ТЕХНИЧЕСКИЕ НАУКИ

УДК: 519.683

Романов С. С.
Магистрант 2 курса,
Хакасский Государственный Университет им. Н. Ф. Катанова

КЛЮЧЕВЫЕ ПОНЯТИЯ И ОСОБЕННОСТИ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ

В работе рассматриваются ключевые понятия и особенности объектно-ориентированного-программирования (ООП), а также обстоятельства возникновения и развития данной парадигмы.

Ключевые слова: *ООП, наследование, инкапсуляция, полиморфизм, программирование, объекты, методы, классы, экземпляры класса, свойства.*

В настоящее время объектно-ориентированный подход при разработке систем различной степени сложности общепризнан, отмечает автор [5, с. 3]. Более того, он применяется не только при разработке, но и при использовании широко распространённых объектно-ориентированных систем.

Недостаток же процедурно-ориентированных языков заключается в наличии двух проблем: неограниченный доступ функций к глобальным данным и тот факт, что разделение данных и функций, которое является основой структурного подхода, плохо отображает картину реального мира [4, с. 34].

Актуальность изучения концепции ООП заключается в том, что ООП является востребованной парадигмой программирования при разработке программного обеспечения, что обусловливает программисту необходимость понимания и применения ООП. Целью работы выступает изучение парадигмы ООП, ее основных понятий и аспектов.

История появления и развития ООП

Авторы [6, с. 160] отмечают, что под ООП понимается технология, появившаяся как реакция на очередную фазу кризиса программного обеспечения, когда методы структурного программирования уже не позволяли справляться с растущей сложностью промышленных программных продуктов.

ООП создавалось, как развитие концепции процедурного программирования, с целью решения некоторых проблем, присущих процессу разработки сложных программных систем при использовании процедурного подхода к программированию и алгоритмического подхода к декомпозиции. Разумеется, процесс планирования при использовании ООП упрощается, так как объекты программы соответствуют объектам реального мира.

Согласно [8, с. 11], первые программисты писали программы посредством использования машинных кодов. В связи со сложностью подобного подхода возникла необходимость создать программу, которая преобразовывала бы написанный человеком и понятный ему текст в машинные коды. Такая программа получила название «компилятор», а используемый для написания исходного текста программы язык — «языком программирования». Первым компилятором был «Assembler».

Впоследствии было создано множество различных языков программирования, например: C, ADA, FoxPro, Fortran, Basic, Pascal и другие. Следующей же ступенью развития стало объектно-ориентированное программирование [там же, с. 14]. Появились соответствующие данной парадигме языки, например: C++, Object Pascal. Наблюдалась борьба между скоростью разработки программного обеспечения и скоростью выполнения программного кода.

После процедурного подхода к программированию, следующей ступенью в развитии технологий программирования стало появление ООП [там же, с. 43] — программный код теперь не представляется «плоским», а программисту подвластны не только процедуры и функции, но и целые классы.

На сегодняшний день существуют более 2500 языков программирования высокого уровня [там же]. Это объясняется направленностью конкретных языков на определенные предметные области, а также тем, что появление новых языков дает возможность разработчикам решать все более сложные задачи.

Язык Simula, разработанный в 1960-х годах считается общим предком большинства современных объектных и объектно-ориентированных языков. Данный язык дополнил идеи языка ALGOL концепцией инкапсуляции и наследования. На рисунке ниже представлена генеалогия наиболее значимых и популярных объектных и объектно-ориентированных языков программирования, где интенсивность разработки языка показана длиной прямоугольника, а стрелочки отображают влияние одних языков на другие.

