают некоторые функции процедурного программирования. Примеры: Java , Python , C++ , C# , Delphi / Object Pascal , VB.NET .
- Модернизированные языки ООП: изначально они были разработаны для других типов программирования, но позже добавили некоторые функции ООП. Примеры: PHP ,

JavaScript , Perl , Visual Basic (производный от BASIC), MATLAB , COBOL 2002 , Fortran 2003 , ABAP , Ada 95 , Pascal .

- Уникальные языки ООП: Эти языки имеют такие возможности ООП, как классы и наследование, но используют их по-своему. Примеры: Oberon , BETA .
- Объектно-ориентированные языки: поддерживают некоторые идеи ООП, но избегают традиционного наследования на основе классов в пользу прямого манипулирования объектами. Примеры: JavaScript , Lua , Modula-2 , CLU , Go .
- Языки с несколькими парадигмами: поддерживают как ООП, так и другие стили программирования, но ООП не является преобладающим стилем в языке. Примерами служат Tcl , где TclOO допускает как прототипное, так и классовое ООП, и Common Lisp , с его Common Lisp Object System .

Популярность и прием

Многие популярные языки программирования, такие как C++, Java и Python, используют объектно-ориентированное программирование. В прошлом ООП было широко принято, [62] но в последнее время некоторые программисты критикуют его и предпочитают функциональное программирование. [63] Исследование Потока и др. не обнаружило существенной разницы в производительности между ООП и другими методами. [64]

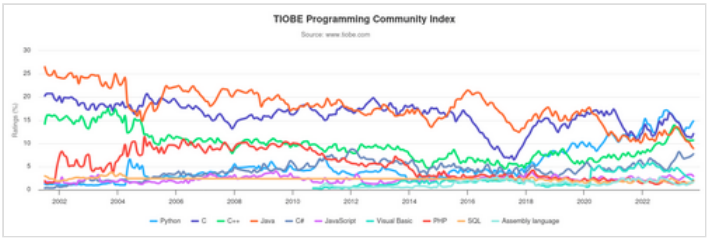


График индекса популярности языков программирования TIOBE с 2002 по 2023 год. В 2000-х годах за первое место боролись объектно-ориентированный Java (оранжевый) и процедурный C (темно-синий).

Пол Грэм , известный компьютерный ученый, считает, что крупные компании любят ООП, потому что это помогает управлять большими командами средних программистов. Он утверждает, что ООП добавляет структуру, затрудняя для одного человека совершение серьезных ошибок, но в то же время сдерживает умных программистов. [65] Эрик С. Рэймонд , программист Unix и сторонник программного обеспечения с открытым исходным кодом , утверждает, что ООП — не лучший способ писать программы. [56]

Ричард Фельдман говорит, что, хотя возможности ООП помогли некоторым языкам оставаться организованными, их популярность обусловлена другими причинами. [66] Лоуренс Крубнер утверждает, что ООП не предлагает особых преимуществ по сравнению с другими стилями, такими как функциональное программирование, и может усложнить кодирование. [67] Лука Карделли говорит, что ООП медленнее и требует больше времени для компиляции, чем процедурное программирование. [54]

ООП в динамических языках

В последние годы объектно-ориентированное программирование (ООП) стало очень популярным в динамических языках программирования . Некоторые языки, такие как Python , PowerShell , Ruby и Groovy , были разработаны с учетом ООП. Другие, такие как Perl , PHP и ColdFusion , начинались как не-ООП языки, но позже добавили возможности ООП (начиная с Perl 5, PHP 4 и ColdFusion версии 6).

В Интернете документы HTML , XHTML и XML используют модель объектов документов (DOM), которая работает с языком JavaScript . JavaScript — известный пример языка, основанного на прототипах . Вместо использования классов, как в других языках ООП, JavaScript создает новые

объекты путем копирования (или «клонирования») существующих. Другой язык, использующий этот метод, — Lua .

ООП в сетевом протоколе

Когда компьютеры взаимодействуют в системе клиент-сервер, они отправляют сообщения для запроса услуг. Например, простое сообщение может включать поле длины (показывающее, насколько велико сообщение), код, который идентифицирует тип сообщения, и значение данных. Эти сообщения могут быть разработаны как структурированные объекты, которые понимают и клиент, и сервер, так что каждый тип сообщения соответствует классу объектов в коде клиента и сервера. Более сложные сообщения могут включать структурированные объекты в качестве дополнительных деталей. Клиент и сервер должны знать, как сериализовать и десериализовать эти сообщения, чтобы их можно было передавать по сети, и сопоставлять их с соответствующими типами объектов. И клиенты, и серверы можно рассматривать как сложные объектно-ориентированные системы.

