

# МАТЕМАТИЧЕСКОЕ МОДЕЛИРОВАНИЕ

## ПРОГРАММНАЯ РЕАЛИЗАЦИЯ МАТЕМАТИЧЕСКИХ МОДЕЛЕЙ

ἀγεωμέτρητος μηδεὶς εἴσιτω

Характерная черта компьютерных программ, соответствующих современным математическим моделям реальных явлений или систем некоторых объектов, — *сложность*. Поскольку сложность программного обеспечения обусловлена сложностью реального мира, она неизбежна: с ней можно справиться при помощи различных приёмов, но избавиться от неё нельзя.

В следствие ограниченности человеческих мыслительных способностей — человек может разом охватить программу из десятков, но не из десятков тысяч строк — возникает необходимость разбиения сложной программной системы на всё более и более простые подсистемы (декомпозиция). Такой способ управления сложными системами известен с древности — *divide et impera* (разделяй и властвуй). Он полезен не только в программировании, но и вообще при решении любых сложных задач, см., например, [5]. Различают два основных вида такой декомпозиции:

- *объектно-ориентированная*;
- *алгоритмическая* (или *структурная*).

При структурной декомпозиции производится разделение *алгоритмов*, где каждый программный блок системы выполняет один из этапов общего процесса. При объектной декомпозиции разбиение производится по *объектам* системы. При разделении по алгоритмам внимание концентрируется на порядке происходящих событий, а при разделении по объектам особое значение придается агентам, на которые направлены или которые сами вызывают какие-то действия. Легко видеть, что эти два подхода, по меньшей мере, неразумно применять одновременно.



назад

закр.

Алгоритмическая декомпозиция является основой для структурного проектирования в программировании. В языках, где используется такой подход, основной базовой единицей является *подпрограмма*, а структуру программы в целом можно представить в форме «дерева», в котором одни подпрограммы в процессе работы вызывают другие подпрограммы. Структурный подход реализован в таких языках высокого уровня, как, например, FORTRAN или COBOL.

На идее объектно-ориентированной декомпозиции основан метод объектно-ориентированного проектирования. В его основе лежит представление о том, что программную систему необходимо проектировать как совокупность *взаимодействующих друг с другом объектов*, рассматривая каждый объект как экземпляр определенного *класса* (типа), причем классы образуют определённую структуру (иерархию). Объектно-ориентированный подход является основой последнего поколения языков высокого уровня, таких как Smalltalk, Object Pascal, C++, Java.

Какой именно подход выбрать в данном конкретном случае зависит, главным образом, от решаемой задачи, а также от имеющихся в наличии программных средств (готовых модулей, библиотек и т. п.). Преимущества объектно-ориентированной декомпозиции проявляются, главным образом, в больших проектах со сложной структурой.



назад

закр.

Структурное проектирование можно условно подразделить на *процедурное*, когда основным «строительным блоком» при проектировании является подпрограмма и *модульное*, при котором основное внимание уделяется организации данных, а не алгоритмов, по которым эти данные обрабатываются. Модулем в последнем случае называют множество взаимосвязанных процедур вместе с данными, которые эти процедуры обрабатывают. Эти два направления не исключают, а взаимно дополняют друг друга.

При структурном проектировании декомпозиция достигается с помощью метода пошаговой детализации, который иногда называют *технологией программирования сверху вниз*. Идея метода состоит в том, что программа не пишется сразу на языке высокого уровня, а придумываются некие воображаемые *исполнители*, которые «решают» поставленные задачи. Если исполнителя невозможно или затруднительно сразу реализовать программно, то придумывается другой исполнитель, конкретизирующий действия первого, для второго исполнителя могут оказаться нужными другие исполнители и т. д., пока исполнители самого нижнего уровня не удастся записать на программном языке.

Практически, при таком проектировании удобно записать программу, считая, что подзадачи, на которые она разбита, уже решены. После чего можно для каждой подзадачи писать отдельную программу (такие программы называют *подпрограммами*), в случае, если это возможно. Если же непосредственная программная реализация подзадачи невозможна или по каким-то причинам неудобна, то эта подзадача снова разбивается на другие подзадачи и т. п. Чем сложнее исходная задача, тем больше получится уровней иерархии больше будет подпрограмм. Подробнее см. [2], [3], [4].



назад

закр.