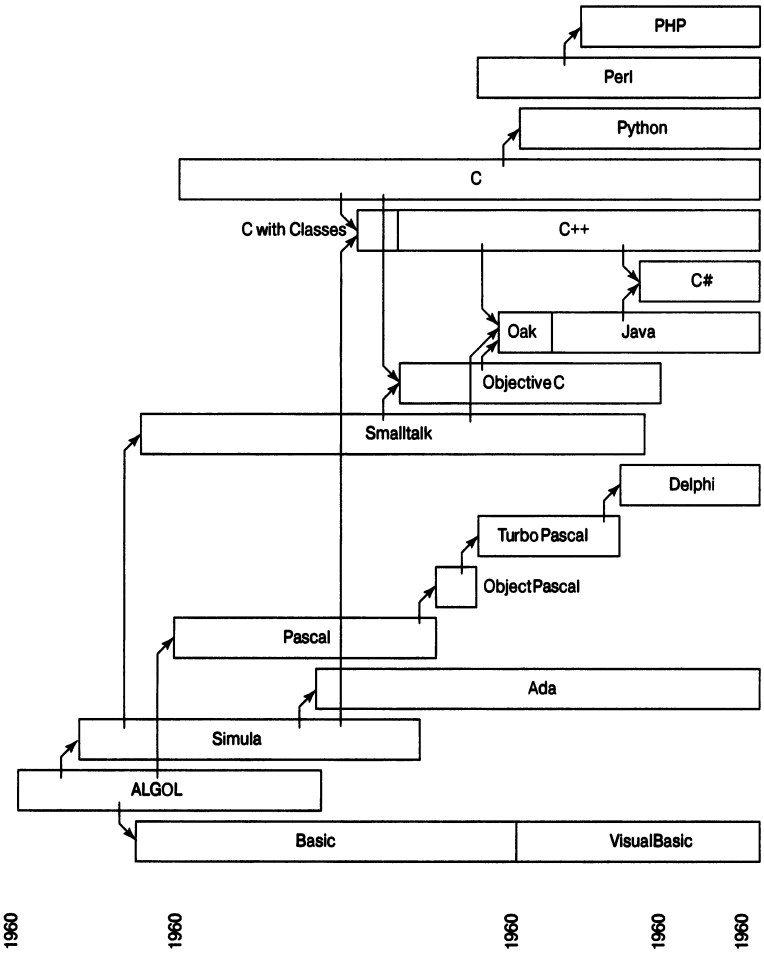

Рисунок 1. — Генеалогия наиболее важных языков программирования, согласно [там же, с. 577]

Язык «Симула» предлагал революционные идеи, такие как: объекты, классы, виртуальные методы, но которые не были тогда восприняты таковыми. Новый, отличный от процедурного, взгляд на программирование предложили Алан Кэй и Дэн Ингаллс в языке Smalltalk, в котором «класс» стал ключевой идее для остальных конструкций языка. Именно этот язык и считается первым широко известным объектно-ориентированным языком программирования.

В наше время количество прикладных языков программирования, реализующих парадигму ООП превышает количество языков, реализующих иные парадигмы. Наиболее популярные языки программирования (C++, Delphi, C#, Java и др.) воплощают объектную модель Симулы. Примерами языков, базирующихся на модели Smalltalk, являются Python и Ruby.

Также выделяются объектные языки, к которым относят, согласно [2, с. 576], те, которые обеспечивают абстракцию данных и создание классов. К объектно-ориентированным же языкам относятся объектные языки, поддерживающие наследование и полиморфизм.

Основные понятия ООП

Согласно [7], «ООП — это методология программирования, которая основана на представлении программы в виде совокупности объектов, каждый из которых является реализацией определенного класса (типа особого вида), а классы образуют иерархию на принципах наследуемости».

При разработке приложений с использованием ООП может применяться объектно-ориентированная декомпозиция, тогда как прежде, в случае применения процедурного подхода к программированию, имела место алгоритмическая декомпозиция.

Первым бросающимся в глаза отличием ООП от структурного программирования является использование классов. Класс — это тип, определяемый программистом, в котором объединяются структуры данных и функции их обработки. Конкретные переменные типа данных «класс» называются экземплярами класса, или объектами [6, с. 162].

Класс содержит константы и переменные, называемые **полями**, а также выполняемые над ними операции и функции. Функции класса называются **методами** (другое название — функции-члены). Предполагается, что доступ к полям класса возможен только через вызов соответствующих методов. Поля и методы являются элементами (членами) класса.

Методы, расположенные в открытой части, формируют интерфейс класса и могут свободно вызываться клиентом через соответствующий объект класса. Доступ к закрытой секции класса возможен только из его собственных методов, а к защищенной — из его собственных методов, а также из методов классов-потомков.

Класс, согласно [8, с. 43] — это «Совокупность свойств, методов и событий». При этом «совокупность» означает, что функционирование класса обеспечивается в совокупности методами, свойствами и событиями.

Под **объектом**, согласно [8, 1], понимается экземпляр класса. Объект функционирует как единое целое, реагируя соответствующими методами на соответствующие события класса. Разница между понятиями объекта и класса заключается в том, что посредством класса осуществляется описание какой-либо сущности, что работает как шаблон, основа. Например, в случае с Delphi, для того, чтобы добавить на форму кнопку, следует объявить класс, описать свойства, методы и события, и, при переносе кнопки на форму, создается экземпляр этой кнопки, т.е. объект.

Немаловажно, что каждый класс должен включать в себя два обязательных метода: создать объект (конструктор), уничтожить объект (деструктор), а процесс создания объекта именуется инициализацией [8, с. 45]. Эти методы выделяют под свойства объекта память и освобождают ее, а также заполняют свойства значениями по умолчанию.

