УДК 519.682.6 DOI: 10.17587/prin.13.119-123

М. В. Сергиевский, канд. техн. наук, доц., sermax@yandex.ru, Национальный исследовательский ядерный университет "МИФИ", Москва

Метаклассы в UML и в языках программирования

Важными этапами процесса разработки объектно-ориентированных информационных систем являются проектирование и программирование. На этапе проектирования строится модель предметной области, в которой могут быть использованы метаклассы. Но, как известно, в языке UML нет прямой поддержки метаклассов. В статье описан способ, как в ряде случаев можно перейти от моделей с метаклассами к моделям с обычными классами.

Приведены примеры, показывающие, какими возможностями обладают языки программирования Python, Scala и Objective-C для реализации таких моделей. Кроме того, анализируется, как по-разному в этих языках трактуется понятие метакласса.

Ключевые слова: *класс, метакласс, объект, UML, наследование, отношение классификации, предметная область*

Введение

Введем несколько определений, связанных с понятиями объекта, класса и метакласса [1].

Класс — сущность, определяющая структуру и функциональность входящих в его состав однотипных элементов, которые называются объектами.

Объект — экземпляр класса.

Метакласс — это класс, экземплярами которого являются классы.

Между классами может существовать отношение наследования, показывающее, что один класс может наследовать структуру и поведение другого. Отношение наследования может существовать и между метаклассами.

Отношение классификации INSTANCE OF устанавливается между экземпляром класса и самим классом, т. е. оно существует, с одной стороны, между объектами и классами, с другой стороны, между классами и метаклассами.

В настоящее время существует несколько концепций метакласса [2]. По крайней мере две из них получили практическое воплощение.

Первая концепция предусматривает наличие только одного метакласса. Каждый класс всегда имеет строгий шаблон, задаваемый выбранной объектной моделью или языком программирования. Он определяет, например, допустимо ли множественное наследование, какие существуют ограничения на именование классов, как описываются поля и методы, набор существующих типов данных и многое другое. Таким образом, класс можно рассматривать как объект, у которого есть свойства: имя, список полей и их типы, список методов, список аргументов для каждого метода и т. д. Также класс может обладать поведением, т. е. поддерживать реализацию методов. А поскольку для любого объекта существует шаблон, описывающий свойства и поведение этого объекта, значит, его можно определить и для класса. Такой шаблон, задающий различные классы, называется метаклассом. В реальных предметных областях использовать такую концепцию метакласса для построения модели предметной области смысла не имеет, поскольку метакласс в данном случае выступает в роли абстрактной категории, являющейся как бы надстройкой над всеми классами. Ее существование при реализации можно предусмотреть по умолчанию.

При использовании второй концепции допускается существование множества метаклассов. Подобно тому, как объекты со сходными свойствами группируются и являются экземплярами определенного класса, так и классы могут группироваться и приписываться к вполне конкретному метаклассу. Эта концепция близка к идее общего предка для группы классов. И чаще всего она реализуется без введения понятия метакласса с помощью отношения наследования между классами. Именно такая концепция метакласса будет преимущественно рассматриваться в данной работе. Таким образом, классы как объекты будем относить к определенному метаклассу. И таких метаклассов в общем случае может быть несколько.

UML и понятие метакласса

Метакласс используется при моделировании с помощью UML только на уровне метамодели [3]. То есть для точного определения, какие сущности (классы, отношения и т. п.) могут использоваться в модели. В UML все элементы модели являются экземплярами какого-либо метакласса. Например, чтобы в модели могли быть использованы классы, на уровне метамодели должен существовать метакласс Class.

Таким образом, непосредственно использовать метаклассы при построении модели предметной области с помощью UML нельзя. Тем не менее метаклассы как элементы абстракции могут быть необходимы. Существует несколько возможностей представлять метаклассы в диаграммах классов

