Объектно-ориентированное программирование (ООП)

- это особый концептуальный подход к проектированию программ, и C++ расширяет язык C средствами, облегчающими применение такого подхода.

Наиболее важные характеристики ООП:

- абстракция;
- инкапсуляция и сокрытие данных;
- полиморфизм;
- наследование;
- повторное использование кода

Класс

— это единственное наиболее важное расширение C++, предназначенное для реализации этих средств и связывающее их между собой.

Объектно-ориентированное программирование (ООП)

При объектно-ориентированном подходе вы концентрируетесь на объекте, как его представляет пользователь, думая о данных, которые нужны для описания объекта, и операциях, описывающих взаимодействие пользователя с данными.

Объектно-ориентированное программирование (ООП)

После разработки описания интерфейса вы перейдете к выработке решений о том,
как реализовать этот интерфейс и как организовать хранение данных.

Объектно-ориентированное программирование (ООП)

И, наконец, вы соберете все это вместе в программу, соответствующую новому проекту.

Абстракции и классы

В компьютерных вычислениях абстракция — это ключевой шаг в представлении информации в терминах ее интерфейса с пользователем.

То есть вы абстрагируете основные операционные характеристики проблемы и выражаете решение в этих терминах.

Абстракции и классы

От абстракций легко перейти к определяемым пользователем типам, которые в C++ представлены классами, реализующими абстрактный интерфейс.

Что такое тип? Спецификация базового типа выполняет три вещи:

- **Определяет, сколько памяти нужно объекту.**
- **Определяет, как интерпретируются биты памяти. (*Типы long и float могут занимать одинаковое количество бит памяти, но транслируются в числовые значения по-разному.*)**
- **Определяет, какие операции, или методы, могут быть применены с использованием этого объекта данных.**

Что такое тип?

Для встроенных типов информация об операциях встроена в компилятор.

Но когда вы определяете пользовательский тип в C++, то должны предоставить эту информацию самостоятельно.

В качестве вознаграждения за эту дополнительную работу вы получаете мощь и гибкость новых типов данных, соответствующих требованиям реального мира.

Классы в C++

Класс **— это двигатель C++, предназначенный для трансляции абстракции в пользовательские типы.**

Он комбинирует представление данных и методов для манипулирования этими данными в пределах одного аккуратного пакета.

Обычно спецификация класса состоит из двух частей

- ***Объявление класса***, описывающее компоненты данных в терминах членов данных, а также открытый интерфейс в терминах функций-членов, называемых ***методами.***

- ***Определения методов класса***, которые описывают, как реализованы определенные функции-члены.

Классы в C++

Грубо говоря, объявление класса предоставляет общий обзор класса, в то время как определения методов снабжают необходимыми деталями.

Что такое интерфейс?

Интерфейс — это совместно используемая часть, предназначенная для взаимодействия двух систем, например, между компьютером и принтером или между пользователем и компьютерной программой.

Что такое интерфейс?

В отношении классов мы говорим об открытом интерфейсе.

В этом случае потребителем его является программа, использующая класс, система взаимодействия состоит из объектов класса, а интерфейс состоит из методов, предоставленных тем, кто написал этот класс.

Что такое интерфейс?

**Интерфейс позволяет вам, как программисту,
написать код, взаимодействующий с объектами класса, и таким образом, дает программе
возможность взаимодействовать с объектами класса.**

Интерфейс

Обычно программисты на C++ помещают интерфейс, имеющий форму определения класса, в заголовочный файл,

а реализацию в форме кода для методов класса — в файл исходного кода.

ПРЕДСТАВЛЕНИЕ ОБ ОБЪЕКТАХ

- В известном смысле объект представляет собой сущность.
- В C++ вы используете класс для определения своих объектов.
- Ваша цель состоит в том, чтобы включить в класс столько информации об объекте, сколько требуется.

Класс

- Класс позволяет вашим программам группировать данные и функции, которые выполняют операции над этими данными.
- Функции класса называются методами.

