Объектно-ориентированное программирование

Преподаватель:
Готовская Светлана Борисовна

ООП — это метод программирования, который основан на представлении программы в виде совокупности объектов, каждый из которых является реализацией определенного класса, а классы образуют иерархию на принципах наследуемости.

Объекты и классы (object и class)

являются базовыми блоками объектно-ориентированной программы.

Класс (объект)— описание (абстракция), которое показывает, как построить существующую во времени и пространстве *переменную*

Класс (объект)— описание (абстракция), которое показывает, как построить существующую во времени и пространстве переменную

Примеры из окружающего мира:

объект характеризуется как совокупностью всех своих свойств (например, для животных – это наличие головы, ушей, глаз и т.д.) и их текущих значений (голова – большая, уши – длинные, глаза – желтые и т.д.), так и совокупностью допустимых для этого объекта действий (умение принимать пищу, сидеть, стоять, бежать и т.д.).

Примеры из программирования:

это …

Базовые принципы ООП

1) **ИНКАПСУЛЯЦИЯ**
2) **НАСЛЕДОВАНИЕ**
3) **ПОЛИМОРФИЗМ**
4) **передача сообщений**

Инкапсуляция

предполагает **объединение** в одном объекте **полей и методов**, которые манипулируют этими полями.

Примеры из окружающего мира:
Поля: голова, уши, глаза.
Методы: думать, слушать, смотреть.

Примеры из программирования:
Поля: переменные.
Методы: функции.

Наследование

метод, который позволяет классы использовать для создания новых классов.

Примеры из окружающего мира:

Поля: глаза как у мамы.

Методы: говорит как папа.

Примеры из программирования:

???

Наследование

метод, который позволяет классы использовать для создания новых классов.

Класс – предок, родитель: класс

Новый класс – дочерний: новый класс: класс

В описание нового класса может входить поле уже существующего класса. Метод создания новых классов называется композицией.

Новый класс (дочерний) - кроме собственных полей и методов может использовать поля и методы базового класса (родителя, предка).

Производный класс, в свою очередь, тоже может выступить в роли базового класса для другого производного класса и так далее - таким образом могут возникать целые иерархии классов.

Наследование

метод, который позволяет классы использовать для создания новых классов.

Класс – предок, родитель: класс

Новый класс – дочерний: новый класс: класс

Наследование позволяет создавать новые классы, изменяя или дополняя свойства прежних.

Класс-наследник получает все поля и методы предка, но может добавить собственные поля, добавить собственные методы или перекрыть своими методами одноименные унаследованные методы.

Полиморфизм

метод, который позволяет использовать одни и те же методы для решения разных задач.

Полиморфизм – это свойство родственных объектов (т.е. объектов, имеющих одного общего родителя) решать схожие по смыслу проблемы (методы) разными способами.

Примеры из окружающего мира:

Поля: глаза как у мамы, но более зеленые.

Методы: говорит с другой интонацией.

Примеры из программирования:

???

Наследование

метод, который позволяет классы использовать для создания новых классов.

Полиморфизм

метод, который позволяет использовать одни и те же методы для решения разных задач.

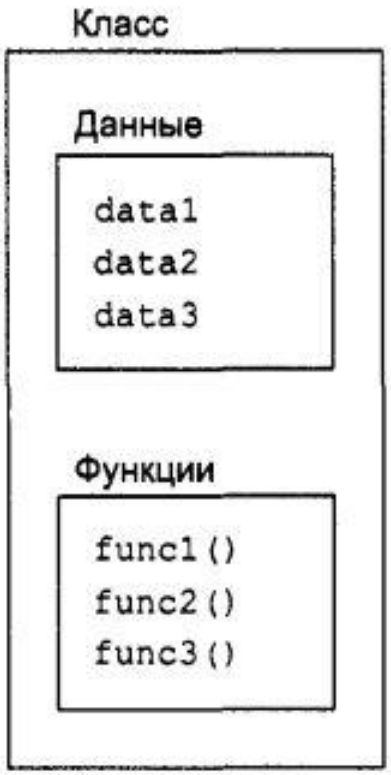
Класс – предок, родитель: класс
Новый класс – дочерний: новый класс: класс

Последовательное проведение в жизнь принципа «наследуй и изменяй» хорошо согласуется с поэтапным подходом к разработке крупных программных проектов.