Архитектура распределенного управления данными (DDM) использует эту идею, организуя объекты на четырех уровнях:

1. Основные сведения о сообщении — такая информация, как длина сообщения, тип и данные.
2. Объекты и коллекции — аналогично тому, как работают объекты в Smalltalk , храня сообщения и их данные.
3. Менеджеры - Подобно файловым каталогам, они организуют и хранят данные, а также предоставляют память и вычислительную мощность. Они похожи на IBM i Objects .
4. Клиенты и серверы — это полноценные системы, включающие менеджеров и обеспечивающие безопасность, службы каталогов и многозадачность.

Первая версия DDM определяла распределенные файловые службы. Позже она была расширена для поддержки баз данных через распределенную реляционную архитектуру баз данных (DRDA).

Шаблоны проектирования

Шаблоны проектирования являются общими решениями проблем в разработке программного обеспечения. Некоторые шаблоны проектирования особенно полезны для объектно-ориентированного программирования, и шаблоны проектирования обычно вводятся в контексте ООП.

Модели объектов

Ниже приведены известные шаблоны проектирования программного обеспечения для объектов ООП. [68]

- Объект функции : класс с одним основным методом, который действует как анонимная функция (в C++ — оператор функции `operator()`).
- Неизменяемый объект : не меняет состояние после создания.
- Первоклассный объект : может использоваться без ограничений.
- Объект-контейнер : содержит другие объекты
- Фабричный объект : создает другие объекты

- Метаобъект : используется для создания других объектов (похож на класс , но является объектом)
- Объект-прототип : специализированный метаобъект, который создает новые объекты путем копирования самого себя.
- Объект Singleton : единственный экземпляр своего класса на протяжении всего жизненного цикла программы.
- Объект фильтра : получает поток данных на входе и преобразует его в выход объекта.

Распространенным антишаблоном является объект «Бог» , объект, который знает или делает слишком много.

Шаблоны проектирования Gang of Four

Design Patterns: Elements of Reusable Object-Oriented Software — известная книга, опубликованная в 1994 году четырьмя авторами: Эрихом Гаммой , Ричардом Хелмом , Ральфом Джонсоном и Джоном Влиссидесом . Люди часто называют их «Бандой четырех». В книге рассказывается о сильных и слабых сторонах объектно-ориентированного программирования и объясняются 23 распространенных способа решения проблем программирования.

Эти решения, называемые «шаблонами проектирования», делятся на три типа:

- *Порождающие шаблоны* (5): Шаблон метода фабрики , Шаблон абстрактной фабрики , Шаблон синглтона , Шаблон строителя , Шаблон прототипа
- *Структурные шаблоны* (7): Шаблон адаптера , шаблон моста , композитный шаблон , шаблон декоратора , шаблон фасада , шаблон облегченного типа , шаблон прокси
- *Поведенческие паттерны* (11): Паттерн «Цепочка ответственности» , Паттерн «Команда» , Паттерн « Интерпретатор » , Паттерн «Итератор » , Паттерн «Посредник» ,, Паттерн « Наблюдатель» , Паттерн «Состояние» , Паттерн «Стратегия» , Паттерн «Шаблонный метод» , Паттерн «Посетитель».

Объектно-ориентированное программирование и базы данных

В современном программном обеспечении широко используются как объектно-ориентированное программирование, так и реляционные системы управления базами данных (СУРБД). Однако реляционные базы данных не хранят объекты напрямую, что создает проблему при их совместном использовании. Эта проблема называется несоответствием объектно-реляционного импеданса .

Для решения этой проблемы разработчики используют разные методы, но ни один из них не идеален. [69] Одним из наиболее распространенных решений является объектно-реляционное отображение (ORM), которое помогает подключать объектно-ориентированные программы к реляционным базам данных. Примерами инструментов ORM являются Visual FoxPro , Java Data Objects и Ruby on Rails ActiveRecord.

Некоторые базы данных, называемые объектными базами данных , предназначены для работы с объектно-ориентированным программированием. Однако они не были столь популярны или успешны, как реляционные базы данных.