Класс

Класс C++ должен иметь уникальное имя, за которым следует открывающая фигурная скобка, один или несколько элементов и закрывающая фигурная скобка:

```
class class_name
    {
        int data_member; // Элемент данных
        void show_member(int); // Функция-элемент
    };
```

Класс

- После определения класса вы можете объявлять переменные типа этого класса (называемые объектами), как показано ниже:

  class_name object_one, object_two, object_three;

Следующее определение создает класс *employee*, который содержит определения данных и метода:

```
class employee {
 public:
  char name[64];
  long employee_id;
  float salary;
  void show_employee(void)
   {
     cout << "Имя: " << name << endl;
     cout << "Номер служащего: " << employee_id << endl;
     cout << "Оклад: " << salary << endl;
   };
};
```

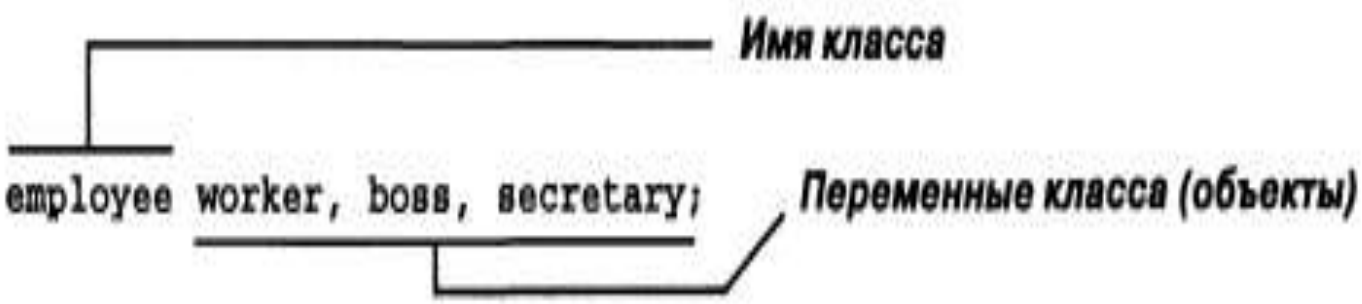
В данном случае класс содержит три переменные и одну функцию-элемент.

Класс

Обратите внимание на использование метки *public* внутри определения класса. Как вы узнаете позже, элементы класса могут быть *частными* (*private*) или *общими* (*public*), от чего зависит, как ваши программы обращаются к элементам класса. В данном случае все элементы являются общими, это означает, что программа может обращаться к любому элементу, используя оператор точку.

Класс

- После определения класса внутри вашей программы вы можете объявить объекты (переменные) типа этого класса, как показано ниже:



Создание простого класса

Следующая программа создает два объекта *employee*.

Используя оператор точку, программа присваивает значения элементам данных. Затем программа использует элемент *show_employee* для вывода информации о служащем:

Создание простого класса

```
class employee {
 public: char name[64];
 long employee_id;
 float salary;
 void show_employee(void)
  {
    cout << "Имя: " << name << endl;
    cout << "Номер служащего: " << employee_id << endl;
    cout << "Оклад: " << salary << endl;
  };
};
```

Создание простого класса

```
void main(void)
 {
    employee worker, boss;

    strcpy(worker.name, "John Doe");
    worker.employee_id = 12345;
    worker.salary = 25000;

    strcpy(boss.name, "Happy Jamsa");
    boss.employee_id = 101;
    boss.salary = 101101.00;
    worker.show_employee();
    boss.show_employee();
 }
```

Создание простого класса

Как видите, программа объявляет два объекта типа *employee* — *worker* и *boss*, а затем использует оператор точку для присваивания значений элементам и вызова функции *show_employee*.

ОПРЕДЕЛЕНИЕ МЕТОДОВ КЛАССА ВНЕ КЛАССА

В предыдущем классе *employee* функция была определена внутри самого класса (*встроенная (inline) функция*).
При увеличении функций определение встроенных функций внутри класса может внести беспорядок в описание класса.

