## 8 ВВЕДЕНИЕ В ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ (ООП)

Объектно-ориентированное программирование (ООП) – это парадигма (метод) программирования, основанный на использовании классов.

Класс – это тип данных, который определяет программист с помощью ключевого слова class. По формату объявления тип «класс» похож на тип «структура».

Объект – это переменная типа класс.

Но ООП – это значит не просто написать слово «class» вместо слова «struct».Это метод программирования, который, наряду с использованием типа class, предполагает, что программист при написании программного обеспечения следует трем принципам:

1) инкапсуляция;

2) наследование;

3) полиморфизм.

При этом ООП не отменяет стиль структурного программирования, структурное программирование поддерживается внутри ООП.

8.1 Немного философии: ООП и моделирование реальных объектов

Программы предназначены для моделирования предметов, процессов и явлений реального мира. Поэтому в языке программирования необходим удобный инструмент для представления моделей реальных предметов и явлений. Кстати, язык С++ был разработан Страуструпом для моделирования телефонных коммутационных систем.

Если в программе мы хотим моделировать поведение какого-то реального объекта (физической сущности), то нам необходимо создать модель (чертеж, шаблон) объекта. Это и будет класс. В эту модель (в этот класс) мы включим некоторые существенные для нашей задачи свойства (или характеристики) объекта и методы (или функции) изменения этих свойств. То есть реальный объект имеет две основные характеристики:

1) состояние, которое определяется набором свойств объекта;

2) поведение, которое определяется набором функций объекта.

Рассмотрим пример. Предположим, реальный объект (физическая сущность), который мы моделируем – это человек-студент. Его состояние – это набор его свойств: рост, вес, пол, количество зубов, степень интеллекта и т.д. Его поведение – это некоторые его действия, влияющие на его свойства. Например, объект может учиться, в результате чего возрастет его степень интеллекта; объект может драться, в результате чего уменьшится количество его зубов, и т. д.

Итак, тип «класс» включает в себя свойства (характеристики) объекта и методы (функции) для изменения этих свойств.

8. 2 Объявление класса

Ниже схематично объявлен тип класса с именем «chel» и две переменные a1, a1 этого типа.

class chel {

Переменные-члены класса;

Функции-члены класса;

};

chel a1, a2;

Терминологически часто используются и другие словосочетания. Так, вместо «переменные-члены класса» и «функции-члены класса» часто говорят «переменные» и «функции», «данные» и «методы», «данные» и «код», соответственно. Переменные a1, a2 часто называют «объекты, или экземпляры класса».

Переменные и функции класса называют членами (элементами) класса. Члены могут иметь атрибуты (спецификаторы доступа):

- public (открытые),

- private (закрытые),

- protected (защищенные).

По умолчанию все члены класса считаются закрытыми (private), то есть они доступны лишь функциям-членам этого класса.

Спецификатор public открывает доступ к членам класса из других частей программы.

Спецификатор protected необходим только при наследовании классов.

Зона влияния спецификатора простирается до следующего спецификатора или до конца объявления файла.

Обычно все данные-члены объявляют закрытыми. Это обеспечивает инкапсуляцию данных (их закрытость, сохранность).

Функции-члены класса имеют доступ ко всем элементам своего класса, включая закрытые члены.

Если к члену класса обращаются извне, надо писать «имя\_объекта.имя\_члена».

Если к члену класса обращаются из своих же методов, то указывается просто имя члена класса.

8.3 Три кита ООП: инкапсуляция, наследование, полиморфизм

ООП – это не просто использование классов.

ООП предполагает определенный подход к написанию программ, включающий в себя 3 свойства:

- инкапсуляция,

- наследование,

- полиморфизм.

8.3.1 Инкапсуляция

Инкапсуляция, или «все в одной капсуле» - это механизм, связывающий вместе код и данные, которыми он манипулирует, а также обеспечивающий их защиту от внешнего вмешательства и неправильного использования. В объектно-ориентированном программировании код и данные помещаются в «черный ящик», который называется объектом.

Внутри объекта код и данные могут быть закрытыми (private) и открытыми (public). Это атрибуты доступа (есть еще protected). Закрытые код и данные доступны только внутри объекта, и к ним нельзя обратиться извне. Открытые код и данные доступны из любого места программы.

Рекомендуется все данные делать закрытыми. Открытая часть кода (открытые функции-методы) часто называется интерфейсом объекта. Этот интерфейс обеспечивает взаимодействие внешней программы с объектом, с его закрытыми элементами.

Итак, основные средства инкапсуляции – это объект и атрибуты (спецификаторы) доступа. При инкапсуляции доступ к закрытым членам класса строго контролируется.

8.3.2 Наследование

Наследование – это процесс, в ходе которого один объект может приобретать свойства другого.

Наследование позволяет создавать иерархию классов, уточняя их свойства от самых общих до более конкретных.

Процесс наследования начинается с определения базового класса (предка), свойства которого будут общими для всех его наследников. Наследники базового класса называются производными классами (наследниками). Производный класс обладает всеми свойствами базового класса и своими специфическими особенностями.

Наследование позволяет повторно использовать ранее написанные коды. Это удобно, экономит время и усилия.

По сути, наследование поддерживает концепцию иерархической классификации. А ведь все знания организованы по принципу иерархической классификации.

