ПОЛИМОРФИЗМ В КУРСЕ ОБЪЕКТНО ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ - ДИДАКТИЧЕСКИЕ АСПЕКТЫ

Ивайло Дончев[1], *Эмилия Тодорова*[2]

Великотырновский университет имени Св.Св. Кирилла и Мефодия
Болгария, 5000 Велико Тырново, ул. "арх. Г. Козарев" 3, тел. +359 62 649831,
E-mail: [1]i.donchev@abv.bg, [2]emilia_todorova@yahoo.co.uk

Abstract

When learning Object-oriented programming, it is extremely important to master polymorphism as one of the three characteristics distinguishing Object-oriented programming from the other program paradigms. Using it skillfully makes it possible to achieve improved code organisation and better readability, on the one hand, and on the other – easy expandability of programs without the need to make changes in already established classes, even a long time after commissioning the program system for operation.

This paper makes a critical analysis of educational literature related to polymorphism – we propose several approaches for classifying the types of polymorphism in a way suitable for the course of training; we examine the relations of the concept of polymorphism with other basic concepts, and we provide specific recommendations for improvement of the teaching process.

Вступление

Полиморфизм в Объектно ориентированном программировании (ООП) – это возможность объектов разных типов реагировать на одно и то же сообщение таким образом, который является специфичным для каждого типа. Программист не обязан предварительно знать точный тип объекта - получателя сообщения. Он определяется во время исполнения программы (динамично). Поэтому этот процесс называется динамичным (или поздним) связыванием. Требование к разным объектам - иметь только совместимый интерфейс – наличность public метода с одним и тем же именем, параметрами и типом результата во всех объектах. В принципе, классы, к которым принадлежат объекты, не должны быть связанны каким-либо образом, но так как они разделяют общий интерфейс, в большинстве языках обязательное ограничение представлять их как производные классы одного общего базового класса. Таким образом, этот вид полиморфизма (называемый еще динамичным полиморфизмом, подтиповым (subtype) полиморфизмом [11], inclusion полиморфизмом [12]) является одним из основных применений наслеживания. Выражается в том, что объекты всех производных классов можно манипулировать однотипно при помощи интерфейса своего базового класса. Чтобы подчеркнуть важность полиморфизма для обучения ООП, приводим цитату по Lippman: "Подтиповой полиморфизм, способом наслеживания и динамического связывания, обеспечивает фундамент для ООП" [11]. Вот почему в большинстве учебников по ООП под термином полиморфизм понимаем, кстати, именно этот вид полиморфизма, и его изучение и правильное применение является особенно важным в процессе обучения программированию вообще.

1. Классификация видов полиморфизма

Так как динамичный полиморфизм не единственный - в языках программирования известны формы полиморфизма и успешно применяются далеко до объектно ориентированного подхода и его популярности, то классификация видов полиморфизма поможет студентам лучше понять и концепции, и выбрать самый удачный в каждом конкретном случае механизм и языковую конструкцию.

В принципе, языки для ООП поддерживают несколько форм полиморфизма, но в учебной литературе для них используются другие термины и не акцентируется на их естественность как виды полиморфизма.

В зависимости от числа типов, в которых можно его применять, полиморфизм можно разделить на две фундаментальные категории:

- **специальный** (ad-hoc) полиморфизм: конечный счет типов, причем комбинации между ними предварительно известны.
- **параметричный** (универсальный) полиморфизм: можно применять и с типами, которые еще неизвестны в программе.

В таком случае динамичный полиморфизм относится к категории специального полиморфизма. К той же самой категории можно соотнести передефинированные операторы и передефинированные функции, которые кроме к классам можно применять и к стандартным типам.

Параметричный полиморфизм тоже находить применение в языках для ООП (например, шаблоны в C++). Там он известен под термином генеричность типов.

В зависимости от вида связывания можем сделать другую классификацию полиморфизма **- статичный и динамичный**. Таким образом к категории статичный полиморфизм можем перечислить передефинированные функции, передефинированные операторы и шаблоны [8].

Так как в учебниках программирования главным образом речь идет о самых популярных языках, как C++, Java, C# в их большинстве скрывается факт, что эти языки по самом деле являются языки с единичным управлением (single dispatch languages), т.е. число полиморфичных аргументов, по которым определяется какой метод надо применить, оказывается единственным. Такой подход обеспечивает эффективное управление динамическим связыванием, способом специальных таблиц, но существуют и некоторые ограничения как строгие правила для перепокрытых функций и ограниченной объектной модели. Поэтому языки как CLOS и Cecil поддерживают управление более одного полиморфичными аргументами. В [9] такие языки названы языками с множественным управлением (multiple dispatch languages).

