Основные принципы объектно-ориентированного программирования

Прикладное программирование

Понятие ООП

- Объектно-ориентированное программирование — это стиль кодирования, который позволяет разработчику группировать схожие задачи в классы. Таким образом код соответствует принципу DRY (don't repeat yourself – не повторяй самого себя) и становится лёгким для сопровождения.

Понятие ООП

- Одним из преимуществ DRY программирования является то, что если некоторая информация требует изменения программы, то нужно изменять код лишь в одном месте, чтобы обновить алгоритм.
- ООП является очень чётким и чрезвычайно простым подходом к программированию.

Понятие класса и объекта

- В основе объектно-ориентированного языка программирования лежат два основных понятия: объект и класс.
- Класс – это способ описания сущности, определяющий состояние и поведение, зависящее от этого состояния, а также правила для взаимодействия с данной сущностью (контракт).
-

Понятие класса и объекта

- С точки зрения программирования класс можно рассматривать как набор данных (полей, атрибутов, членов класса) и функций для работы с ними (методов).
- С точки зрения структуры программы, класс является сложным типом данных.

Понятие класса и объекта

- Представьте себе, что вы проектируете автомобиль. Вы знаете, что автомобиль должен содержать двигатель, подвеску, две передних фары, 4 колеса, и т.д. Ещё вы знаете, что ваш автомобиль должен иметь возможность набирать и сбавлять скорость, совершать поворот и двигаться задним ходом.
-

Понятие класса и объекта

- И, что самое главное, вы точно знаете, как взаимодействует двигатель и колёса, согласно каким законам движется распредвал и коленвал, а также как устроены дифференциалы. Вы уверены в своих знаниях и начинаете проектирование.

Понятие класса и объекта

- Вы описываете все запчасти, из которых состоит ваш автомобиль, а также то, каким образом эти запчасти взаимодействуют между собой. Кроме того, вы описываете, что должен сделать пользователь, чтобы машина затормозила, или включился дальний свет фар. Результатом вашей работы будет некоторый эскиз. Вы только что разработали то, что в ООП называется класс.

Понятие класса и объекта

- **В нашем случае, класс будет отображать сущность – автомобиль. Атрибутами класса будут являться двигатель, подвеска, кузов, четыре колеса и т.д. Методами класса будет «открыть дверь», «нажать на педаль газа», а также «закачать порцию бензина из бензобака в двигатель». Первые два метода доступны для выполнения другим классам (в частности, классу «Водитель»). Последний описывает взаимодействия внутри класса и не доступен пользователю.**

Понятие класса и объекта

- Вы отлично потрудились и машины, разработанные по вашим чертежам, сходят с конвейера. Вот они, стоят ровными рядами на заводском дворе. Каждая из них точно повторяет ваши чертежи. Все системы взаимодействуют именно так, как вы спроектировали. Но каждая машина уникальна. Они все имеют номер кузова и двигателя, но все эти номера разные, автомобили различаются цветом, а некоторые даже имеют литьё вместо штампованных дисков. Эти автомобили, по сути, являются объектами вашего класса.

Понятие класса и объекта

- Объект (экземпляр) – это отдельный представитель класса, имеющий конкретное состояние и поведение, полностью определяемое классом.
- Объект имеет конкретные значения атрибутов и методы, работающие с этими значениями на основе правил, заданных в классе. В данном примере, если класс – это некоторый абстрактный автомобиль из «мира идей», то объект – это конкретный автомобиль, стоящий у вас под окнами.

Интерфейс

- Когда мы садимся за руль, мы начинаем взаимодействие с ним. Обычно, взаимодействие происходит с помощью некоторого набора элементов: руль, педали, рычаг коробки переключения передач в автомобиле. Всегда существует некоторый ограниченный набор элементов управления, с которыми мы можем взаимодействовать.
- Интерфейс – это набор методов класса, доступных для использования другими классами.

Интерфейс

- Очевидно, что интерфейсом класса будет являться набор всех его публичных методов в совокупности с набором публичных атрибутов. По сути, интерфейс специфицирует класс, чётко определяя все возможные действия над ним.
-

Интерфейс

- Хорошим примером интерфейса может служить приборная панель автомобиля, которая позволяет вызвать такие методы, как увеличение скорости, торможение, поворот, переключение передач, включение фар, и т.п. То есть все действия, которые может осуществить другой класс (в нашем случае – водитель) при взаимодействии с автомобилем.

Интерфейс

- При описании интерфейса класса очень важно соблюсти баланс между гибкостью и простотой. Класс с простым интерфейсом будет легко использовать, но будут существовать задачи, которые с помощью него решить будет не под силу.