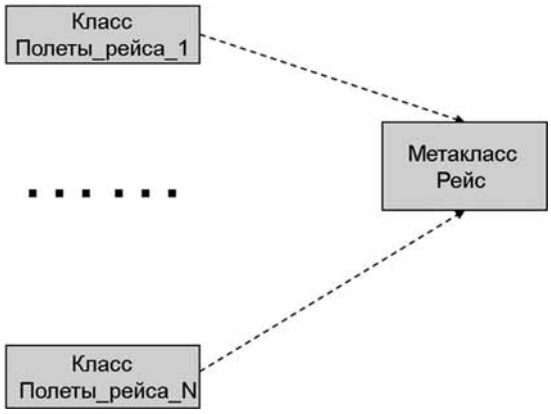

Рис. 1. Модель предметной области с метаклассом

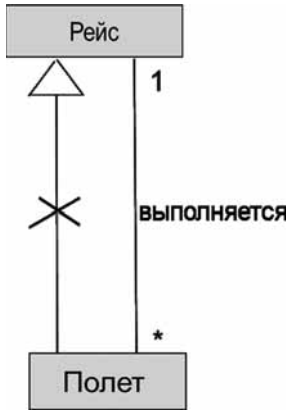

Рис. 2. Модель предметной области "Авиарейсы"

UML. Лучше всего рассмотреть эти возможности на реальных примерах.

Предположим, что предметная область включает данные о авиарейсах и полетах, выполняемых в рамках этих рейсов. Все множество авиаполетов можно разбить на подмножества, относящиеся к одному рейсу. Тогда конкретный рейс можно рассматривать как класс, а все множество рейсов — как метакласс, состоящий из объектов-классов — конкретных рейсов. Модель предметной области в данной трактовке представлена на рис. 1.

Как средствами UML описать такую ситуацию? Поскольку в UML нет возможности использовать метаклассы для описания предметной области, нужно довольствоваться обычными классами и объектами [4—6].

Проведем упрощение модели. Будем рассматривать множество рейсов не как метакласс, а как класс, объектами которого являются отдельные рейсы, идентификаторами которых служат номера рейсов. Для простоты будем считать, что совмещенных рейсов нет. Отдельные рейсы в данном случае являются обычными объектами, а не классами. В рамках стандартного UML можно предложить три модели, описывающие такую ситуацию.

Модель 1. В этой модели для представления рейсов используется абстрактный класс Рейс. Тогда между ним и другими классами предметной области (в первую очередь, полетами) могут существовать только отношения наследования (в терминологии UML — обобщения) и зависимости. Отношений ассоциации в данном случае быть не может, поскольку объектов абстрактного класса не создается; достаточно использовать только отношение обобщения.

Модель 2. В этой модели между классами предметной области в принципе могут быть предусмотрены два отношения: обобщение и ассоциация. Существование отношения ассоциации сразу же говорит то, что будут создаваться объекты родительского класса, т. е. класса Рейс. И объекты класса-потомка Полет будут обязательно связаны с объектами родителя (все зависит от кратности). Тогда нет смысла вообще использовать отношение обобщения, чтобы не дублировать атрибуты у объектов потомка и его потенциального родителя. Недостающие атрибуты объект, который потенциально является потомком, заимствует у прежнего родителя, с которым он будет связан только отношением ассоциации (рис. 2).

Модель 3. В этой модели полеты одного рейса объединяются в классы, и таких классов столько же, сколько различных рейсов. Целесообразно поступать именно так, если рейсы имеют определенную специфику, выражающуюся в наличии уникальных атрибутов и операций. Но поскольку в UML метаклассы не поддерживаются, вместо метакласса в этой модели используется абстрактный класс-предок. Преимущество такого подхода в том, что он дает возможность унифицировать часть операций для всех классов-полетов (рис. 3).

Расширим представление о предметной области, чтобы она точнее описывала реальную ситуацию. Включим в нее данные о рейсах, полетах, выполняемых в рамках определенных рейсов, и пассажирах. Класс Рейс включает следующие атрибуты: номер; пункт отправления, пункт назначения, день недели; время вылета. Класс Полет содержит такие атрибуты: дата; время в пути; доп. параметры, характеризующие полет. Класс Пассажир содержит данные о пассажирах и билетах. Понятно, что атрибуты рейса неявно являются и атрибутами полета. Но, если существует отношение ассоциации между рейсом и полетом, то повторять эти атрибуты в классе Полет нецелесообразно (рис. 4).

