Лекция 4. Объекты и классы

Казахский национальный университет имени аль-Фараби

Кафедра компьютерных наук

Объектно-ориентированное программирование

Лектор: старший преподаватель, PhD, Кенжебек Ержан Ғалымжанұлы

Цель лекции

Цель лекции

Изучить понятия класса и объекта, сокрытие данных, методы класса, а также научиться определять классы и использовать их в программах.

Ожидаемые результаты

1. Определять класс и создавать его объекты
2. Применять спецификаторы доступа private и public
3. Писать методы, геттеры и сеттеры
4. Использовать оператор :: и возвращать объекты из функций

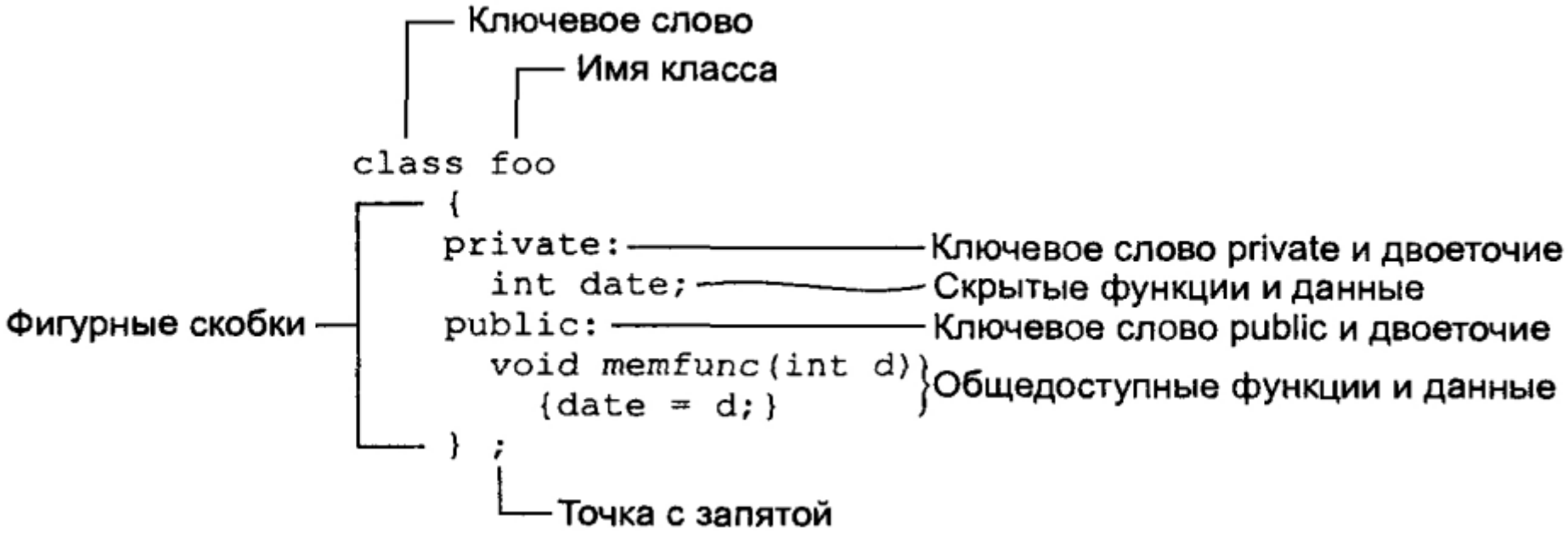
План лекции

- Введение в классы и создание простого класса
- Сокрытие данных: private, public
- Данные и методы класса
- Геттер и сеттер
- Оператор разрешения области видимости
- Создание библиотеки и их запуск

Введение

- Теперь мы приступим к изучению того раздела программирования, к которому так долго готовились: **объекты** и **классы**. Мы предварительно рассмотрели все, что нам будет необходимо: структуры, позволяющие группировать данные, и функции, объединяющие фрагменты программы под одним именем. В этой главе мы соединим эти два понятия и создадим новый элемент программы — класс. Мы начнем с создания самых простых классов, постепенно усложняя наши примеры.