Следовательно, в зависимости от числа аргументов, по которым динамично выбираем метод, можем классифицировать полиморфизм как **единичный и множественный**.

Методы, которые содержат больше одного полиморфичных аргументов называют мультиметодами. Типичные ситуации, в которых необходимы такие методы описаны в [9][5]. В языках, которые поддерживают только единичный полиморфизм, можно симулировать множественное полиморфичное поведение [9]. Существуют и расширения популярных языков, которые предоставляют възможность для множественного полиморфизма. Для C++ укажем на открытые мультиметоды [14], для Java – Featherweight Multi Java [4] и MultiJava [5].

Изучение множественного полиморфизма превышает цели вводного курса по ООП и является подходящим для курса по программированнию для опередевших. Конечно, в курсе по ООП была бы польза продемонстрировать как средствами единичного полиморфизма можно симулировать множественный.

2. Связь полиморфизма с другими основными понятиями

Понятия динамическое связывание, виртуальные методы (функции), абстрактные классы и полиморфизм в ООП неразрывно связаны и невозможно рассматривать их раздельно. В некоторых книгах, например [6, с. 311] полиморфизм и динамическое связывание рассматриваются как синонимы. Для успешного применения объектного подхода студенты должны уметь разграничивать эти понятия и понимать связь между ними.

В языках для ООП динамическое связывание применяется при специальном виде члена-функции классов, называемых виртуальными функциями или виртуальными методами. В C++ в целях лучшей эффективности подпонимания применяется статическое связывание. Программист может выбрать данную член-функцию использовать динамическое связывание объявив ее как виртуальную спецификатором virtual. В Java маханизм динамического связывания на этапе подпонимания, и не надо использовать специальное ключевое слово. Такое слово можно использовать для активизации статической связывании – метод объявяется как final. Поэтому в Java не пользуемся термином виртуальный метод – согласно подпониманию все методы являются виртуальными.

Преимущество виртуальных функций втом, что детали реализации остаются скрытыми для программиста. Кроме того, при добавлении нового класса в наследственную йерархию, он автоматически включается в механизм динамического связывания. Это минимизирует объем программной системы и опрощает ее поддержку.

Среди студентов, изучающих C++ замечается некоторый переполох при использовании одного и того же ключевого слова – virtual для объявления класса как виртуальный базовый класс и декларации виртуальной функции. Преподаватели должны обострять внимание для разграничения этих принципно разных ситуаций.

Danny Kalev [8] обращает внимание на проблему разграничения понятий абстрактный тип данных (АТД) и абстрактный класс – это существенный для понимания объектно ориентиранного подхода момент. АТД самостоятельный, дефинированный программистом тип, который объединяет данные и связанные с ними операции. Он не может расширяться или проявлять динамичный полиморфизм. Абстрактный класс – это не тип данных – он обычно не содержит данных и, кроме того, от него невозможно инстанцировать объекты. ”Абстрактный класс это скорее ”скелет”, который определяет множество услуг или операций, которые остальные (не абстрактные) классы имплементируют” [8]. Таким образом логично классы, которые можно инстанцировать, называются конкретными.

В исследуемой литературе понятие абстрактный класс можно дефинировать двумя способами:
- понятием чистая виртуальная функция (или абстрактный метод в Java);
- с точки зрения его роли интерфейса для производных классов: как суперкласс для других классов.

Абстрактные классы – это классы, которые формируют связанную, но незаконченную концепцию, которая предоставляет свои характеристики для специализации способом наслеживания. Так как они служат только для предоставления наслеживаемой функциональности, в литературе их называют еще абстрактными суперклассами, абстрактными супершаблонами (в языке Beta) или абстрактными базовыми классами, а в терминологии CLOS и основными классоми (basic classes). Жаргонно иногда программисты называют их “частичными типами”.

Когда абстрактный класс онаслеживается, все его чистые виртуальные методы надо имплементи-

ровать в производном классе. Если это не случится, производный класс тоже становится абстрактным.

Одна из основных целей обучения по ООП должна быть приобретение умений работы с абстрактными классами. Как доказательство, приводим цитату слов Stroustrup: „Во многих случаях абстрактные классы - это идеальный способ представления важных внутренних интерфейсов системы. Они простые, ефикасные, строго типизированные, позволяют одновременное использование многих разных реализаций концепции, представленной интерфейсом и в полне изолируют потребителя от возможных перемен в этих реализациях." [17].

3. Преподавание полиморфизма

Во многих учебниках полиморфизм рассматривается только как „свойство членов-функций объектов" [21] или как „возможность для динамичного многовариантного выбора члена-функции класса в йерархической наследственной структуре классов" [19, с. 10], тоесть есть возможность обрабатывать объекты по разному, в зависимости от их типа. И более конкретного, передефинировать методы в производных классах.

