**Объектно-ориентированное программирование**

Бутч. ООПроектирование

ООП – одна из парадигм программирования. Использует в качестве основных конструкции. Каждый Объект является экземпляром определенного класса. Классы образуют иерархию на основе принципа наследования. Каждый дочерний класс включает в себя все элементы, присущие родителю с возможностью их расширения и модификации

Программа считается ООП, если выполнены все эти условия.

Элементными Объектной модели являются:

1. Абстракция. Предполагает выделение существенных характеристик некоторого объята, отличающего его от других Объектов. Таким образом четко описываются концептуальные границы с точки зрения некоторого наблюдателя. Абстракция подразумевает разделеие незисимое распределение интерфейса и реализацией. Выделяют 2 вида астрации
   1. Абстракция данных. Предполает, что Объект представляет собой полезную модель некоторой сущности в предметной области
   2. Процедурная абстрация. Предполагает, что Объект состоит из обобщенного множества операций. Каждая из которых выполняет определенную функцию.

Говорят, что абстрация. Представляет каждый класс как совокупность интерфейсов и реализаций. Интерфей обуславливает внешнее часть Объекта, а реализация – механизм обеспечивающий его поведение.

1. Инкапсуляция. Инкапсуляция – процесс разделения элементов абстракции определяющих его структуру и поведение. Она предназначена для изоляции конкретных элементов абстракции от ее реализации. Процесс обьединения в рамках единой структуры данных и методов их обработки с целью сокрытия подробностей связанных с абстракцией данных.
2. Модульность. Свойство системы разделяться на цельные, но слабосвязанные между собой модули.
3. Иерархия. Элемент ранжирования или упорядочивания абстракции, реализованное в принципе наследование. Наследование основанно на отношении является is a. Наследование означает такое отношение между классами, когда один класс заимствует поведение другого класса. Наследование бывает одиночное и множественное. Говорят, что наследование создает иерархию абстракции, в которой подклассы заимствуют свойства одного или нескольких супер-классов.
4. Контроль типов. Определяет правило использования Объектов не допускающее или ограничивающее замену Объектов разных классов.
5. Параллелизм. Свойство отличающее активные Объекты от пассивных. Все Объекты системы разбиваются на 2 типа. Любой Объект имеющий отдельный поток управления называет активным Объектов. Объект способный лишь подвергаться воздействию называется пассивным Объектом. Активные Объекты могут работать независимо друг от друга, периодически обмениваясь специальными сообщениями, что обеспечивает корректность работы системы.
6. Персистентность. Способность объекта преодолевать временные рамки, т. Е. продолжать свое существование после исчезновения своего создателя, либо выходить за рамки пространства.

Преимущества:

* Использование ООП стимулирует не только повторное использование кода, но и повторное использование объектных решений.
* Упрощенное использование объектов за счет создание устойчивых промежуточных форм и отсутствие необходимости изучение внутреннего содержимого класса при известных интерфейса доступа к нему.
* ООП модуль учитывае общепринятный процесс познания от простого к сложному и предполагает написание программы исходя из определения предметной области.

Любой объект характеризуется состоянием, поведением и индивидуальностью. Состояние – характеристика объекта, определяемая набором его свойств и значениями, которые эти свойства имеют в конкретный момент времени. Поведение – действие и реакция объекта, выраженная через изменение состояния этого объекта или предача сообщения другим обьектам. Передача сообщения – операция, которую один объект выполняет с целью изменения состояния другого объекта. Поведение объекта зависит от выполняемых им операций. Наиболее распространеными видами операциями являются следующие:

* Операция – модификатор. Изменяет состояние объекта
* Операция – селектор. Выбирает.
* Операция – итератор. Получает доступ ко всем элементам последовательно.
* Конструктор. Операция, которая создает объект
* Деструктор. Операция, уничтожающая объект

Индивидуальность – то что отличает объект от других.