Приведем еще один пример [7]. Предположим, что предметная область включает данные о литературных произведениях и книгах (не сборниках), как материальных объектах. Литературное произведение

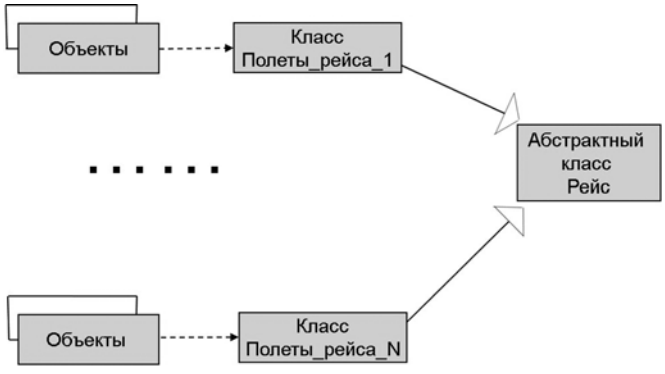

Рис. 3. Модель предметной области "Авиарейсы " с множеством классов

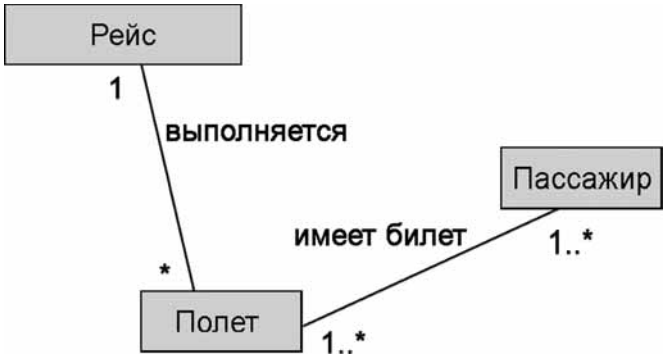

Рис. 4. Расширенная модель предметной области "Авиарейсы"

как класс характеризуется следующими атрибутами: название, годы написания, оглавление, объем в знаках. Книга, как класс материальных объектов, может иметь такие атрибуты: число страниц, переплет, состояние, идентификационный номер. Очевидно, что атрибуты произведения являются и атрибутами книги. Можно рассматривать произведение, как метакласс, состоящий из классов — конкретных произведений, а книги — как экземпляры конкретных произведений. Но разумнее поступить так же в предыдущем случае. То есть, либо отказаться от концепции метакласса, а рассматривать только два обычных класса и связать их не отношением обобщения, а отношением ассоциации, чтобы не дублировать значения атрибутов, относящихся к литературному произведению, в объектах-книгах. Либо объединить объекты-книги в классы, и таких классов будет столько же, сколько различных произведений, а вместо метакласса использовать общий абстрактный класс-предок.

Метаклассы в Python

Как известно, структура объекта полностью определена в его классе. Следовательно, структура любого класса должна быть определена в его метаклассе. Класс должен иметь имя, набор полей и набор методов, т. е. метакласс должен предписывать любому классу иметь именно эти атрибуты. В таком случае метакласс выступает в роли шаблона. Именно такая концепция реализована в языке Python [8].

Классы в Python — это объекты, а значит, как и любой другой объект, класс можно создавать динамически. Именно это и делается в Python во время выполнения оператора `class`. Впоследствии с классом, как с объектом, можно проводить различные операции, например, добавлять поля и методы. А поскольку класс — это объект, то должен существовать специальный класс (метакласс), экземпляром которого он является. Отметим, что с метаклассом никаких операций не совершается, и, что очень важно: метакласс в Python всего один. Таким образом, метакласс в Python — это некая абстракция, но вполне реальная. Иерархия отношений классификации между объектами, классами и метаклассами приведена на рис. 5.

Но есть и другой способ задания классов. Класс в Python можно не объявлять стандартным образом, а напрямую создавать как специальный объект, используя метод `type`. Модель "Авиарейсы", приведенная на рис. 3, на языке Python может быть реализована путем генерации классов следующим образом:

```
type (Var, (Flight), {Date, Time}),
```

где `Var` — переменная, последовательно принимающая значения `"FlightA"... "FlightU"`, которые являются именами классов, определяющих относящиеся к одному рейсу объекты-полеты; `Flight` — абстрактный класс рейсы; `Date` (дата полета), `Time` (время в пути) — атрибуты классов полеты. Поскольку заранее число классов неизвестно, то их надо создавать динамически. Число созданных классов, таким, образом в разных случаях будет различным.