Дейт и Дарвен предложили теоретическую основу, которая использует ООП как своего рода настраиваемую систему типов для поддержки СУРБД, но она запрещает объекты, содержащие указатели на другие объекты. [70]

Ответственность и дизайн, основанный на данных

В дизайне, основанном на ответственности , классы строятся вокруг того, что им нужно делать, и информации, которой они делятся, в форме контракта. Это отличается от дизайна, основанного на данных , где классы строятся на основе данных, которые им нужно хранить. По словам Вирфс-Брок и Вилкерсона, создателей дизайна, основанного на ответственности, дизайн, основанный на ответственности, является лучшим подходом. [71]

Руководства SOLID и GRASP

SOLID — это набор из пяти правил разработки хорошего программного обеспечения, созданный Майклом Физерсом:

- Принцип единой ответственности : у класса должна быть только одна причина для изменения.
- Принцип открытости/закрытости : программные объекты должны быть открыты для расширения, но закрыты для модификации.
- Принцип подстановки Лисков : Функции, использующие указатели или ссылки на базовые классы, должны иметь возможность использовать объекты производных классов, не зная об этом.
- Принцип разделения интерфейсов : клиенты не должны зависеть от интерфейсов, которые они не используют.
- Принцип инверсии зависимости : зависимость от абстракций, а не от конкретики.

GRASP (шаблоны программного обеспечения для распределения общей ответственности) — это еще один набор правил проектирования программного обеспечения, созданный Крейгом Ларманом , который помогает разработчикам распределять обязанности между различными частями программы: [72]

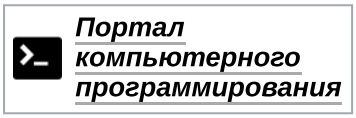
- Принцип создателя: позволяет классам создавать объекты, которые они тесно используют.
- Принцип эксперта по информации: распределяет задания по классам, содержащим необходимую информацию.
- Принцип низкой связанности: снижает зависимость классов, повышая гибкость и удобство обслуживания.
- Принцип высокой сплоченности: проектирование классов с единой, целенаправленной ответственностью.
- Принцип контроллера: назначает системные операции отдельным классам, которые управляют потоком и взаимодействиями.
- Полиморфизм: позволяет использовать различные классы через общий интерфейс, что способствует гибкости и повторному использованию.
- Принцип чистого изготовления: создание вспомогательных классов для улучшения дизайна, повышения связности и уменьшения связанности.

Формальная семантика

В объектно-ориентированном программировании объекты — это вещи, которые существуют во время работы программы. Объект может представлять что угодно, например, человека, место, банковский счет или таблицу данных. Многие исследователи пытались формально определить, как работает ООП. Записи являются основой для понимания объектов. Они могут представлять поля, а также методы, если можно хранить функциональные литералы . Однако наследование представляет

трудности, особенно при взаимодействии между открытой рекурсией и инкапсулированным состоянием. Исследователи использовали рекурсивные типы и коалгебраические типы данных для включения основных функций ООП. [73] Абади и Карделли определили несколько расширений System F $_{<:,}$ которые работают с изменяемыми объектами, допуская как полиморфизм подтипов , так и параметрический полиморфизм (дженерики), и смогли формально смоделировать многие концепции и конструкции ООП. [74] Хотя статический анализ объектно-ориентированных языков программирования, таких как Java, далек от тривиальности, он является зрелой областью, [75] с несколькими коммерческими инструментами. [76]

Смотрите также



- Сравнение языков программирования (объектно-ориентированное программирование)
- Компонентно-ориентированная разработка программного обеспечения
- Ассоциация объектов
- Язык моделирования объектов
- Объектно-ориентированный анализ и проектирование
- Объектно-ориентированная онтология

Системы

- КАДЕС
- Общая архитектура брокера объектных запросов (CORBA)
- Распределенная компонентная объектная модель
- Джеру

Языки моделирования

- IDEF4
- Язык описания интерфейса
- УМЛ

Ссылки

1. "Доктор Алан Кей о значении "объектно-ориентированного программирования" " (http://www.purl.org/stefan_ram/pub/doc_kay_oop_en) . 2003 . Получено 11 февраля 2010 .
2. Киндлер, Э.; Кривой, И. (2011). «Объектно-ориентированное моделирование систем со сложным управлением». *International Journal of General Systems* . **40** (3): 313– 343. doi : 10.1080/03081079.2010.539975 (https://doi.org/10.1080%2F03081079.2010.539975) . (https://doi.org/10.1080%2F03081079.2010.539975)
3. Льюис, Джон; Лофтус, Уильям (2008). *Java Software Solutions Foundations of Programming Design 6-е изд* . Pearson Education Inc. ISBN 978-0-321-53205-3., раздел 1.6 «Объектно-ориентированное программирование»
4. Блох 2018, стр. xi–xii, Предисловие.