Интерфейс

- В то же время, если интерфейс будет гибким, то, скорее всего, он будет состоять из достаточно сложных методов с большим количеством параметров, которые будут позволять делать очень многое, но использование его будет сопряжено с большими сложностями и риском совершить ошибку, что-то перепутав.

Основными характеристическими свойствами понятий классов и объектов являются:

- ✔Абстракция
- ✔Инкапсуляция;
- ✔Наследование;
- ✔Полиморфизм.

Абстракция

- Абстракция — это мощнейшее средство программирования. Именно то, что позволяет нам строить большие системы и поддерживать контроль над ними.
- Процесс создания уровней абстракции распространяется практически на все области знаний человека. Так, мы можем делать суждения о материалах, не вдаваясь в подробности их молекулярной структуры.

Абстракция

- Или говорить о предметах, не упоминая материалы, из которых они сделаны. Или рассуждать о сложных механизмах, таких как компьютер, турбина самолёта или человеческое тело, не вспоминая отдельных деталей этих сущностей.
- Во-вторых, абстракции в программировании были всегда, начиная с записей Ады Лавлейс, которую принято считать первым в истории программистом.

Инкапсуляция

- Объекты моделируют характеристики и поведение элементов мира, в котором мы живем. Они являются окончательной абстракцией данных.
- Объекты содержат вместе все свои характеристики и особенности поведения. Отношения частей к целому и взаимоотношения между частями становятся понятнее тогда, когда все содержится вместе в одной упаковке. Это и называется инкапсуляцией.

Инкапсуляция

- Инкапсуляция - комбинирование записей с процедурами и функциями, манипулирующими полями этих записей, формирует новый тип данных - объект (под записью понимается переменная типа "запись").

Наследование

- Не менее важным является и тот факт, что объекты могут наследовать характеристики и поведение того, что мы называем порождающие, родительские объекты (или предки). Здесь происходит качественный скачок: наследование, возможно, является сегодня единственным самым крупным различием между обычным программированием и объектно-ориентированным программированием.

Наследование

- Процесс, с помощью которого один тип наследует характеристики другого типа, называется наследованием. Наследник называется порожденным (дочерним) типом, а тип, которому наследует дочерний тип, называется порождающим (родительским) типом.

Наследование

- Наследование - 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.

Полиморфизм

- 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.

Структура класса

- Класс имеет имя, состоит из полей, называемых членами класса и функций - методов класса.
- Описание класса имеет следующий формат:
- class name // name – имя класса
- {
- private:

Структура класса

- // Описание закрытых членов и методов класса
- protected:
- // Описание защищенных членов и методов класса
- public:
- // Описание открытых членов и методов класса
- }

Открытые и закрытые члены класса

- В отличии от полей структуры доступных всегда, в классах могут быть члены и методы различного уровня доступа:
- открытые public (публичные), вызов открытых членов и методов
- класса осуществляется с помощью оператора . ("точка");

Открытые и закрытые члены класса

- закрытые private (приватные), доступ к которым возможен только с помощью открытых методов.
- защищенные методы (protected).
- После описания класса необходимо описать переменную типа class.

Открытые и закрытые члены класса

- Например: name_class name;
- здесь name_class – имя класса, name – имя переменной.
- В дальнейшем переменную типа class будем называть «объект» или «экземпляр класса». Объявление переменной типа class (в нашем примере переменная name типа name_class) называется созданием (инициализацией) объекта (экземпляра класса).

Открытые и закрытые члены класса

- После описания переменной можно обращаться к членам и методам класса.
- Обращение к членам и методам класса осуществляется аналогично обращению к полям структуры с помощью оператора «.» (точка).

Открытые и закрытые члены класса

- name.p1; //Обращение к полю p1 экземпляра класса name.
- name.f1(par1,par2,…parn); //Обращение к методу f1 экземпляра класса name,
- //par1, par2, …, parn – список формальных параметров функции f1.
- Члены класса доступны из любого метода класса и их не надо передавать в качестве параметров функций-методов.

Пример работы с классами

- Рассмотрим класс complex для работы с комплексными числами1.
- В классе complex будут члены класса:
- double x – действительная часть комплексного числа;
- double y – мнимая часть комплексного сила.

Пример работы с классами

- методы класса:
- double modul() – функция вычисления модуля комплексного числа;
- double argument() – функция вычисления аргумента комплексного числа;
- void show_complex() – функция выводит комплексное число на экран.