ОПРЕДЕЛЕНИЕ МЕТОДОВ КЛАССА ВНЕ КЛАССА

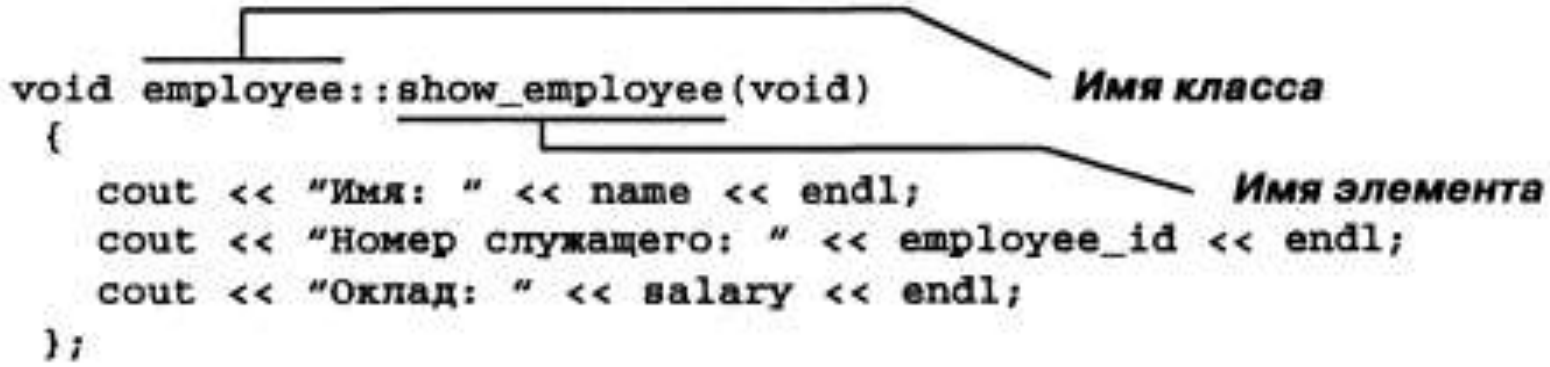
В качестве альтернативы вы можете поместить прототип функции внутри класса, а затем определить функцию вне класса:

```
class employee {
 public:
  char name[64];
  long employee_id;
  float salary;
  void show_employee(void);
};
```

Прототип функции

ОПРЕДЕЛЕНИЕ МЕТОДОВ КЛАССА ВНЕ КЛАССА

Так как разные классы могут использовать функции с одинаковыми именами, вы должны предварять имена определяемых вне класса функций именем класса и оператором глобального разрешения (::). В данном случае определение функции становится следующим:



ОПРЕДЕЛЕНИЕ МЕТОДОВ КЛАССА ВНЕ КЛАССА

Любая функция с определением внутри объявления класса автоматически становится встроенной.
Если хотите, можете определить функцию-член вне объявления класса и, тем не менее, сделать ее встроенной. Чтобы это сделать, просто используйте квалификатор ***inline*** при определении функции в разделе реализации класса:

```
class Stock
{
private:
    ...
    void set_tot();             // определение оставлено отдельным
public:
    ...
};
inline void Stock::set_tot()  // использование inline в определении
{
    total_val = shares * share_val;
}
```

Частные и общие данные

Сокрытие информации **представляет собой процесс, в результате которого программе предоставляется только минимальная информация, необходимая для использования класса.**

Частные и общие **элементы класса помогают вам получить информацию, скрытую внутри вашей программы.**

