33. Объектно-ориентированное программирование. Перегруженные функции. Конструкторы. Перегрузка операций. указатели. шаблоны

Объектно-ориентированное программирование – это методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования. Объектно-ориентированная модель программирования представляет собой методику анализа, проектирования и написания приложений с помощью объектов. Используя ООП, программист может моделировать в своём приложении материальные явления реального мира.

ООП значительно упрощает процесс разработки программы, так как свойства и методы объектов подчиняются простейшей модели безопасности. Если необходимо сделать так, чтобы приложение могло свободно обращаться к объекту, то нужно организовать доступ к нему через открытые свойства и методы. Закрытые свойства и методы используются только самим объектом и не могут использоваться в других процедурах.

Объект — это “строительный блок” в ООП-приложении. Такой строительный блок инкапсулирует часть приложения — процесс, порцию данных или какой-то более абстракный объект. Объекты в языке C# создаются из типов, как и хорошо знакомые составные переменные. Для типа объекта в ООП имеется специальное название — класс. Определения классов позволяют создавать объекты — т.е. реальные именованные экземпляры класса. Понятия “экземпляр класса” и “объект” эквивалентны, но термины “класс” и “объект” означают совершенно разные вещи.

Свойства и поля обеспечивают доступ к содержащимся в объекте данным. Эти данные как раз и являются тем, что отличает отдельные объекты друг от друга, поскольку в свой- ствах и полях разных объектов одного и того же класса могут храниться разные значения.

Термином метод принято называть предоставляемую объектом функцию. Методы могут вызываться так же, как и любые другие функции, и так же возвращать значения и прини- мать параметры (функции были рассмотрены в главе 6). Методы применяются для доступа к функциональным возможностям объекта

Абстракция данных

Абстрагирование означает выделение значимой информации и исключение из рассмотрения незначимой.

Инкапсуляция

Инкапсуляция — свойство системы, позволяющее объединить данные и методы, работающие с ними, в классе.

Наследование

Наследование — свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствующейся функциональностью

Полиморфизм подтипов

Полиморфизм подтипов (в ООП называемый просто «полиморфизмом») — свойство системы, позволяющее использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта. Другой вид полиморфизма — параметрический — в ООП называют обобщённым программированием.

Класс

Класс является описываемой на языке терминологии исходного кода моделью ещё не существующей сущности (объекта). Фактически он описывает устройство объекта, являясь своего рода чертежом. Говорят, что объект — это **экземпляр** класса. При этом в некоторых исполняющих системах класс также может представляться некоторым объектом при выполнении программы посредством динамической идентификации типа данных. Обычно классы разрабатывают таким образом, чтобы их объекты соответствовали объектам предметной области. Спецификаторы доступа private и public управляют видимостью эле-

ментов класса. Элементы, описанные после служебного слова private, ви-

димы только внутри класса. Этот вид доступа принят в классе по умолча-

нию. Интерфейс класса описывается после спецификатора public. Дей-

ствие любого спецификатора распространяется до следующего специфика-

тора или до конца класса. Можно задавать несколько секций private и
public, порядок их следования значения не имеет.
Поля класса:
26
 могут быть простыми переменными любого типа, указателями, мас-
сивами и ссылками (т.е. могут иметь практически любой тип, кроме типа
этого же класса, но могут быть указателями или ссылками на этот класс);
 могут быть константами (описаны с модификатором const), при этом
они инициализируются только один раз (с помощью конструктора) и не мо-
гут изменяться;
 могут быть описаны с модификатором static, но не как auto, extern
и register.
Инициализация полей при описании не допускается.
3.3 Глобальные и локальные классы
Классы могут быть глобальными (объявленными вне любого блока) и
локальными (объявленными внутри блока, например, внутри функции или
внутри другого класса). Обычно классы определяются глобально.
Локальные классы имеют некоторые особенности:
 локальный класс не может иметь статических элементов;
 внутри локального класса можно использовать из охватывающей его
области типы, статические (static) и внешние (extern) переменные, внеш-
ние функции и элементы перечислений;
 запрещается использовать автоматические переменные из охватыва-
ющей класс области;
 методы локальных классов могут быть только встроенными (inline);
 если один класс вложен в другой класс, они не имеют каких-либо осо-
бых прав доступа к элементам друг друга и могут обращаться к ним только
по общим правилам.

Объект

Сущность в адресном пространстве вычислительной системы, появляющаяся при создании экземпляра класса (например, после запуска результатов компиляции и связывания исходного кода на выполнение).

В центре ООП находится понятие *объекта*. Объект — это сущность, которой можно посылать сообщения и которая может на них реагировать, используя свои данные. Объект — это экземпляр класса. Данные объекта скрыты от остальной программы. Сокрытие данных называется инкапсуляцией.

Наличие инкапсуляции достаточно для объектности языка программирования, но ещё не означает его объектной ориентированности — для этого требуется наличие наследования.

Но даже наличие инкапсуляции и наследования не делает язык программирования в полной мере объектным с точки зрения ООП. Основные преимущества ООП проявляются только в том случае, когда в языке программирования реализован полиморфизм подтипов — возможность единообразно обрабатывать объекты с различной реализацией при условии наличия общего интерфейса.

