C++ Модули 3-4

Тарабонда Герман C++

Содержание

1 Пространство имен (namespace)

Пишем большую программу, у нас есть 2 класса User, но хотим использовать это слова и в бд и на сервере. Можно сделать так

```
class Database{
   class User{ ... };
};
class Application{
   class User{ ... };
};
```

Есть проблема. Первая: надо всегда писать слово `static` или постоянно проверять, что объект типа `Database` ровно один. Вторая: если файлов много, то надо писать 10 разных классов или 1 хедер на всех и тогда каждый раз нужно будет перекомпилировать этот большой хедер.

Придумали namespace.

Пример обращения:

```
namespace database{
class User{ ... };
} // namespace database
namespace application{
class User{ ... };
} // namespace application

// чтобы обратиться нужно
application::User;
```

Можно использовать так:

```
int main(){
   using application::User; // делаем User из application до }
}
```

```
void connectTo(){
   using namespace database; // в среднем не очень
}

std::vector // так все же лучше
```

Обычно using пишут в сппишниках. В хедерах ставим бан этим штукам.

Бывают вложенные namespace

```
namespace database{
   void connect(){internal::conectEx( ... );};
   namespace internal{
      void connectEx();
   } // namespace internal
} // namespace database
database::internal::connectEx( ... );

namespace database::internal {
}
namespace database {
} // можно переоткрывать ыnamespac
// это все работает как допиши" перед названием database::"
```

Еще бывает глобальный namespace.

```
int x;
::x = 10; // глобальный namespace
static int y; // влияет только на линковку
```

Еще существуют анонимные namespace

```
namespace {
   void foo();
} // анонимный namespace == все что внутри становится static
foo(); // можно писать так, а не A::foo() и foo() не виден в других единицах
    трансляции.
// В C++ лучше писать всегда анонимный namespace
```

---

2 Правила поиска имен

Unqualified lookup – когда пишем без двоеточий: `foo();`

```
// (4) потом ищется в глобальном namespace до строчки с foo()
namespace application{ // (3) потом здесь до строчки с foo()
class User{ // (2) затем везде в классе ищем foo()
   void x() { foo(); }; // (1) сначала ищется ссылка на foo в x до строчки с foo()
};
```

```
}
```

Qualified lookup – когда пишем с двоеточиями: `::foo();`

Тут ясно где искать нужную функцию/ переменные.

Например:

```
database::internal::foo();
// Порядок вызовов
// (1) qualified lookup foo() в internal
// (2) qualified lookup internal в database
// (3) unqualified lookup database
```

А теперь стреляем себе в ногу

```
#include <vector>
namespace foo {
   std::vector<int> v;
} // OK

#include <vector>
namespace foo {
   namespace std{
   }
   std::vector<int> v; // не надо пересекаться с глобальным
} // не OK
```

```
namespace mysql {
   void connect();
}

namespace application::database::mysql {
}
namespace application::database{
   void connect(){ mysql::connect(); } // не OK, так как будет искать в
    application::database
   // void connect(){ ::mysql::connect(); }  OK
}
```