Динамическое связывание и виртуальные функции надо рассматривать как механизм в язык для ООП для реализации динамичного полиморфизма, не не и рассматривать полиморфизм как следствие применение механизма. Название других видов полиморфизма, реализованных в языках, также не принесло бы никакого вреда.

Ross в [16] делает критический анализ множества популярных современных учебников ООП, которые пользуются C++, Java и VB.NET с точки зрения как выяснен полиморфизм. Он рекомендует преподавателям быть внимательным при подборе учебника. Критерием должно быть посследовательнное, очищенное и пространное рассмотрение полиморфизма.

Так как понятие полиморфизм может звучать слишком абстрактно для студентов, его можно иллюстрировать примерами и аналогиями из реальной жизни. В [8] использован пример с глаголом "закрой", обозначающий разные действия, когда применяется к разным объектам. Операция „закрытие", примененная к объекту „дверь", объект „банковый счет" и объект „окно программы" вызывает разные действия, т.е., какое действие совершить зависит от объекта, на который направлено. Другой удачный пример есть в [10] – ректор отправляет инструкцию всем студентам, и студенты каждой специальности (филологи, информатики, историки, спортсмены) совершают разные действия в зависиммости от полученной инструкции. Можеть быть чаще всего полиморфизм применяется при графичных интерфейсах программ и операционных систем. И примеры в них удачные. Еще один хороший пример, близкий до студентов – это библиотека Microsoft Foundation Classes. В ней все визуальные контроли виртуально наслеживают класс CWindow, который капсулирует основной интерфейс функциональности контролей. CWindow предлагает методы как UpdateWindow(), BeginPaint(), GetWindowText(), и т.д. Каждый из этих методов можно передефинировать в каждом классе, который наслеживает прямо или нет от CWindow. Такими являются классы как CButton, CRadioButton, CListBox, и др.

Важным условием для успешного обучения ООП является правильный подбор задач, которые рассматриваются на лабораторных занятиях и задаются для самостоятельной работы. Они должны быть такими, чтобы полиморфизм стал естественной частью решения.

Автор популярной обучающей среды программирования BlueJ - Michael Kölling [1] ставит акцент на другой важный аспект – на визуальное прослеживание полиморфичного поведения способом интерактивного вызыва одного и того же метода для объектов разных классов и предачи параметров объектов производных классов к методу, требующему псевдонима объекта базового класса. Таким образом разницы в поведении, вызванные полиморфизмом, можно рассматривать директно.

В средах научной общности верх берет мнение, что изучение полиморфизма и его реализации в языках для ООП - задача трудная [18]. Анализ проблем в [15] указывает на частичную причину в том, что она состоит в трудности дефинировать понятия вне контекста примеров. Исследование в [13] указывает на основную причину для большинства проблем как "неспособность студентов понять что происходит с их программой в памяти, так как они не могут сделать ясную мысловную модель исполнения программы". Студенты трудно привыкают с нотацией указателей и псевдонимов в совместимости между классами наследственной йерархии. Отметим, что даже студенты, у которых уже выработано концептуальное понимание, иногда неспособны понять почему полиморфизм является таким важным компонентом ООП. Притом, для них трудно решать где в программе надо использовать полиморфизм и где нет. В [15] есть твердение, что эту проблему можно решить путем мотивации студентов, причем предложена методология, требующая программирования игр.

Конечно, есть и авторы [2][3], которые твердят что динамичный полиморфизм одновременно является и фундаментальной, и легкой для усвоения концепцией ООП, с который студенты сталкиваются постоянно в ежедневии и хорошо разбираются в ней. Поэтому ее изучение должно начаться как можно раньше (даже с первого дня обучения компьютерным наукам). Подход, который предлагает Bergin в [3] в связи с изучением полиморфизма, требует применения некоторых элементарных шаблонов проектирования [20], которые студенты должны усвоить еще прежде чем начали писать код.

Выводы и рекомендации, связанные с полиморфизмом и его применением:

1. Для правильного понимания механизма динамического связывания удачно его сравнение со статическим (называемым еще ранним) связыванию, как это сделано в [19][21] и демонстрация преимуществ и не-

достатки двух видов путем моделирования одной и той же реальной проблемы динамичным связыванием или без него.

2. Изучение реализациии механизма динамического связывания в компиляторе конкретного языка, которым пользуемся в курсе ООП , помогает студентам приобрести представление и проектировать модель работы программы, а потом научиться эффективно применять полиморфизм. Отладка программ является эффикасным средством достижения этой цели.

3. В стандарт языка C++ [7] вводится термин полиморфичной класс. Это такой класс, которыи дефинирует или наслеживает виртуальные функции. Использование этого термина в курсе ООП будет содействовать точному акцентированию на полиморфичное поведение объектов.