Существует так называемые **права доступа**, в зависимости от которых методы и классы видны другим классам. В Delphi, например, это public, private, protected, published [там же, с. 51]. Открытыми для доступа должны быть лишь некоторые возможности класса.

Роль классов в ООП заключается в том, что они выполняют две функции, всегда разделенные до появления объектно-ориентированных технологий: класс — это одновременно и модуль и тип [1].

Под модулями понимаются структурные единицы, из которых состоит программа. Поскольку модуль всегда рассматривается как синтаксическая концепция, то разбиение на

них влияет только на форму записи исходных кодов, не определяя их функциональность. Тип — это статистическое описание определенных динамических объектов — элементов данных, обрабатываемых в процессе выполнения программы.

Одним из важнейших достижений в области ООП является методология паттернов проектирования, иногда называемых шаблонами проектирования [6, с. 202].

Паттерн — это описание взаимодействия объектов и классов, адаптированных для решения общей задачи проектирования в конкретном контексте. Паттерны выявляются по мере накопления опыта разработок, когда программист использует одну и ту же схему организации и взаимодействия объектов в разных контекстах. Паттерны предназначены, прежде всего, для инкапсуляции изменений.

Основные свойства ООП

К основным принципам ООП относятся [7]: пакетирование или инкапсуляция; наследование; полиморфизм; передача сообщений.

Наследование (или наследственность) — определение объекта и его дальнейшее использование для построения иерархии порожденных объектов с возможностью для каждого порожденного объекта, относящегося к иерархии, доступа к коду и данным всех порождающих объектов [9]. В [6] представлено несколько иное определение данного термина: «Наследование — механизм получения нового класса из существующего путем его дополнения или изменения».

Как структурная, так и объектно-ориентированная методологии ставят перед собой цель построения дерева иерархии взаимосвязей между объектами. При этом структурная иерархия формируется по простому принципу разделения целого на составляющие, тогда как в случае объектно-ориентированной иерархии отражается наследование свойств вышележащих типов объектов нижележащим типам объектов [7].

Абстрактными типами называют типы верхних уровней иерархии, которые, как правило, не имеют экземпляров. Конкретными экземплярами зачастую обладают типы нижних уровней иерархии.

Наследственность в ООП — это его основа, как отмечает автор [8, с. 49]. Следует понимать, что потомок — класс, от которого происходят другие классы, наследует свойства предка — класса, который происходит либо порожден из другого класса. Следовательно, потомок всегда «знает», какими он обладает свойствами, а предок не может «знать» свойства своего потомка, поскольку не может «знать» те свойства, которые будут добавлены в новый класс. Данный метод наследования и принят в объектно-ориентированных языках, согласно [там же, с. 50].

Именно наследование дает возможность повторно использовать уже существующий код, как отмечается автором [3, с. 207]. С целью применения существующего кода, программист создает новый класс на базе уже существующих классов. В итоге, наследование выполняет две функции: предотвращение дублирования кода, развитие работы в нужном направлении. Отношения между родительскими классами и его потомками именуется термином **«Иерархия наследования».**

Под **полиморфизмом** подразумевается свойство родственных объектов решать схожие по смыслу проблемы разными способами, отмечается в [7]. Например, действие «бежать» свойственно большинству животных. Но каждое из них действует различным образом.

Полиморфизм — присваивание действию одного имени, которое затем совместно используется вниз и вверх по иерархии объектов, причем каждый объект иерархии выполняет это действие способом, именно ему подходящим [9]. «Полиморфизм» представляет собой способность обладать несколькими формами, отмечается автором [1]. В объектно-ориентированной разработке это относится к сущностям (элементам структур данных), способным в процессе выполнения присоединяться к объектам разных типов.

Полиморфизм дает возможность применения одних и тех же функций для решения различных задач и находит свое отражение в том, что под одним именем содержатся разные действия, наполнение которых зависит от типа объекта. Производные одного базового класса считаются **полиморфными**. Это подразумевает, что им присущи как общие характеристики, так и собственные свойства.

Касательно средств, которые связаны с возможностью реализации полиморфизма, то следует упомянуть следующие виды методов в ООП:

- Статические методы класса включаются в код программы в процессе ее компиляции. Таким образом, до запуска программы уже известно, какая процедура будет вызвана в определенной точке.
- Виртуальные методы, которые подключаются к основному коду на этапе выполнения программы и дают возможность определить тип и конкретизировать экземпляр объекта в процессе исполнения, а затем вызвать методы этого объекта. Данный механизм обеспечивает полиморфизм и именуется также **поздним связыванием.**

Автор [8, с. 50] объясняет возможность полиморфизма на примере гаража, потомком которого выступает дом: «Представим, что у гаража дверь открывается вверх, а у дома должна открываться в сторону. Дом происходит от гаража, поэтому у него дверь будет открываться тоже вверх. Как тогда быть? Вы просто должны изменить (переписать) у дома процедуру, отвечающую за открытие двери. Тогда дом получит все свойства гаража, но при открывании двери подставит свою процедуру». Иначе говоря, полиморфизм — различная реакция объектов разных иерархий на одно и то же событие.