Синтаксис определения класса



Простой класс

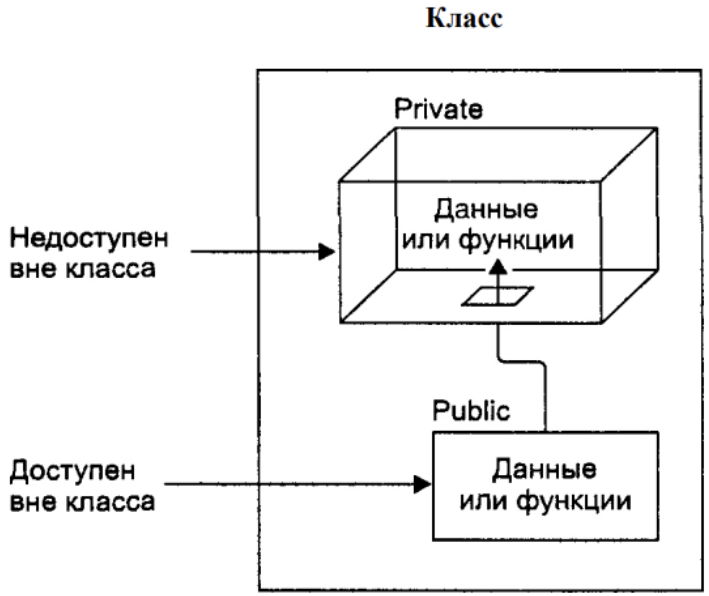
```
#include <iostream>
using namespace std;

class simpleobj // определение класса
{
private:
 int somedata; // поле класса
public:
 void setdata(int d) // метод класса, изменяющий значение поля
 { somedata = d; }
 void showdata() // метод класса, отображающий значение поля
 { cout << "Значение поля равно " << somedata << endl; }
};

int main() {
 simpleobj s1, s2; // определение двух объектов(экземпляры) класса simpleobj
 s1.setdata(1066); // вызовы метода setdata()
 s2.setdata(1776);
 s1.showdata(); // вызовы метода showdata()
 s2.showdata();
 return 0;
}
```



Данные + функции (методы) = Класс

private && public

- Ключевой особенностью объектно-ориентированного программирования является возможность сокрытия данных. Этот термин понимается в том смысле, что данные заключены внутри класса и **защищены от несанкционированного доступа функций, расположенных вне класса**.
- Если необходимо защитить какие-либо данные, то их помещают внутрь класса с ключевым словом **private**. Такие данные доступны только внутри класса. Данные, описанные с ключевым словом **public**, напротив, доступны за пределами класса.



Зачем скрывать данные?

- Не путайте сокрытие данных с техническими средствами, предназначенными для защиты баз данных. В последнем случае для обеспечения сохранности данных можно, например, попросить пользователя ввести пароль перед тем, как разрешить ему доступ к базе данных. Пароль обеспечивает защиту базы данных от несанкционированного или злоумышленного изменения, а также копирования и чтения ее содержимого.
- Сокрытие данных в нашем толковании означает ограждение данных от тех частей программы, которые не имеют необходимости использовать эти данные. **В более узком смысле это означает сокрытие данных одного класса от другого класса.** Сокрытие данных позволяет уберечь опытных программистов от своих собственных ошибок. Программисты могут сами создать средства доступа к закрытым данным, что значительно снижает вероятность случайного или некорректного доступа к ни

Данные и методы класса

- Данные, содержащиеся внутри класса, называют данными-членами или полями класса. Число полей класса, как и у структуры, теоретически может быть любым.
- Методы класса — это функции, входящие в состав класса.

- **Сокрытие данных и доступность функции:**
  - Как правило, скрывая данные класса, его методы оставляют доступными. Это объясняется тем, что данные скрывают с целью избежать нежелательного внешнего воздействия на них, а функции, работающие с этими данными, должны обеспечивать взаимодействие между данными и внешней по отношению к классу частью программы.
  - Тем не менее, не существует четкого правила, которое бы определяло, какие данные следует определять как private, а какие функции — как public. Вы можете столкнуться с ситуациями, когда вам будет необходимо скрывать функции и обеспечивать свободный доступ к данным класса.