Наиболее естественно, просто и наглядно программная реализация математической модели в общем случае строится при использовании объектно-ориентированного подхода. К тому же, структурная декомпозиция становится затруднительной при программировании больших, сложных моделей. Основными преимуществами объектно-ориентированного метода по сравнению со структурным программированием являются:

- лучшая применимость к моделированию реальных физических объектов и явлений;
- *правильно выполненные* объектно-ориентированные проекты гораздо легче кодировать и сопровождать;
- возможность повторного использования программного кода.

К недостаткам можно отнести необходимость более тщательного проектирования и, как правило, большее время, затрачиваемое на разработку программ.



назад

закр.

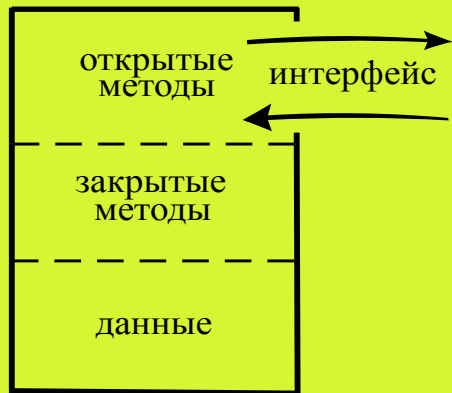
Базовыми понятиями объектно-ориентированного проектирования и программирования (ООП) являются *объекты* (они являются электронными аналогами моделируемых физических объектов) и *классы*. Объект в ООП реально существует в памяти компьютера на этапе выполнения программы, а класс — это абстракция, которая задаёт общую структуру и поведение сходных объектов. Другими словами, класс — это абстрактное множество объектов, имеющих общую структуру и поведение, а объект представляет собой экземпляр, представителя, имеющий *тип* данного класса. Объект определяется своим *состоянием*, *поведением* и *именем* (идентификатором), позволяющим отличить его от объектов того же класса или других классов. Например, класс — Homo\_Sapiense, объекты этого класса — Ivanoff, Petroff.

У объекта можно выделить две важнейшие характеристики: интерфейс и реализацию. *Интерфейс* определяет как объект «общается» с внешней средой (с пользователем, другими объектами, операционной системой). *Реализация* — это внутренние особенности объекта. Интерфейс и реализация объекта должны быть максимально независимы друг от друга в том смысле, что изменение внутренних функций не должно изменять соответствующего интерфейса.

Можно условно представлять себе объект как «чёрный ящик» с окном, через которое объект взаимодействует с внешним миром (интерфейс). Это взаимодействие осуществляется через открытые («видимые» в окне) методы, доступа к другим, закрытым, методам нет — они предназначены исключительно для внутренней реализации объекта («не видны» в окне). Также не должно быть прямого доступа к данным, хотя многие языки программирования допускают нарушение этого принципа.



Сам интерфейс организован посредством пересылки сообщений к объекту от «окружающей среды» и от объекта к «внешнему» миру. Основными базовыми принципами ООП являются:

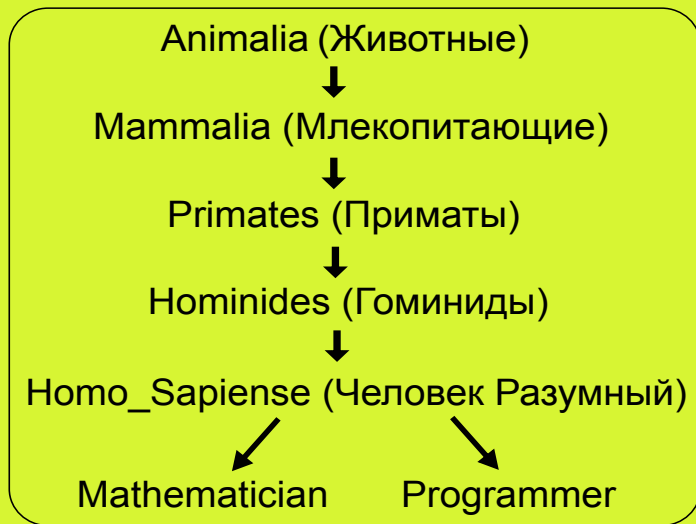


- *инкапсуляция* (encapsulation);
- *наследование* (inheritance);
- *полиморфизм* (polimorphism);
- *пересылка сообщений*.

*Инкапсуляция* — объединение в классе в единое целое данных (переменных, констант различных типов) и методов (процедур и функций), обрабатывающих эти данные. Данные класса это то, что объект этого класса «знает», а методы — то, что он «умеет». Например, для объектов класса Homo\_Sapiense данными могут быть: рост, цвет глаз, возраст, знание русского языка и арифметики, а методами — умение читать, писать и считать.