Пример работы с классами

- #include <iostream>
- #include <string>
- #include <math.h>
- #define PI 3.14159
- using namespace std;
- class complex //Определяем класс complex

Пример работы с классами

- {
- public:
- double x; //Действительная часть комплексного числа.
- double y; //Мнимая часть комплексного числа.

Пример работы с классами

- //Метод класса complex – функция modul, для вычисления модуля комплексного числа.
- double modul()
- {
- return pow(x*x+y*y,0.5);
- }

Пример работы с классами

- //Метод класса complex – функция argument,

для вычисления аргумента комплексного числа.

- double argument()
- {
- return atan2(y,x)*180/PI;
- }

Пример работы с классами

- //Метод класса complex – функция show_complex, для вывода комплексного числа.
- void show_complex()
- {
- if (y>=0)
- //Вывод комплексного числа с положительной мнимой частью.

Пример работы с классами

- cout<<x<<"+"<<y<<"i"<<endl;
- else
- //Вывод комплексного числа с отрицательной
- //мнимой частью.
- cout<<x<<y<<"i"<<endl;
- }
- };

Пример работы с классами

- int main()
- {
- //Определяем переменную chislo типа complex.
- complex chislo;
- //Определяем действительную часть комплексного числа.
- chislo.x=3.5;
- //Определяем мнимую часть комплексного числа.
- chislo.y=-1.432;

Пример работы с классами

- //Вывод комплексного числа, chislo.show_complex() – обращение к методу класса.
- chislo.show_complex();
- //Вывод модуля комплексного числа, chislo.modul() – обращение к методу класса.
- cout<<"Modul' chisla="<<chislo.modul();

Пример работы с классами

- //Вывод аргумента комплексного числа,
- //chislo.argument() – обращение к методу класса.
- cout<<endl<<"Argument chisla="<<chislo.argument()<<endl;
- return 1;
- }

Пример работы с классами

- **Результат работы программы:**
- **3.5-1.432i**
- **Modul chisla=3.78162**
- **Argument chisla=-22.2516**

Пример работы с классами

- **Использование открытых членов и методов позволяет получить полный доступ к элементам класса, однако это не всегда хорошо. Если все члены класса объявить открытыми, то при непосредственном обращении к ним появится потенциальная возможность внести ошибку в функционирование взаимосвязанных между собой методов класса. Поэтому, общим принципом является следующее: «Чем меньше открытых данных о классе используется в программе, тем лучше».**

Пример работы с классами

- Если в описании элементов класса отсутствует указание метода доступа, то члены и методы считаются закрытыми (private). Принято описывать методы за пределами класса.
- Изменим рассмотренный ранее пример класса complex.

Пример работы с классами

- Добавим метод vvod, предназначенный для ввода действительной и мнимой части числа, члены класса и метод show_complex сделаем закрытыми, а остальные методы открытыми. Текст программы будет иметь вид:
- #include <iostream>
- #include <string>
- #include <math.h>
- #define PI 3.14159

Пример работы с классами

- using namespace std;
- class complex {
- //Открытые методы.
- public:
- void vvod();
- double modul();
- double argument();

Пример работы с классами

- //Закрытые члены и методы.
- private:
- double x;
- double y;
- void show_complex();
- };

Пример работы с классами

- //Описание открытого метода vvod класса complex.
- void complex::vvod()
- {
- cout<<"Vvedite x\t";
- cin>>x;
- cout<<"Vvedite y\t";
- cin>>y;

Пример работы с классами

- // Вызов закрытого метода show_complex из открытого метода vvod.
- show_complex();
- }
- //Описание открытого метода modul класса complex.
- double complex::modul()
- {
- return pow(x*x+y*y,0.5);
- }

Пример работы с классами

- //Описание открытого метода argument класса complex.
- double complex::argument()
- {
- return atan2(y,x)*180/PI;
- }

Пример работы с классами

- //Описание закрытого метода modul класса complex.
- void complex::show_complex()
- {
- if (y>=0)
- cout<<x<<"+"<<y<<"i"<<endl;
- else cout<<x<<y<<"i"<<endl;
- }

Пример работы с классами

```
int main()
{
complex chislo;
chislo.znach();
cout<<"Modul kompleksnogo chisla="<<chislo.modul();
cout<<endl<<"Argument kompleksnogo
chisla="<<chislo.argument()<<endl;
return 1;
}
```

Пример работы с классами

- Результат работы программы:

```
Vvedite x 3
Vvedite y -1
3-1i
Modul kompleksnogo chisla=3.16228
Argument kompleksnogo chisla=-18.435
```