Параметрами метода `type` являются: имя создаваемого класса, имя класса-предка, списки полей и методов создаваемого класса (в данном случае методов нет). Тогда в принципе можно трактовать `type` как метакласс, который используется для генерации классов. А уже потом при обращении к сгенерированному классу можно, в свою очередь, создавать обычный объект.

Моделирование метаклассов на языке Scala

В большинстве случаев метаклассы нужны, чтобы сначала создавать различные классы, а потом их объекты. И желательно делать это в как можно более общем виде. Но существует хорошо известный шаблон проектирования — Factory [9], который как раз и позволяет делать подобные вещи, т. е. в зависимости от условий создавать объекты разных классов. Чаще всего имя класса, объект которого должен быть создан, или информация о нем передаются из внешнего источника. Можно сказать, что в какой-то степени этот шаблон моделирует работу с метаклассами.

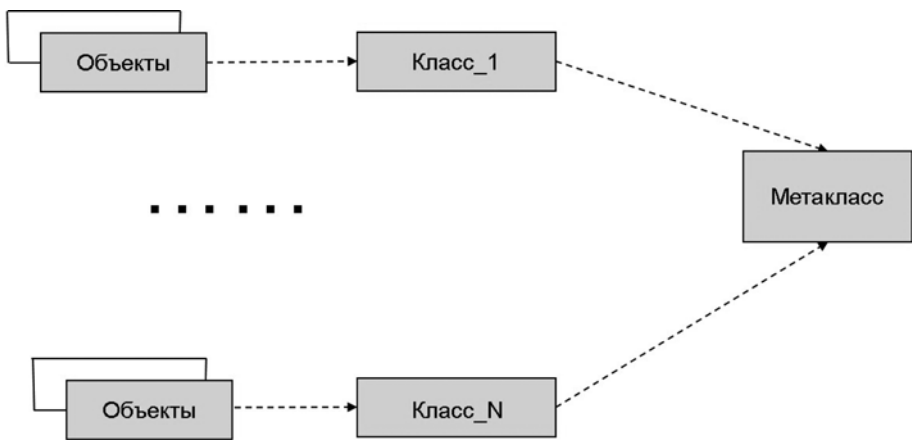

Рис. 5. Иерархия отношений классификации в Python:
---------► — отношение классификации INSTANCE OF

Продемонстрируем, как это может быть сделано на языке Scala [10] для задачи о рейсах и полетах.

```
abstract class Flight{
    val par: String
}
object Flight {
   case class FlightA(par: String) extends
   Flight
. . . . . .
      case class FlightU (par: String)
      extends Flight def apply(par:
      String): Flight={
      if (par=="FlightA")
            FlightA(par)
. . . . . .
           else
            FlightU(par)
  }
}
```

В данном случае, в зависимости от значения параметра `par` создаются объекты разных классов: от `FlightA` до `FlightU`. Делается это следующим образом:

```
var ObjFl = Flight(par)
```

Метод `apply` возвращает значение типа `Flight`, потомками которого являются классы `FlightA,...,FlightU`. Правда, недостатком такого подходя является необходимость предварительно определить все классы. Case-классы, а не обычные классы, использованы здесь, чтобы упростить запись кода, избежав явного создания объектов.

Данные о рейсах, относящиеся к различным множествам объектов-полетов — экземплярам классов `FlightA...FlightU` — удобно хранить в соответствующих классам объектах-спутниках. Например, для класса `FlightA` объект-спутник будет определяться так:

```
Object FlightA {
val NFlight=303
val DepTime=15.30
}
```