Описание объектного типа

Класс (объект) – это структура данных, которая содержит поля и методы.

Компонент класса – поле, метод.
Поле: имя и тип данных.
Метод: функция, объявленная внутри класса и специально созданная для работы с данными полями.




```
class ИмяКласса
{
   // тело класса
};
```

Объявление класса

```
class /*имя класса*/
{
  private:
  /* список свойств и методов для использования внутри класса */
  public:
  /* список методов доступных другим функциям и объектам программы */
  protected:
  /*список средств, доступных при наследовании*/
};
```

Доступ для	Спецификаторы доступа		
	private	protected	public
Членов собственного класса	**открыт**	**открыт**	**открыт**
Членов производных (дочерних) классов	**закрыт**	**открыт**	**открыт**
Не членов класс	**закрыт**	**закрыт**	**открыт**

В C++ принято защищать некоторые данные класса от внешнего вмешательства. То есть, чтобы из главной функции, например, никто не мог напрямую обратиться к этим данным через объект и внести какие-либо изменения.

Чтобы сделать данные “закрытыми” надо разместить их в поле **private**.

ЗАДАЧА: создать класс, включающий поля (имя, возраст) и метод (вывод имени и возраста).

```
#include <iostream>
#include <windows.h>
#include <string>
#include <stdio.h>
using namespace std;
     class persona
     {
     public:
          string name;
          int age;
          void say()
          {
          cout << "Мое имя " << name << endl;
          cout << "Мой возраст " << age << endl;
          }
     };
int main()
{
SetConsoleCP(1251);
SetConsoleOutputCP(1251);

     persona x;
     cin >> x.name >> x.age;
     x.say();

cout << endl;
system("pause");
return 0;
}
```

ЗАДАЧА: создать класс, включающий поля (имя, возраст) и **ДВА** метода (вывод имени и вывод возраста).

```
class persona
{
public:
    string name;
    int age;
    void sayN()
    {
    cout << "Мое имя " << name << endl;
    }
    void sayA()
    {
    cout << "Мой возраст " << age << endl;
    }
};
int main()
{
SetConsoleCP(1251);
SetConsoleOutputCP(1251);

    persona x;
    cin >> x.name >> x.age;
    x.sayN();
    x.sayA();

cout << endl;
system("pause");
return 0;
}
```

ЗАДАЧА: создать класс, включающий поля (имя, возраст) и ДВА метода (вывод имени и вывод возраста). ИСПОЛЬЗОВАТЬ **private**

Если элементы данных объявлены в **private**,
то доступ к ним могут получить только методы класса,
внешний доступ к элементам данных запрещён.

Поэтому принято объявлять в классах **специальные методы** — так называемые **set** и **get** функции (сеттеры и геттеры), с помощью которых можно манипулировать элементами данных.

Сеттеры инициализируют элементы данных.
Геттеры позволяют просмотреть значения элементов данных.

ЗАДАЧА: создать класс, включающий поля (имя, возраст) и ДВА метода (вывод имени и вывод возраста). ИСПОЛЬЗОВАТЬ **private**

Обратите внимание на ошибку!!!

```
class persona
{
private:
    string name;
    int age;
public:
    void sayN()
    {
    cout << "Мое имя " << name << endl;
    }
    void sayA()
    {
    cout << "Мой возраст " << age << endl;
    }
};
int main()
{

    persona x;
    cin >> x.name >> x.age;      //ОШИБКА
    x.sayN();
    x.sayA();

}
```

```
class persona
{
private:
    string name;
    int age;
public:
    void setName(string Name)
    {
        name = Name;
    }
    void setAge(int Age)
    {
        age = Age;
    }
    string getName()
    {
        return name;
    }
    int getAge()
    {
        return age;
    }
    void sayN()
    {
        cout << "Мое имя " << name << endl;
    }
    void sayA()
    {
        cout << "Мой возраст " << age << endl;
    }
};
```

```
string N;
int A;
cin >> N >> A;
persona x;

x.setName(N);
x.setAge(A);
cout << "Мое имя " << x.getName() << endl;
cout << "Мой возраст " << x.getAge() << endl;

x.sayN();
x.sayA();
```

```
void setName(string Name)
{
    name = Name;
}
```

```
void setName(string name)
{
    this->name = name;
}
```