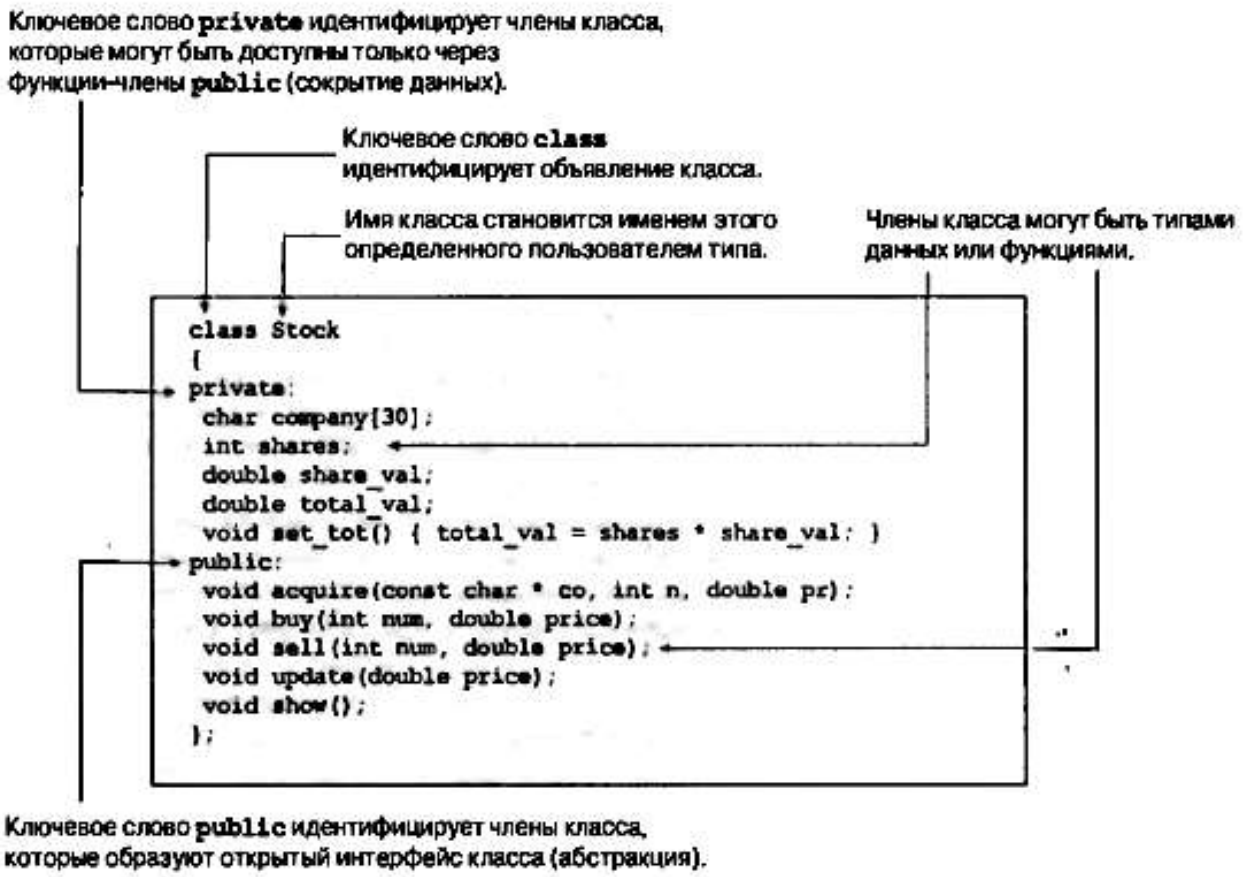
Частные и общие данные

Ключевые слова *private и public*.

Эти метки позволяют *управлять доступом* к членам класса. Любая программа, которая использует объект определенного класса, может иметь непосредственный доступ к членам из раздела *public*.

Доступ к членам объекта из раздела *private* программа может получить *только* через открытые функции-члены из раздела *public* (или же, как будет показано позже, через *дружественные функции*).

Частные и общие данные



Частные и общие данные

Таким образом, открытые функции-члены действуют в качестве посредников между программой и закрытыми членами объекта; они предоставляют интерфейс между объектом и программой.
Эта изоляция данных от прямого доступа со стороны программы называется *сокрытием данных.*

Частные и общие данные

Проектное решение класса пытается отделить открытый интерфейс от специфики реализации. Открытый интерфейс представляет абстрактный компонент проектного решения.
Собрание деталей реализации в одном месте и отделение их от абстракции называется ***инкапсуляцией.***

Частные и общие данные

Сокрытие данных (помещение данных в раздел ***private*** класса) является примером инкапсуляции.

Другим примером инкапсуляции может служить обычная практика помещения определений функций класса в файл, отдельный от объявления класса.

Частные и общие данные

Сокрытие данных не только предотвращает прямой доступ
к данным, но также избавляет вас (в роли пользователя этого класса) от необходимости знать то, как представлены данные.

Классы и структуры

Описания классов выглядят очень похожими на объявления структур с дополнениями в виде функций-членов и меток видимости ***private*** *и* ***public.*** Фактически C++ расширяет на структуры те же самые свойства, которые есть у классов. Единственная разница состоит в том, что типом доступа по умолчанию у структур является ***public*****,** в то время как у классов — ***private*****.**

Программисты на C++ обычно используют классы для реализации описаний классов, тогда как ограниченные структуры применяются для чистых объектов данных (которые часто называются *простыми старыми структурами данных* (plain-old data — POD)).

Клиент-серверная модель

Программисты, соблюдающие принципы ООП, часто обсуждают проект программ в терминах ***клиент-серверной модели***.

Клиент-серверная модель

Согласно этой концепции, ***клиентом*** является программа, которая использует класс. Объявление класса, включая его методы, образует ***сервер***, который является ресурсом, доступным нуждающейся в нем программе.
Клиент взаимодействует с сервером только через открытый (*public*) интерфейс. Это означает, что единственной ответственностью клиента и, как следствие — программиста, является знание интерфейса. Ответственностью сервера и, как следствие — его разработчика, является обеспечение того, чтобы его реализация надежно и точно соответствовала интерфейсу.

Клиент-серверная модель

Любые изменения, вносимые разработчиком сервера в класс, должны касаться деталей реализации, но не интерфейса.
Это позволяет программистам разрабатывать клиент и сервер независимо друг от друга, без внесения в сервер таких изменений, которые нежелательным образом отобразятся на поведении клиента.