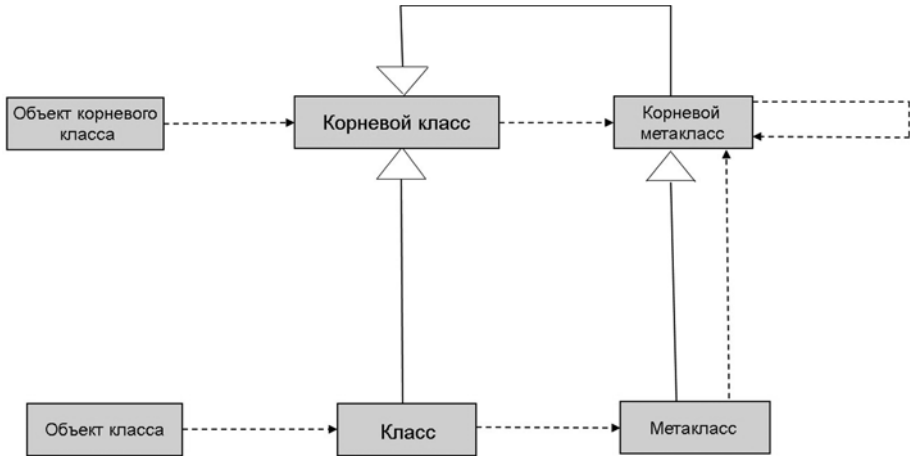

Рис. 6. Иерархия отношений объект — класс — метакласс в Objective-C

Метаклассы в языке Objective-C

Иная концепция метакласса используется в языке Objective-C [11]. Основной постулат: каждый элемент в Objective-C является объектом. Также считается, что у каждого класса есть свой уникальный метакласс. В теле каждого объекта Objective-C (т. е. в области памяти, которая для него распределяется) обязательно есть ссылка на класс. Поскольку класс — это тоже объект, у него должна быть ссылка на его класс, т. е. метакласс. Определяя класс, по умолчанию создаем парный для него метакласс. Говоря упрощенно, метакласс характеризуется теми статическими методами и полями, которые есть в классе. Таким образом, в Objective-C принята концепция раздельного хранения обычных методов, которые могут вызываться только тогда, когда объект создан, и методов класса, которые могут быть вызваны до создания объектов.

Наличие метаклассов обязательно, поскольку именно метаклассы хранят статические поля и методы классов. Следовательно, для каждого класса должен быть создан уникальный метакласс, так как каждый класс потенциально может иметь уникальный набор статических полей и методов.

Для создания пары класс — метакласс используется метод `objc _ allocateClassPair`. Для рассматриваемого примера, связанного с предметной областью "Авиарейсы", данные, относящиеся ко всем объектам класса (такие как номер рейса `NFlight`, время отправления `DepTime`), размещаются в метаклассе. Напомним, что в Objective-C метаклассы обычно создаются средой выполнения по умолчанию и не имеют имен. Иерархии отношений классификации и наследования между объектами, классами и метаклассами в языке Objective-C приведены на рис. 6.

Обратим внимание на то, что только корневой класс не имеет класса-предка, а корневой метакласс является объектом самого себя.

Заключение

Концепция метакласса появилась достаточно давно, но реального применения до последнего времени практически не находила. Причин здесь несколько. Основная состоит в том, что в реальных предметных областях аналогов метаклассов, по существу, нет. Поэтому нет метаклассов и в универсальном средстве моделирования программных систем — UML. Но как элементы абстракции метаклассы могут возникать. В принципе можно допустить существование множества метаклассов. То есть обычные классы могут группироваться и приписываться к вполне конкретному метаклассу. Показано, как в ряде случаев можно перейти от моделей с метаклассами к стандартым UML-моделям.

В отличие от UML в некоторых языках программирования явно

или неявно понятие метакласса все-таки используется. Тут, в первую очередь, надо говорить о языках Python и Objective-C. В Objective-C понятие метакласса далеко от канонического: допускается существование множества метаклассов, поскольку автоматически в пару к каждому обычному классу добавляется метакласс, хранящий статические поля и методы. В Python принята концепция единого метакласса для всех классов. В этом случае метакласса в модели предметной области создавать не нужно, но он по умолчанию будет присутствовать в реализации.