Указатель **this** указывает на текущий объект данного класса. Соответственно через **this** мы можем обращаться внутри класса к любым его членам.

Указатель **this** хранит адрес определённого объекта класса (в нашем примере он хранит адрес объекта **x**). Таким образом он неявно указывает методам класса с данными какого объекта надо работать.

```
class persona
{
private:
     string name;
     int age;
public:
     void setName(string name)
     {
     this->name = name;
     }
     void setAge(int age)
     {
      this->age = age;
     }
     string getName()
     {
          return name;
     }
     int getAge()
     {
          return age;
     }
     void sayN()
     {
          cout << "Мое имя " << name << endl;
     }
     void sayA()
     {
          cout << "Мой возраст " << age << endl;
     }
};
```

Недостаток!

В примере объявлено 2 объекта: **name, age**.

А если объявить 50 объектов класса,

Придется 50 раза вызывать метод,

который присваивает значения полям класса.

Для инициализации полей класса, а так же для выделения динамической памяти, используется **конструктор**.

Конструктор (от construct – создавать) – это особый метод класса, специальная функция, которая выполняет начальную инициализацию элементов данных.

- ✔Конструктор вызывается автоматически при создании экземпляра класса.
- ✔Конструктор имеет тоже имя, что и класс.
- ✔Конструктор не может возвращать значений.

Виды конструкторов:

- ✔конструктор по умолчанию,
- ✔конструктор с параметрами,
- ✔конструктор копирования.

Деструктор – метод, вызываемый автоматически при уничтожении объекта.

- ✔ Деструктор не имеет параметров и не возвращает никакого значения.
- ✔ Имя деструктора совпадает с именем класса, но передним ставится символ тильда (~).
- ✔ У класса может быть только один деструктор.
- ✔ Деструктор выполняет освобождение использованных объектом ресурсов и удаление нестатических переменных объекта.

Деструктор – метод, вызываемый автоматически при уничтожении объекта.

Деструктор автоматически вызывается, когда удаляется объект.

Удаление объект происходит в следующих случаях:

- ✔ когда завершается выполнение области видимости, внутри которой определены объекты
- ✔ когда удаляется контейнер (например, массив), который содержит объекты
- ✔ когда удаляется объект, в котором определены переменные, представляющие другие объекты
- ✔ динамически созданные объекты удаляются при применении к указателю на объект оператора delete

Когда объект автоматически выходит из области видимостиКогда объект автоматически выходит из области видимости или динамически выделенный объект явно удаляется с помощью ключевого слова delete, вызывается деструктор класса (если он существует) для выполнения необходимой очистки до того, как объект будет удалён из памяти.

Для простых классов (тех, которые только инициализируют значения обычных переменных-членов) деструктор не нужен, так как C++ автоматически выполнит очистку самостоятельно

Вывод:

- ✔ конструктор и деструктор объявляется в разделе public;
- ✔ конструктор и деструктор не возвращают значений;
- ✔ имя конструктора и класса должно быть идентично;
- ✔ имя деструктора идентично имени конструктора, но с приставкой тильда ~ ;
- ✔ в классе допустимо создавать несколько конструкторов. Имена будут одинаковыми. Компилятор будет их различать по передаваемым параметрам (как при перегрузке функций). Если мы не передаем в конструктор параметры, он считается конструктором по умолчанию;
- ✔ в классе может быть объявлен только один деструктор.