*Наследование*. Любой класс может быть порождён от другого класса (*базового* или *родительского*). Порождённый класс (*потомок*, *производный класс*) автоматически получает доступ к данным и методам родителя, т. е. наследует его структуру, и, кроме того, имеет возможности переопределять («улучшать») методы и дополняться новыми данными и методами.

Например, классу `Programmer`, наследнику класса `Homo_Sapiense`, можно добавить свойство знания языка программирования C++ и метод — умение работать в IDE (интегрированных средах разработки), а способность считать усовершенствовать до владения калькулятором. Точно так же, порождённый



от `Homo_Sapiense` класс `Mathematician` можно наделить «знанием», скажем, интегрального исчисления или топологии и добавить метод — умение интегрировать. Все свойства и методы базового класса `Homo_Sapiense` будут также присущи любому объекту классов `Programmer` и `Mathematician`. Класс-потомок может, в свою очередь, быть базовым для другого класса. Таким образом, совокупность классов, связанных отношениями наследования, может образовывать

иерархию или «дерево» классов любой сложности. У каждого класса может быть сколько угодно потомков. Что касается предков, то в некоторых языках программирования (например, C++) их может быть несколько, а в других (Object Pascal, C#) множественное наследование не допускается.



*Полиморфизм* даёт возможность решать сходные задачи различными способами. Для изменения метода родительского класса в классе-потомке его можно переопределить. Наследование позволяет различным типам данных совместно использовать один и тот же код, что приводит к уменьшению его размера и повышению функциональности, а полиморфизм перекраивает этот общий код так, чтобы удовлетворить конкретным особенностям отдельных типов данных. Практически полиморфизм часто реализуется через механизм т. н. *виртуальных функций*.

Виртуальные функции позволяют объекту производного класса изменять поведение, определённое на уровне базового класса или обеспечить какие-либо возможности базового класса, которые он не может осуществить из-за того, что нужная для этого информация объявляется на уровне производного класса. Виртуальные методы позволяют объявить класс общего назначения, не требуя для этого знания конкретных особенностей, которые могут быть доступны только в производных классах. Например, поскольку для работы с различными IDE (Integrated Development Environment — интегрированная среда разработки) требуются различные навыки, то соответствующий метод в `Programmer` можно объявить виртуальным, а в классах-наследниках, которые назовём, к примеру, `Borland_Programmer` и `Microsoft_Programmer`, должным образом его переопределить, чтобы наш программист обрёл способность работать в `Borland C++ Builder` и `Visual C++` соответственно.



назад

закр.


Другим важным понятием и принципом ООП является *пересылка сообщений*, которая содержательно выражает основную методологию построения объектно-ориентированных программ. Любой объект может отправлять сообщения другим объектам и принимать сообщения от них. При получении сообщения объекты выполняют определённые действия, причём разные объекты приняв одно и то же сообщение, вообще говоря, могут выполнять разные действия. Другими словами, поведение и реакция, инициируемые сообщением, зависят от объекта-получателя. Практически пересылка сообщений осуществляется вызовом соответствующего метода, обрабатывающего это сообщение. Сообщения это то, при помощи чего объекты «общаются» друг с другом. Например, для объекта класса `Programmer` можно предусмотреть отправку сообщения объекту класса `Mathematician` с запросом на вычисление некоторого интеграла.


Таким образом,

**ООП** — метод построения программ в виде множества взаимодействующих посредством передачи сообщений объектов, структура и поведение которых заданы соответствующими классами, и все эти классы принадлежат иерархии классов, выражающей отношение наследования.

Не трудно заметить аналогию между методами ООП и инженерным проектированием. Действительно, техническая система в самом общем виде является совокупностью некоторых физических объектов и устройств (их электронные аналоги — объекты в ООП) различного типа (класса). Эти устройства обладают (инкапсулируют) некоторыми определёнными характеристиками (данные) и функциональностью (методы) и взаимодействуют между собой посредством каких-либо механических, электрических или иных связей (пересылка сообщений). Кроме этого, такую систему можно модифицировать, изменив свойства некоторых её объектов или добавив к ней новые (наследование и полиморфизм).

**ООП-проектирование.** Сейчас не существует единого и универсального способа объектно-ориентированного проектирования. К настоящему времени не разработаны строгие методы классификации и нет правил, позволяющих выделять классы и объекты. Можно, однако, *рекомендовать* следующий порядок действий:

 **1.** Разработку программы следует начать с анализа функционирования моделируемой системы, так как её поведение обычно известно задолго до всех остальных свойств. Затем следует спроектировать объекты и сообщения, которыми они будут обмениваться. *После* определения того, как объекты будут «общаться» друг с другом, можно приступить к проектированию классов.