Полиморфизм дает возможность создавать множественные определения для операций и функций. Какое именно определение будет использоваться, зависит от контекста программы. ООП предоставляет возможности, связанные с полиморфизмом, такие как: шаблоны функций, перегрузка функций, перегрузка операций, использование виртуальных методов, шаблоны классов [6, с. 163]. Перегрузка операций позволяет применять для собственных классов те же операции, что используются для встроенных типов C++. Виртуальные методы обеспечивают возможность выбрать на этапе выполнения нужный метод среди одноименных методов базового и производного классов. Шаблоны классов позволяют описать классы, инвариантные относительно типов данных.

Согласно [8, с. 51], под **инкапсуляцией** понимается «свойство, благодаря которому разработчику, использующему определенный строительный блок (код), не нужно знать, как он на самом деле реализован и работает для корректного использования этого строительного блока».

Другое преимущество инкапсуляции заключается в том, что разработанную программу легче модифицировать, поскольку при сохранении интерфейса класса можно менять его реализацию, и это не затронет внешний программный код (код клиента).

Выводы

Появление парадигмы ООП обусловливалось необходимостью решения возникающих проблем при построении сложных программных систем. Наиболее важными событиями, связанными с историей ООП считаются появление языка «Симула», содержащего базовые аспекты ООП, а также возникновение языка Smalltalk — первого объектно-ориентированного языка программирования.

Основными понятиями ООП являются: класс, объект, метод, свойство, поле, наследование, полиморфизм, инкапсуляция, права доступа.

Основными свойствами объектно-ориентированного языка программирования являются: наследование, полиморфизм, инкапсуляция. Современное программирование базируется не только на идеях ООП, но и на применении паттернов проектирования, аккумулирующих наиболее удачные решения типичных проблем, неоднократно возникавших при разработке ПО.

Литература

1. Бертран М. Основы объектно-ориентированного программирования [Электронный ресурс] / Бертран М. — Электронное издательство «Интернет-университет информационных технологий — ИНТУИТ.ру», 2005 г.

2. Буч Г., Максимчук Р. А., Энгл М. У., Янг Б. Дж., Коналлен Д., Хьюстон К. А. Объектно-ориентированный анализ и проектирование с примерами приложений [Текст] / Г. Буч, Р. А. Максимчук, М. У. Энгл, Б. Дж. Янг, Д. Коналлен, К. А. Хьюстон. — 3-е изд. — М.: ООО «И. Д. Вильямс», 2008. — 720 с.

3. Лаптев В. В. C++. Объектно-ориентированное программирование: Учебное пособие [Текст] / В. В. Лаптев. — СПб.: Питер. 2008. — 464 с.

4. Лафоре Р. Объектно-ориентированное программирование в C++ [Текст] / Р. Лафоре. — 4-е изд. — СПб.: Питер, 2004. — 928 с.

5. Медведев В. И. Особенности объектно-ориентированного программирования на C++/CLI, C# и Java. [Текст] / В. И. Медведев. — 2-е изд., испр. и доп. — Казань: РИЦ «Школа», 2010. — 444 с.

6. Павловская Т. А., Щупак Ю. А. C/C++. Структурное и объектно-ориентированное программирование: Практикум [Текст] / Т. А. Павловская, Ю. А. Щупак. — СПб.: Питер, 2011. — 352 с.

7. Программирование. ООП в Pascal-Паскаль [Электронный ресурс] // Pascal-Паскаль: Исходники Программирование Pascal-Паскаль. — URL: http://www.pascal.helpov.net/index/object-oriented_programming_pascal_programming#pascal-oop_8 (дата обращения: 04.03.2016).

8. Фленов М. Е. Библия Delphi [Текст] / М. Е. Фленов. — 3-е изд., перераб. и доп. — СПб.: БХВ-Петербург, 2011. — 688 с.

9. Borland Pascal. Руководство пользователя: Глава 9. Объектно-ориентированное программирование [Электронный ресурс] // Библиотека on-line. — URL: http://citforum.ru/programming/bp70_ug/bp70ug_09.shtml (дата обращения: 18.03.2016).