4. Чтобы могли программировать эффективно, студенты должны знать и разграничивать полиморфизм во всех его формах, а не только динамичный полиморфизм. Классификация видов полиморфизма в курсе ООП является хорошей предпосылкой для этого.

5. Студенты должны сначала научиться разграничивать полиморфизм и моделировать, пользуясь им, а потом познакомиться с его реализацией в конкретном языке для ООП (дефинировать и применять виртуальные функции и абстрактные классы).

6. Подходить осторожно при подборе учебников и учебных принадлежностей, а также и при выборе примеров и задач для самостоятельной работы.

7. Если возможно, пользоваться дидактическими средствами визуализации полиморфичного поведения объектов.

Литература:

[1] Barnes, D., Kolling, M., „Objects First with Java: A Practical Introduction Using BlueJ, 2nd ed.”, Prentice Hall, 2005, 520 pages

[2] Bergin J., Wallingford E., Caspersen M., Goldweber M., Kolling M., “Teaching polymorphism early”, Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education, Caparica, Portugal, ACM, 2005, Pages: 342 – 343

[3] Bergin, J., „Teaching Polymorphism with Elementary Design Patterns”, Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, Anaheim, CA, USA, ACM, 2003, Pages: 167 - 169

[4] Bettini, L., Capecchi, S., Venneri, B., „Featherweight Java with MultiMethods”, Proceedings of the 5th international symposium on Principles and practice of programming in Java, Lisboa, Portugal, ACM, 2007, Pages: 83 - 92

[5] Clifton, C., Millstein, T., Leavens, G., Chambers, Cr., „MultiJava: Design Rationale, Compiler Implementation, and Applications”, ACM Transactions on Programming Languages and Systems, Vol. 28, No. 3, May 2006, Pages 517–575.

[6] Eckel, B., “Thinking in Java”, Prentice Hall, 4 edition, 2006, 1150 pages

[7] International Organization for Standards, “International Standard ISO/IEC 14882. Programming Languages - C++, 2nd edition”, 2003, 757 pages.

[8] Kalev, D., “ANSI/ISO C++ Professional Programmer's Handbook”, Que, 1999, 356 pages

[9] Kumar, R., Agrawal, V., Mangolia, A., „Realization of Multimethods in Single Dispatch Object Oriented Languages”, ACM SIGPLAN Notices, Volume 40 , Issue 5 (May 2005), ACM, 2005, Pages: 18 - 27

[10] Lafore, R., “C++ Interactive Course: Fast Mastery of C++”, Waite Group Press, 1996, 944 pages

[11] Lippman, St., Lajoie, J., “C++ Primer (3rd edition)”, Addison Wesley, 1998, 1264 pages

[12] McGregor, J., Sykes, D., „A Practical Guide to Testing Object-Oriented Software (The Addison-Wesley Object Technology Series)”, Addison-Wesley Professional, 2001, 416 pages

[13] Milne, I., Rowe, G., „Difficulties in Learning and Teaching Programming - Views of Students and Tutors”, Education and Information Technologies, Volume 7, Number (March, 2002), Springer Netherlands, Pages 55-66

[14] Pirkelbauer, P., Solodkyy, Y., Stroustrup, Bj., „Open Multi-Methods for C++”, Proceedings of the 6th international conference on Generative programming and component engineering, Salzburg, Austria, ACM, 2007, Pages: 123 - 134

[15] Purewal, T., Bennett Chris, „A framework for teaching polymorphism using game programming”, Journal of Computing Sciences in Colleges, Volume 22 , Issue 2 (December 2006), Consortium for Computing Sciences in Colleges, USA, 2006, Pages: 154 - 161

[16] Ross, J., „Polymorphism in Decline?”, Journal of Computing Sciences in Colleges, Volume 21 , Issue 2 (December 2005), Consortium for Computing Sciences in Colleges , USA, 2005, Pages: 328 - 334

[17] Stroustrup, Bj., „An Overview of the C++ Programming Language, From ‘Handbook of Object Technology’ (Editor: Saba Zamir)”, CRC Press LLC, Boca Raton. 1999, 1168 pages

[18] Walter, J., Kalev, D., “The Waite Group’s C++ How-To”, SAMS, 2001.

[19] Богданов, Д., “Обектно-ориентирано програмиране със C++”, Техника, София, 1994, 239 стр.

[20] Гама, Е., Хелм, Р., Джонсън, Р., „Design Patterns (Шаблони за дизайн)”, СофтПрес, 2005, 464 стр.

[21] Тодорова, М., „Програмиране на C++, Първа и Втора част”, СИЕЛА, София, 2002