ЗАДАЧА: создать класс, включающий поля (имя, возраст) и ДВА метода (вывод имени и вывод возраста). ИСПОЛЬЗОВАТЬ **конструктор и деструктор**

```
class persona
{
private:
    string name;
    int age;
public:
   persona(string name, int age)
        {
        this->name = name;
        this->age = age;
    cout << "Мое имя " << name << endl;
    cout << "Мой возраст " << age << endl;
    }
   void sayN()
    {
    cout << "Мое имя " << name << endl;
    }
   void sayA()
    {
    cout << "Мой возраст " << age << endl;
    }
   ~persona()
    {cout << "Деструктор сработал" << endl; }
};
```

```
string N;
int A;
cin >> N >> A;
   persona x (N,A);

   x.sayN();
   x.sayA();
```

ЗАДАЧА: создать класс, включающий поля (имя, возраст). ИСПОЛЬЗОВАТЬ конструктор и деструктор (деструктор выводит сообщение об удалении)

```
class persona
{
private:
    string name;
    int age;
public:
persona(string name, int age)
     {
     this->name = name;
     this->age = age;
     cout << "Мое имя " << name << endl;
     cout << "Мой возраст " << age << endl;
     }
void sayN()
    {
    cout << "Мое имя " << name << endl;
    }
void sayA()
    {
    cout << "Мой возраст " << age << endl;
    }
~persona()
    {cout << "Деструктор сработал" << endl; }
};
```

```
string N;
int A;
cin >> N >> A;
     persona *x =new persona(N,A);
     x->sayN();
     x->sayA();
     delete x;
cout << endl;
system("pause");
return 0;
```

Наследование

метод, который позволяет классы использовать для создания новых классов.

Класс – предок, родитель: класс

Новый класс – дочерний: новый класс: класс

Класс-наследник получает все поля и методы предка, но может добавить собственные поля, добавить собственные методы или перекрыть своими методами одноименные унаследованные методы.

Доступ для	Спецификаторы доступа		
	private	protected	public
Членов собственного класса	**открыт**	**открыт**	**открыт**
Членов производных (дочерних) классов	**закрыт**	**открыт**	**открыт**
Не членов класс	**закрыт**	**закрыт**	**открыт**

Наследование

метод, который позволяет классы использовать для создания новых классов.

ЗАДАЧА:

1) Создать класс `persona`, включающий поля (имя, возраст) и метод вывода имени и возраста
2) Создать класс `family`, включающий поля (имя, возраст, родство) и метод вывода имени, возраста и родства

```
class persona
{
private:
      string name;
      int age;
public:
persona(string name, int age)
      {
      this->name = name;
      this->age = age;
      }
void say()
      {
      cout << "Мое имя " << name << endl;
      cout << "Мой возраст " << age << endl;
      }
~persona()
    {cout << "Деструктор сработал" << endl; }
};
```

```
string N;
int A;
cin >> N >> A;
persona x(N,A);
x.say();
      string N1;
      int A1;
      string R1;
      cin >> N1 >> A1 >> R1;
      family x1(N1, A1, R1);
      x1.say();
```

```
class family
{
private:
      string name;
      int age;
      string rodst;
public:
family(string name, int age, string rodst)
      {
      this->name = name;
      this->age = age;
      this->rodst = rodst;
      }
void say()
      {
      cout << "Мое имя " << name << endl;
      cout << "Мой возраст " << age << endl;
      cout << "Мое родство " << rodst <<
endl;
      }
~family()
      {
      cout << "Деструктор сработал" << endl;
      }
};
```

Наследование

метод, который позволяет классы использовать для создания новых классов.

Синтаксис наследования:

```
class  Имя_Производного_Класса : спецификатор_доступа  Имя_Базового_Класса
{
    /* код */
};
```

Двоеточие **:** не путать с двойным двоеточием :: (определение области действия).

Используя оператор **:** мы показываем, наследником какого класса является производный класс.