Класс – множество объектов имеющие общую структуру и общее поведение.

С точки зрения С++ в качестве базы для создания объекта может быть использован тип struct. Основное отличие структуры от класса заключается в том, что в структуре все элементы публичные, а в классе приватные.

Выделяют 3 области видимости:

* Public. Все элементы являются видимыми
* Private. Скрыты
* Protect. Скрыты, но прямые наследники могут получить доступ.

Правилом хорошего тона является объявление всех данных в private.

Основные правила использования конструкторов:

1. Конструктор имеет тоже имя, что и класс
2. Конструктор не возвращает значение
3. Конструктор не наследуются в производных классах
4. Конструктор может иметь параметры по умолчанию
5. Конструктор является функцией, но его нельзя объявить ключевым словом Virtual
6. Невозможно получить адрес конструктора
7. Если конструктор в программе не задан, то он автоматически генерируется средой
8. Конструктор вызывается автоматически при описании объекта
9. Объект содержащий конструктор, нельзя включить в виде компонента в Union
10. Конструктор класса Х не может иметь параметр типа Х, может иметь параметр ссылку на объект типа Х, в этом случае он называется конструктором для копирования

Деструкторы:

1. Имеет тоже самое имя, что и класс, только с тильдой
2. Не возвращает значение
3. Не наследуется
4. Производный класс может вызывать деструкторы своих базовых классов.
5. Деструктор не имеет параметра
6. Только один деструктор в классе
7. Деструктор – функция, и он может быть виртуальным
8. Невозможно получить в программе адрес деструктора
9. Если деструктор не задан в программе, то он автоматически сгенерируется.
10. Деструктор вызывается автоматически при разрушении объекта.

**Области видимости.**

Каждый класс представляет собой отдельную область видимости. Имена членов класса локальной в классе и могут быть доступны извне только при обьявлении их в разделе public. Внутри класса область видимости не зависит от точки обьявления членов класса. Т.е. член класса виден внутри всего класса. Для того чтобы предоставить доступ к глобальной переменной необходимо использовать операцию расширения видимости. (::х).

Каждый класс имеет свою копию данных. В ряде случаев необходимо чтобы все обьекты класса имели доступ к некоторой общей переменной. В этом случае общедоступный член данных должен быть определен в классе. Со спецификатором static. Для того чтобы изменять значение статической переменной, необходимо обращаться к нему через указатель имени класса без конкретизации объекта, с расширением области видимости.

*Int Example::x = 67;*

Некоторые функции класса могут быть объявлены таким образом, чтобы им было разрешено только читать значения член-данных данного класса без модификации значения. Подобные функции должны быть объявлены со спецификатором const (очевидно же). При попытке модификации компилятор выдает ошибку. В общем случае любой экземпляр класса может быть объявлен как константный, тогда работать с член-данными этого класса можно только через константные функции.

Указатель this.

На практике часто приходится использовать доступ к текущему экземпляру класса с явным указанием его местоположением. В этих случаях необходимо обращаться к компонентам объекта непосредственно по их имени без указания имени объекта. Каждый объект имеет неявный указатель на свою структуру. Неявный указатель размещается в первых 4-х байтах области памяти выделенной под структуру обьекта. Любое обращение к член-данному внутри метода приводит к подстановке вместо обращения неявного указателя this. (this->day)

*Data today, my\_day;*

*Today.out();*

*My\_day.out(); // Cout << this->day << “ “ << month;*

В ряде случаев необходимо использовать конструкции позволяющие инициализировать данные через заголовок конструктора. Чаще всего такая необходимость в случае когда в качестве член-данных выступают объекты другого класса или выполняется наследование свойств другого класса. Инициализация данных через конструктор выполняется через указание двоеточия, после чего должно должны быть определены имя параметра подлежащие инициализации и инициализирующее значение.