 **2.** Проектирование иерархии классов. Обработчики всех *общих* сообщений объединяйте в один базовый класс. Например, если имеется объект, получающий сообщения А, В, С и другой, получающий сообщения В, С, D, Е, то нужно создать базовый класс, обрабатывающий сообщения В, С и два производных от него класса, один из которых будет содержать метод, обрабатывающий А, а другой — методы, работающие с D и Е. Если продолжать этот процесс, пока не будут исчерпаны все объекты, то в результате получится требуемая иерархия классов, в «вершине» которой будет располагаться самый общий класс, от которого наследуют все другие классы. Любой базовый класс должен иметь не менее двух потомков. Возможности базового класса должны использоваться всеми его производными классами. Все методы, обрабатывающие сообщения объявите открытыми (public), а все другие, по возможности, должны быть закрытыми (private, protected).

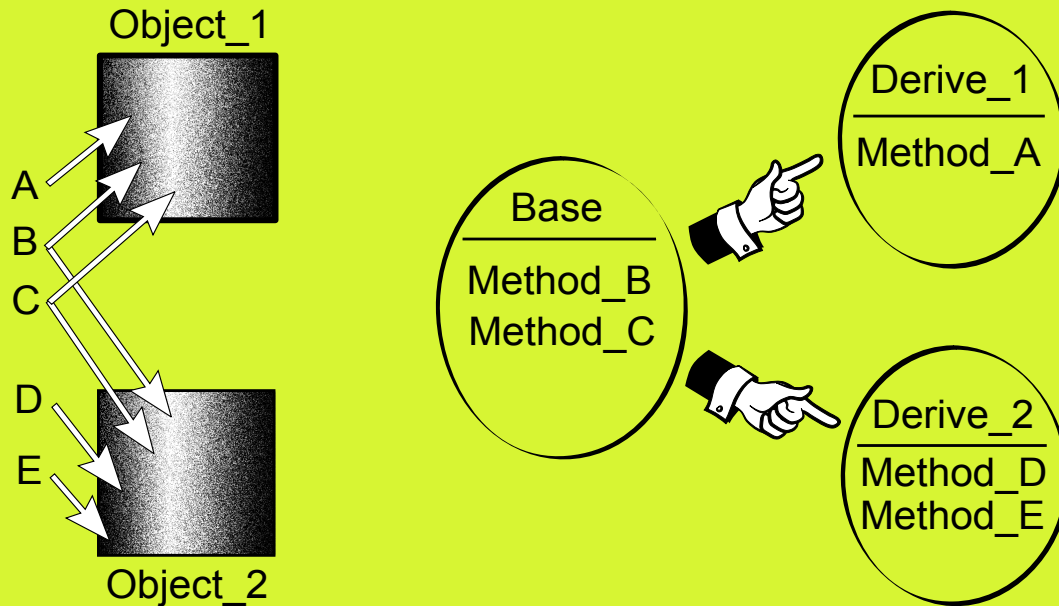


Рис. 1. Процесс проектирования классов

👉 3. Данные добавляются в последнюю очередь. Для поддержки работы с данными на этом этапе также добавляются методы, которые должны быть *закрытыми*, как и сами данные. Обращение к данным должно быть реализовано *только с помощью методов класса*, т. е. интерфейс объекта с его окружением должен полностью определяться его методами, чтобы к его состоянию не было другого доступа извне, кроме как через методы [7], [1].

# Список литературы

1. В. И. Зенкин. Практический курс математического и компьютерного моделирования. Учеб. пособие. Калининград: изд. РГУ им. Канта, 2006.
2. У. Дал, Э. Дейкстра, Э. Хоор. Структурное программирование. М, 1975.
3. Н. Вирт. Алгоритмы и структуры данных. М, 1989.
4. А. Г. Кушниренко, Г. В. Лебедев. Программирование для математиков. М, 1988.
5. Д. Пойа. Математическое открытие. М, 1970.
6. Т. Бадд. Объектно-ориентированное программирование в действии. СПб, 1998.
7. Г. Буч. Объектно-ориентированный анализ и проектирование с примерами приложений на C++. М.-СПб., 1998.
8. Д. Кнут. Искусство программирования для ЭВМ. М., Т. 1, Т. 2: Получисленные алгоритмы, 1977; Т. 3: Сортировка и поиск, 1978.



назад

закр.