Базовый класс

```
class persona
{
private:
      string name;
      int age;
public:
persona(string name, int age)
      {
      this->name = name;
      this->age = age;
      }
void say()
      {
      cout << "Мое имя " << name << endl;
      cout << "Мой возраст " << age << endl;
      }
~persona()
    {cout << "Деструктор сработал" << endl; }
};
```

```
class persona
{
protected:
      string name;
      int age;
public:
persona(string name, int age)
      {
      this->name = name;
      this->age = age;
      }
void say()
      {
      cout << "Мое имя " << name << endl;
      cout << "Мой возраст " << age << endl;
      }
~persona()
    {cout << "Деструктор сработал" << endl; }
};
```

Производный класс

```
class family
{
private:
      string name;
      int age;
      string rodst;
public:
family(string name, int age, string rodst)
      {
      this->name = name;
      this->age = age;
      this->rodst = rodst;
      }
void say()
      {
      cout << "Мое имя " << name << endl;
      cout << "Мой возраст " << age << endl;
      cout << "Мое родство " << rodst <<
endl;
      }

~family()
      {
      cout << "Деструктор сработал" << endl;
      }
};
```

```
class family : public persona
{
private:
     string rodst;

public:
family(string name, int age, string rodst) : persona(name, age)
      {
      this->rodst = rodst;
      }

void say()
     {
     persona::say();
     cout << "Мое родство " << rodst << endl;
     }

~family()
     {
      cout << "Деструктор сработал" << endl;
     }
};
```

Важный момент при наследовании — перегруженные функции-методы класса потомка.

Если в классе родителе и в его классах потомках встречаются методы с одинаковым именем **say()**, то для класса потомка компилятор будет использовать методы именно класса потомка. Может произойти перегрузка метода класса потомка (вызов функцией самой себя).

В таком случае необходимо правильно определить область действия с помощью оператора **::**

```
persona::say();
```

Главная программа

```
string N;
int A;
cin >> N >> A;
persona x(N,A);
x.say();
   string N1;
   int A1;
   string R1;
   cin >> N1 >> A1 >> R1;
   family x1(N1, A1, R1);
   x1.say();
```

Полиморфизм

метод, который позволяет использовать одни и те же методы для решения разных задач.

```
class family : public persona
{
private:
    string rodst;

public:
family(string name, int age, string rodst) : persona(name, age)
     {
     this->rodst = rodst;
     }


void say()
     {
     persona::say();
     cout << "Мое родство " << rodst << endl;
     }


~family()
    {
     cout << "Деструктор сработал" << endl;
    }
};
```

Виртуальная функция — это функция-член, которую предполагается переопределить в производных классах.

Виртуальные методы — один из важнейших приёмов реализации полиморфизма.

Они позволяют создавать общий код, который может работать как с объектами базового класса, так и с объектами любого его класса-наследника.

Базовый класс

```
class persona
{
protected:
      string name;
      int age;
public:
persona(string name, int age)
      {
      this->name = name;
      this->age = age;
      }
void say()
      {
      cout << "Мое имя " << name << endl;
      cout << "Мой возраст " << age << endl;
      }
~persona()
    {cout << "Деструктор сработал" << endl; }
};
```

```
virtual void say()
  {
  cout << "Мое имя " << name << endl;
  cout << "Мой возраст " << age <<
endl;
  }
```

Производный класс

```
class family : public persona
{
private:
    string rodst;

public:
family(string name, int age, string rodst) : persona(name, age)
     {
     this->rodst = rodst;
     }

void say()
    {
    persona::say();
    cout << "Мое родство " << rodst << endl;
    }

~family()
    {
     cout << "Деструктор сработал" << endl;
    }
};
```

```
void say() override
    {
    cout << "Мое имя " << name << endl;
    cout << "Мой возраст " << age << endl;
    cout << "Мое родство " << rodst << endl;
}
```

ЗНАТЬ!!!

ООП
Класс
Поле
Метод
Инкапсуляция
Наследование
Полиморфизм
private
public
protected
Сеттеры
Геттеры
this
Конструктор
Деструктор
Виртуальная функция