Схематичный пример наследования. Имеем базовый класс «Человек», его свойства: рост, вес. У этого базового класса есть два производных класса: «Мальчик» и «Девочка». Для этих производных классов уже будут определены свойства рост и вес, плюс к этому для каждого производного класса можно определить еще свои специфические свойства.

8.3.3 Полиморфизм

Полиморфизм (или многоформие) реализует принцип «один интерфейс, несколько методов».

Полиморфизм позволяет упростить программу, создавая один интерфейс для выполнения разных действий. Выбор конкретного действия (метода) осуществляет компилятор. Программисту надо лишь правильно применять общий интерфейс.

Язык С++ поддерживает статический и динамический полиморфизм.

Примеры статического полиморфизма – это перегрузка функций и операторов.

Перегрузка функций – это использование функций с одинаковыми именами, но с разными объявлениями параметров: тип и/или количество параметров должны быть разными. Нельзя перегрузить функции, отличающиеся только типом возвращаемого значения, так как компилятор не сможет правильно распознать нужный вариант функции.

Пример. В языке С существует три функции определения абсолютного значения: abs (для int), labs (для long int), fabs (для double). В языке С++ их можно назвать одинаково и перезагрузить. Плюс для программиста: не надо запоминать три названия функций, нужно запомнить только одно.

Динамический полиморфизм реализуется с помощью виртуальных функций и в сферу наших интересов на текущем этапе не входит.

8.4 Конструкторы и деструкторы

Этот подраздел рекомендуется читать после ознакомления с 1 и 2 примером.

Автоматическую инициализацию части или всего объекта в момент его создания позволяет осуществить конструктор. Конструктор – это особая функция-член класса, имя которой совпадает с именем класса. Отметим, что функции-конструкторы в принципе не могут возвращать значение.

Конструкторы обычно ничего не вводят и не выводят, а просто выполняют различные виды инициализации.

Конструктор автоматически вызывается в момент создания объекта, то есть при его объявлении.

Антипод конструктора – это деструктор. Деструктор – это особая функция-член класса, имя которой совпадает с именем класса, но перед ним ставится знак «~» (тильда). Деструктор объекта вызывается автоматически при разрушении объекта (например, локальные объекты разрушаются при выходе из блока, в котором были созданы).

Конструкторы и деструкторы не являются обязательными для написания, но лучше их всегда писать, даже если они пустые.

Конструкторам можно передавать аргументы, предназначенные для инициализации объекта. Параметры конструктора задаются так же, как и для любой другой функции. Таким образом получается конструктор с параметром.

Конструкторов с параметром может быть несколько. Это удобно, когда необходимо по-разному создавать объекты одного и того же класса. Но каждый конструктор должен иметь уникальный набор параметров (количество аргументов и/или их типы должны быть разные).

Итак, у одного класса может быть несколько конструкторов. Конструкторы могут быть открытыми и закрытыми.

У класса может быть только один деструктор, он должен быть открытым. Деструктору нельзя передать значения. Основная задача деструктора – освобождение памяти, занимаемой объектом; закрытие файла, ранее открытого объектом.

8.5 Путеводитель по примерам

Примеры, приведенные в отдельных файлах class1, class2 и т.д., рекомендуется смотреть в порядке нумерации их названий. Каждый очередной пример – это предыдущий пример, в который добавлен какой-то новый элемент. Читайте комментарии-заголовки к примерам. Примеры с номерами 3 и 8 отсутствуют, так как на текущем этапе оказались не нужны.

Пример class1. Создание класса «Прямоугольник».

Пример class2. Добавлен класс – наследник «Цветной прямоугольник»Создание класса «Прямоугольник».

Пример class4. Добавлены пустые конструктор и деструктор. Они ничего не делают, это просто иллюстрация того, как их называют и объявляют.

Пример class5. Конструктор и деструктор выводят сообщения для иллюстрации своей работы. При запуске программы в момент создания и уничтожения объекта эти сообщения будут выводиться на экран.

Пример class6. Два вида конструкторов: без параметров и с параметрами.

Пример class7. Указатель на объект. Новый код добавлен только в основной программе.

Пример class9. Указатель this.

8.6 Замечания по организации функций-элементов класса

8.6.1 Часто в классах оформляют функции-члены, названия которых начинаются с set и get. Традиционно функции-члены с именами, начинающимися с set, задают значения закрытых данных, а функции-члены, названия которых начинаются с get, извлекают и возвращают эти значения.

Желательно, для каждой переменной-члена класса писать отдельные функции set и get.

Желательно, чтобы функция set проверяла допустимость присваиваемых значений, и возвращала значения true (1) и false (2), соответственно.

Желательно, любые изменения значений членов-переменных класса делать через set. Например, инициализацию переменных-членов в объекте лучше проводить в конструкторе через функцию set.

8.6.2 Методы можно объявлять непосредственно внутри класса. Но лучше внутри класса указывать прототипы, а объявлять методы отдельно.

8.6.3 Если необходимо написать функцию-член класса для сравнения двух объектов, то имеется в виду метод, который по некоторому ключу сравнивает текущий объект (this), через который вызван этот метод, с другим объектом, который передается этому методу по списку параметров. Другими словами, в функцию передается как параметр один объект, а второй – это текущий объект this, через который функция вызывается.