**Основные свойста и правила использования THIS:**

* Каждый оъект имеет свой указатель this
* This указывает на начало своего объекта в памяти
* This не надо дополнительно объявлять
* This передается как скрытый парамер во все нестатические член-функции своего объекта.
* This – локальная переменная недоступная за пределами объекта.

В языке С++ одна и та же функция не может быть компонентом двух или более различных классов, однако в некоторых ситуациях необходимо иметь доступ из одной функции к локальным компонентам разных классах.

Дружественные функции позволяют определять внутри какого-либо класса либо на внешнем программном уровне функцию получающую доступ к скрытым элементам различных объектов. По сути такая функция становится принадлежностью одновременно всех классов с которыми она работает, при этом не является членом классом к которому получает доступ. Если некоторая функция определена как дружественная для класса Х, то она:

* Не является член-функцией данного класса.
* Получает доступ ко всем компонентам класса.

Дружественная функция является глобальной функцией с возможностью доступа к локальным компонентам. Объявление функции со спецификатором friend в пределах класса Х позволяет ввести имя функции в границы класса, такое объявление может быть помещено в любой раздел класса. Friend-функции рассматриваются как часть интерфейса класса и позволяет реализовать взаимодействие класса с внешней средой. Член-функция одного класса может быть объявлена со спецификатором friend для другого класса. В данном случае фукнция function\_of\_x имеет доступ как к локальным компонентам класса Х так и к Y. В качестве дружественного может быть объявлен целый класс. Дружественными будут считаться все внутренние функции класса Y для класса Z.

Основные свойства и правила использования friend:

* Они не являются компонентами класса, но получают доступ ко всем его компонентам
* Если friend функции одного класса не являются компонентами другого класса, то они вызываются так же как и обычные глобальные функции
* Если friend функции одного класса не являются компонентами другого класса, то они не имеют указателя this
* Friend функции не наследуются в производных классах

Создание и уничтожение объектов

Если в классе определен конструктор, то этот конструктор вызывается каждый раз когда создается объект этого класса. Если в классе имеется деструктор, то он вызывается всегда, когда объект уничтожается. Каждый объект имеет время, которое принято называть временем жизни объекта. Время жизни определяется тем где и как создан объект. Объекты могут быть созданы в следующих формах:

* Глобальные объекты. Создаются в начале выполнения программы и разрушаются при ее завершении
* Автоматические объекты. Создаются когда их объявление встречается в программе. Разрушаются, когда блок программы, в котором они объявлены разрушается или уничтожается. (локальные)
* Статические объекты. Создаются один раз при запуске программы, к которой они относятся и разрушаются, при завершении выполнения программы.
* Объекты динамически выделяемые. Создаются принудительно путем вызова функции new и освобождаются путем вызова функции delete.
* Объекты являющиеся компонентами классов. Создаются при построении объектов класса в котором они описаны и разрушаются при разрушении объекта класса, в котором они описаны.

**Наследование**

Наследование – один из основных принципов ООП предполагающий организацию связи между абстрактными типами данных при которой появляется возможность на базе существующих типов создавать новые, при этом вновь создаваемый тип способен наследовать как данные так и функции того типа от которого он порождается, это позволяет создавать иерархию классов.

Новый класс может включать в свой состав новые член-данные, новые член-функции, а так же изменять член-функции наследуемые из классов родителей. По сути производные классов является эффективным средством для расширения функциональных возможностей существующих классов без их перепрограммирования и повторной компиляции.

Производным классом является новый создаваемый класс, класс на базе которого он создается – базовый класс, прародителем всех классов называют супер класс.

В общем случае возможны 2 типа наследования:

* Простое. Когда любой производный класс имеет единственного прямого, непосредственного родителя.
* Множественное наследование предполагает, что производный класс может быть порожден более чем от одного прямого базового родителя.