Классы могут наследоваться друг от друга. Класс-потомок получает все поля и методы класса-родителя, но может дополнять их собственными либо переопределять уже имеющиеся. Большинство языков программирования поддерживает только единичное наследование (класс может иметь только один класс-родитель), лишь в некоторых допускается множественное наследование — порождение класса от двух или более классов-родителей. Множественное наследование создаёт целый ряд проблем, как логических, так и чисто реализационных, поэтому в полном объёме его поддержка не распространена.

Интерфейс — это класс без полей и без реализации, включающий только заголовки методов. Если некий класс наследует (или, как говорят, реализует) интерфейс, он должен реализовать все входящие в него методы. Использование интерфейсов предоставляет относительно дешёвую альтернативу множественному наследованию

3.6 Указатель this Каждый объект содержит свой экземпляр полей класса. Методы места в классе не занимают и не дублируются для каждого объекта. Единственный 30 экземпляр метода используется всеми объектами совместно, поэтому каж- дый нестатический метод класса должен «знать», для какого объекта он вызван. Для этого, при вызове каждого нестатического метода класса, ему не- явно передается указатель на объект, вызвавший его T * const this. Выражение *this представляет собой разыменование указателя и имеет тип определяемого класса. Обычно это выражение возвращается в качестве результата, если метод возвращает ссылку на свой класс (return *this;).

4.1 Конструкторы и их свойства Конструктор предназначен для инициализации объекта и вызывается ав- томатически при его создании. Ниже перечислены основные свойства кон- структоров. 1. Конструктор не возвращает значения, даже типа void. Нельзя полу- чить указатель на конструктор. 2. Класс может иметь несколько конструкторов с разными параметрами для разных видов инициализации (при этом используется механизм пере- грузки). 3. Конструктор, который можно вызвать без параметров, называется конструктором по умолчанию. 4. Параметры конструктора могут иметь любой тип, кроме этого же класса. Можно задавать значения параметров по умолчанию. Их может со- держать только один из конструкторов. 5. Если программист не указал ни одного конструктора, компилятор со- здает его автоматически (кроме случая, когда класс содержит константы и ссылки, поскольку их необходимо инициализировать). Такой конструктор вызывает конструкторы по умолчанию для полей класса и конструкторы ба- зовых классов. 6. Конструкторы не наследуются. 7. Конструктор не может быть константным, статическим и виртуаль- ным (нельзя использовать модификаторы const, virtual и static). 8. Конструкторы глобальных объектов вызываются до вызова функции main. Локальные объекты создаются, как только становится активной об- ласть их действия. Конструктор запускается и при создании временного объекта (например, при передаче объекта из функции). При объявлении объектов вызывается один из конструкторов. При от- сутствии инициализирующего выражения в объявлении объекта вызывается конструктор по умолчанию, при инициализации другим объектом того же типа – конструктор копирования (см. далее), при инициализации полей – один из явно определенных конструкторов инициализации (т.е. конструкто- ров, которым передаются параметры для инициализации полей объекта).

Можно перегружать любые операции, существующие в C++, за исключением: . .* ?: :: # ## sizeof Перегрузка операций осуществляется с помощью функций специального вида (функций-операций) и подчиняется следующим правилам: – сохраняются количество аргументов, приоритеты операций и правила ассоциации (справа налево или слева направо) по сравнению с использова- нием в стандартных типах данных; – нельзя переопределить операцию по отношению к стандартным типам данных; – функция-операция не может иметь аргументов по умолчанию; – функции-операции наследуются (за исключением =). Функцию-операцию можно определить тремя способами: она должна быть либо методом класса, либо дружественной функцией класса, либо обычной функцией. В двух последних случаях функция должна принимать хотя бы один аргумент, имеющий тип класса, указателя или ссылки на класс (особый случай: функция-операция, первый параметр которой – стандарт- ного типа, не может определяться как метод класса). Функция-операция содержит ключевое слово operator, за которым сле- дует знак переопределяемой операции:

8.1 Использование шаблонов функций При создании функций иногда возникают ситуации, когда две функции выполняют одинаковую обработку, но работают с разными типами данных (например, одна использует параметры типа int, а другая типа float). Вы уже знаете, что с помощью механизма перегрузки функций можно исполь- зовать одно и то же имя для функций, выполняющих разные действия и име- ющих разные типы параметров. Однако, если функции возвращают значе- ния разных типов, вам следует использовать для них уникальные имена. Предположим, например, что у вас есть функция с именем max, которая воз- вращает максимальное из двух целых значений. Если позже вам потребу- ется подобная функция, которая возвращает максимальное из двух значений с плавающей точкой, вам следует определить другую функцию, например fmax.

Шаблон определяет набор операторов, с помощью которых ваши про- граммы позже могут создать несколько функций. Программы часто используют шаблоны функций для быстрого опреде- ления нескольких функций, которые с помощью одинаковых операторов ра- ботают с параметрами разных типов или имеют разные типы возвращаемых значений. Шаблоны функций имеют специфичные имена, которые соответствуют имени функции, используемому вами в программе. После того как ваша программа определила шаблон функции, она в даль- нейшем может создать конкретную функцию, используя этот шаблон для задания прототипа, который включает имя данного шаблона, возвращаемое функцией значение и типы параметров. В процессе компиляции компилятор C++ будет создавать в вашей про- грамме функции с использованием типов, указанных в прототипах функций, которые ссылаются на имя шаблона. Шаблоны функций имеют уникальный синтаксис, который может быть на первый взгляд непонятен. Однако после создания одного или двух шаб- лонов вы обнаружите, что реально их очень легко использовать.