Список литературы

1. **Olive A.** Conceptual Modeling of Information Systems. Springer Science & Business Media, 2007. 455 p.
2. **Forman I. R., Danforth S. H.** Putting Metaclasses to Work, Addison-Wesley, 1998. 447 p.
3. **Rumbaugh J., Jacobson I., Booch G.** Unified Modeling Language. 2nd edition, Boston, Addison-Wesley, 2004, 742 p.
4. **Sergievskiy M.** N-ary Relations of Association in Class Diagrams: Design Patterns. // International Journal of Advanced Computer Science and Applications. 2016. Vol. 7, No. 2. P. 265—268.
5. **Sergievskiy M.** Modeling Unified Language Templates for Designing Information Systems. // Automatic Documentation and Mathematical Linguistics. 2020. Vol. 54, No. 1. P. 26—35.
6. **Sergievskiy M., Kirpichnikova K.** Optimizing UML Class Diagrams. // ITM Web of Conferences. 2018. Vol. 18. Article Number. 03003.
7. **Сергиевский М. В.** Шаблоны унифицированного языка моделирования для проектирования программных систем. // Научно-техническая информация. Серия 2: Информационные процессы и системы. 2020. № 1. С. 19—27.
8. **Рамальо Л.** Python. К вершинам мастерства. М.: ДМК-Пресс, 2015. 768 с.
9. **Gamma E., Johnson R., Helm R., Vlissides J.** Design Patterns. Elements of Reusable Object-Oriented Software. Addison-Wesley, 2001. 416 p.
10. **Одерски М., Спун Л., Веннерс Б.** Scala. Профессиональное программирование, 3-е изд. СПб.: Питер, 2017. 1100 с.
11. **Gallagher M.** What is a meta-class in Objective-C? 2010. URL: https://www.cocoawithlove.com/2010/01/what-is-meta-class-in-objective-c.html

Metaclasses in UML and in Programming Languages

M. V. Sergievskiy, sermax@yandex.ru, National Research Nuclear University MEPhI (Moscow Engineering Physics Institute), Moscow, 115409, Russian Federation

Corresponding author:

Sergievskiy Maxim V., Associated Professor, National Research Nuclear University MEPhI (Moscow Engineering Physics Institute), Moscow, 115409, Russian Federation

Received on January 16, 2022
Accepted on January 26, 2022

Design and programming are important stages of the development process of object-oriented information systems. At the design stage, a domain model is built, in which metaclasses can be used. But, as widely known, there is no direct support for metaclasses in UML. The article describes how in some cases it is possible to switch from models with metaclasses to models with regular classes. Several variants of such a transition are given. Two concepts of the metaclass are considered, which have now found application.

Examples are given showing what capabilities the Python, Scala and Objective-C programming languages have for implementing such models. In addition, it analyzes how the concept of a metaclass is interpreted differently in programming languages.

Keywords: *class, metaclass, object, UML, inheritance, classification relation, domain area*

For citation:

Sergievskiy M. V. Metaclasses in UML and in Programming Languages, *Programmnaya Ingeneria*, 2022, vol. 13, no. 3, pp. 119—123.

DOI: 10.17587/prin.13.119-123

References

1. **Olive A.** *Conceptual Modeling of Information Systems,* Springer Science & Business Media, 2007, 455 p.
2. **Forman I. R., Danforth S. H.** *Putting Metaclasses to Work*, Boston, Addison-Wesley, 1998, 447 p.
3. **Rumbaugh J., Jacobson I., Booch G.** *Unified Modeling Language*, 2nd edition, Boston, Addison-Wesley, 2004, 742 p.
4. **Sergievskiy M.** N-ary Relations of Association in Class Diagrams: Design Patterns, *International Journal of Advanced Computer Science and Applications*, 2016, vol. 7, no. 2, pp. 265—268.
5. **Sergievskiy M.** Modeling Unified Language Templates for Designing Information Systems, *Automatic Documentation and Mathematical Linguistics*, 2020, vol. 54, no. 1, pp. 26—35.
6. **Sergievskiy M., Kirpichnikova K.** Optimizing UML Class Diagrams, *ITM Web of Conferences*, 2018, vol. 18, article number 03003.
7. **Sergievsky M. V.** UML Templates for Information Systems Design, *Nauchno-tekhnicheskaya informaciya. Seriya 2: Informacionnye processy i sistemy*, 2020, no. 1, pp. 19—27 (in Russian).
8. **Ramalho L.** *Python. To the heights of mastery,* Moscow, DMK-Press, 2015, 768 p. (in Russian).
9. **Gamma E., Johnson R., Helm R., Vlissides J.** *Design Patterns. Elements of Reusable Object-Oriented Software,* Addison-Wesley, 2001, 416 p.
10. **Odersky M., Spoon L., Winners B.** *Scala. Professional programming,* 3rd edition, Saint Petersburg, Piter, 2017, 1100 p. (in Russian).
11. **Gallagher M.** What is a meta-class in Objective-C? 2010, available at: https://www.cocoawithlove.com/2010/01/what-is-meta-class-in-objective-c.html