Основные правила использования базовых и производных классов:

Пусть функция F принадлежит базовому классу B. Тогда в производном классе П можно:

1. Полностью заменить функцию F.
2. Доопределить (частично изменить) функцию F. Т. Е. определить функцию с тем же именем, а в теле ее вызвать функцию F базового класса.
3. Использовать функцию F без изменения.

- Если объявить указатель на базовый класс, то ему можно присвоить значение указателя на объект производного класса. Это связано с тем, что структура производного класса является обобщением структур вышестоящих классов и компилятор непосредственно может получить адрес расположения в структуре каждого базового класса, не изменяя при этом неявного указателя this.

- Указателю на производный клас нельзя присвоить значение указателя на объект базового класса

- Регулирование достигается при помощи использования атрибутов компонентов базового класса:

* Если наследование используется с public, то область видимости не меняется
* Если с protected то все, что описано в области public трансформируется в protect
* Если Private, то все в private

Правило ромба

Наследование от виртуального класса позволяет изменить формальный вызов класса global.

Отличительной особенностью виртуального класса является следующее: он инициализируется от непосредственно порожденным классом, а более отдаленым порождением. Конструкторы и деструкторы выполняются в следующем порядка. Конструкторы базовых классов выполняются до виртуальных классов.

* Если класс имеет несколько виртуальных классов, то они вызываются со список порождений
* Все деструкторы виртуальных классов выполняются после не виртуальных

Функции на разных уровнях иерархии могут иметь одинаковые имена, при этом при вызове приоритетной будет являться функция принадлежащая более далекому от основания базовому классу. Определение того какая функция будет выбрана выполняется при компиляции и носит название связывание.

В ООП выделяют 2 типа связывания:

* Ранее. Осуществляется на этапе компиляции программы, когда в точку вызова той или иной функции компилятор, а затем и компоновщик записывает адрес точки входа конкретной вызываемой функции.
* Позднее. Предполагает определение контректной функции на этапе выполнения программы. Компилятор в точку вызова функции подлежащей позднему связыванию вставляет индекс этой функции из таблицы функции класса. Конкретная функция и ее адрес будет определена после выявления типа объекта, который осуществил вызов данной функции.

Таким образом на раннем и позднем связывнии основан принцип полиморфизма. Он предполагает возможность перегрузки имен операции и позволяет использовать одинаковые по именам функции для решения различных задач. По сути полиморфизм может быть определен как один интерфейс с разными реализациями. Среди ЯП различают полиморфные и мономорфные. Полиморфные поддерживают концепцию полиморфизма, а другие нет.

В зависимости от момента связывания функции с объектом. Полиморфизм делится на:

* Статический. Реализуется через механизм прямой перегрузки функции, т.е. в случае статическое полиморфизма член-функции должны иметь разные сигнатуры. Статический полиморфизм может реализован как в иерархии классов так и в пределах одного класса.
* Динамический. Предполагает использование виртуальных функции. Необходимость динамического полиморфизма возникает в том случае, когда поведения базового класса и класса наследника различаются.

Правила описания и использования виртуальных функции:

1. Виртуальная функция может быть только методом класса
2. Любую перегружаемую операцию-метод класса можно сделать виртуальной
3. Виртуальная функция, как и сама виртуальность, наследуется
4. Виртуальная функция может быть константной
5. Если в базовом классе впервые объявлена виртуальная функция, то функция должна быть либо чистой, либо для нее должно быть задано определение
6. Если в базовом классе определена виртуальная функция, то метод производного класса с такими же именем и прототипом автоматически является виртуальным
7. Конструкторы не могу быть виртуальными.
8. Статические методы не могут быть виртуальными.
9. Деструкторы могут (чаще – должны) быть.
10. Если некоторая функция вызывается с использованием ее полного имени, то виртуальность игнорируется

Вызов функции может оказаться не виртуальным:

* Когда метод вызывается не через указатель или ссылку
* Вызывается через указатель, но с уточнением имени класса
* Вызывается в конструкторе или деструкторе базового класса