Геттер и сеттер

```cpp
#include<iostream>
struct Distance
{
private:
    int iFeet;
    float fInches;
public:
    void setFeet(int x)
    {
        iFeet = x; //LEGAL: private member accessed by
        //member function
    }
    int getFeet()
    { return iFeet; }

    void setInches(float y)
    { fInches = y; }
    float getInches()
    { return fInches; }
};

int main()
{
    Distance d1, d2;
    d1.setFeet(2);
    d1.setInches(2.2);
    d2.setFeet(3);
    d2.setInches(3.3);
    //d1.iFeet++; //ERROR!!: private member accessed by
    //non-member function
    std::cout << d1.getFeet() << " " << d1.getInches() << std::endl;
    std::cout << d2.getFeet() << " " << d2.getInches() << std::endl;
    return 0;
}
```

```cpp
#include<iostream>
struct Distance
{
private:
int iFeet;
float fInches;
public:
void setFeet(int x)
{
iFeet = x; //LEGAL: private member accessed by
//member function
}

int getFeet()
{ return iFeet; }

void setInches(float y)
{ fInches = y; }

float getInches()
{ return fInches; }
};

int main(){
Distance d1, d2;
d1.setFeet(2);
d1.setInches(2.2);
d2.setFeet(3);
d2.setInches(3.3);
d1.iFeet++; //ERROR!!: private member accessed by
//non-member function
std::cout << d1.getFeet() << " " << d1.getInches() << std::endl;
std::cout << d2.getFeet() << " " << d2.getInches() << std::endl;
return 0;
}
```

Оператор разрешения области видимости (::)

```
class Distance
{
   int iFeet;
   float fInches;
 public:
   void setFeet(int); //prototype only
   int getFeet(); //prototype only
   void setInches(float); //prototype only
   float getInches(); //prototype only
};
```

lhs operator rhs

```
void Distance::setFeet(int x) //definition
{
iFeet = x;
}

int Distance::getFeet() //definition
{
return iFeet;
}

void Distance::setInches(float y) //definition
{
fInches = y;
}

float Distance::getInches() //definition
{
return fInches;
}
```

Создание библиотеки используя оператор ::

- Для программиста который создает библиотеки в C++, создание классов можно разделить на 3 шага:
  1. Расположить определения класса в заголочном файле (**header file**)
  2. Расположить определении функции членов класса в C++ исходном файле (**library source code**), то есть здесь запускается реализация класса Distance например.
  3. Обеспечить header файл и библиотеку для тех программистов которые хотят использовать новый тип данных

- **Пример в Visual Studio**

Использование классов в программах

- Шаги программиста для использования нового типа данных:
  1. Добавить header файл который обеспечил программист отвечающий за библиотеки
  2. Создать объекты нового типа данных
  3. Вызвать связанные функции этих объектов в их исходный код

Пример в Visual Studio

Возвращения объектов класса

- **Пример в Visual Studio**

Возвращения объектов с помощью ссылки

- **Пример в Visual Studio**

Выводы

1. Класс объединяет данные (поля) и функции (методы) и является шаблоном для создания объектов.
2. Сокрытие данных (private) защищает поля от неконтролируемого доступа; интерфейс класса образуют открытые (public) методы.
3. Геттеры и сеттеры обеспечивают контролируемый доступ к закрытым полям.
4. Оператор :: позволяет определять методы вне класса и выносить реализацию в отдельную библиотеку.
5. Объекты можно передавать в функции и возвращать из них, в том числе по ссылке.

Контрольные вопросы

1. Что такое класс и объект? Чем они различаются?
2. Каков синтаксис определения класса?
3. Для чего нужно сокрытие данных? Чем private отличается от public?
4. Что такое геттер и сеттер? Зачем они нужны?
5. Для чего используется оператор разрешения области видимости ::?
6. Как вернуть объект класса из функции? Чем отличается возврат по значению от возврата по ссылке?

Список литературы

Основная литература

1. Лафоре Р. Объектно-ориентированное программирование в C++. 4-е изд. — СПб.: Питер, 2004.
2. Страуструп Б. Язык программирования C++. Специальное издание. — М.: Бином, 2011.
3. Липпман С., Лажойе Ж., Му Б. Язык программирования C++. Базовый курс. 5-е изд. — М.: Вильямс, 2014.

Дополнительная литература

4. Шилдт Г. Полный справочник по C++. 4-е изд. — М.: Вильямс, 2006.
5. Страуструп Б. Программирование: принципы и практика с использованием C++. 2-е изд. — М.: Вильямс, 2016.
6. Майерс С. Эффективный и современный C++. — М.: Вильямс, 2016.

Электронные ресурсы

7. Учебник по C++ (learncpp): www.learncpp.com
8. Справочник по C++ и стандартной библиотеке: en.cppreference.com

- Спасибо за внимание!