### 1. Пояснить смысл термина, привести пример:

**Методология** **ООП** - основана на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определённого класса, а классы образуют иерархию наследования.

**Класс –** это элемент ПО, описывающий абстрактный тип данных и его частичную или полную реализацию.

class MyClass : public ParentClass // ParentClass — класс-предок, если таковой имеется

{

public:

// элементы в этой секции доступны из любой части программы

MyClass();

// конструктор +

~MyClass();

// деструктор

protected:

// элементы в этой секции доступны из класса и его потомков

private:

// элементы в этой секции доступны только из класса; это область доступа по умолчанию

};

**Экземпляр класса –** это описание конкретного объекта в памяти. В объектно-ориентированной программе с применением классов каждый объект является «*экземпляром*» некоторого конкретного класса, и других объектов не предусмотрено. То есть «*экземпляр класса*» в данном случае означает не «пример некоторого класса» или «отдельно взятый класс», а «объект, типом которого является какой-то класс».

MyClass myinstance();

**Инстанцирование -** создание экземпляра класса. В отличие от слова «создание», применяется не к объекту, а к классу. То есть, говорят: *(в виртуальной среде) создать экземпляр класса* или, другими словами, *инстанцировать класс*.

MyClass myinstance;

**Указатель -** переменная, диапазон значений которой состоит из адресов ячеек памяти или специального значения — нулевого адреса.

MyClass myinstance(my);

MyClass\* i\_my = &myinstance;

**Объект** – это отдельный представитель класса, имеющий конкретное состояние и поведение, полностью определяемое классом.

MyClass myinstance(my);

**Инкапсуляция –** это механизм, который объединяет данные и код, манипулирующий этими данными, а также защищает и то, и другое от внешнего вмешательства или неправильного использования.

class A

{

public:

int a, b; //данные открытого интерфейса

int ReturnSomething(); //метод открытого интерфейса

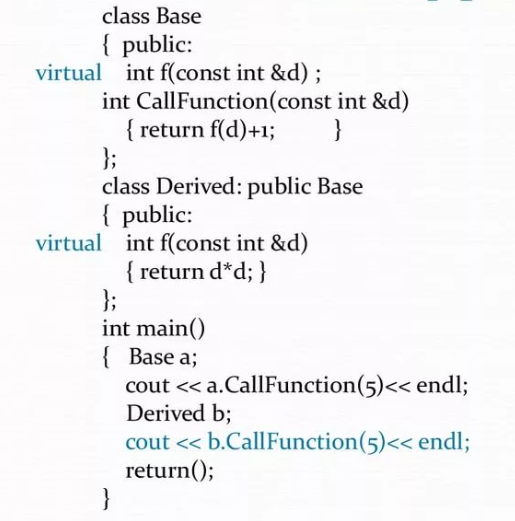
private:

int Aa, Ab; //скрытые данные

void Do\_Something(); //скрытый метод

}

Класс А инкапсулирует свойства Aa, Ab и метод Do\_Something(), представляя внешний интерфейс ReturnSomething, a, b.

**Полиморфизм —**  это свойство, которое позволяет одно и то же имя использовать для решения двух или более схожих, но технически разных задач. Целью полиморфизма, применительно к объектно-ориентированному программированию, является использование одного имени для задания общих для класса действий. Выполнение каждого конкретного действия будет определяться типом данных.

**Наследование —** это процесс, посредством которого один объект может приобретать свойства другого. Точнее, объект может наследовать основные свойства другого объекта и добавлять к ним черты, характерные только для него.

**Предок / потомок**

**Сигнатура** — часть общего объявления функции, позволяющая средствам трансляции идентифицировать функцию среди других.

public: // спецификатор доступа public

CppStudio(int date\_day, int date\_month, int date\_year)

// конструктор класса

**Интерфейс**  — программная/синтаксическая структура, определяющая отношение между объектами, которые разделяют определённое поведенческое множество и не связаны никак иначе. При проектировании классов, разработка интерфейса тождественна разработке спецификации (множества методов, которые каждый класс, использующий интерфейс, *должен* реализовывать).

**Интерфейс класса** — это класс без полей и без реализации, включающий только заголовки методов. Если некий класс наследует (или, как говорят, реализует) интерфейс, он должен реализовать все входящие в него методы.

**Поле** — переменная, связанная с [классом](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) или [объектом](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)). Все данные объекта хранятся в его полях. Доступ к полям осуществляется по их имени. Обычно [тип данных](https://ru.wikipedia.org/wiki/%D0%A2%D0%B8%D0%BF_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) каждого поля задаётся в описании класса, членом которого является поле.

**public**: int y;

int get\_x() {**return** x;};

void set\_x(int x) {**this**->x = x; };

**Метод** — это функция, принадлежащая какому-то классу или объекту.

**public**: void set\_x(int x) {**return x\*x** };

**Конструктор** — специальный блок инструкций, вызываемый при создании объекта

**class** **ClassWithConstructor**

{

**public**:

ClassWithConstructor(float parameter)…

}

**Переопределение метода -** одна из возможностей [языка программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), позволяющая [подклассу](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B4%D0%BA%D0%BB%D0%B0%D1%81%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) или дочернему классу обеспечивать специфическую реализацию [метода](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_(%D1%8F%D0%B7%D1%8B%D0%BA%D0%B8_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)), уже реализованного в одном из [суперклассов](https://ru.wikipedia.org/wiki/%D0%A1%D1%83%D0%BF%D0%B5%D1%80%D0%BA%D0%BB%D0%B0%D1%81%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) или родительских классов.

class Rectangle

{

public:

virtual void print() const

{

cout << "Length = " << length << "; Width = " << width;

}

private:

double length;

double width;

};

class Box : public Rectangle

{

public:

void print() const

{

Rectangle::print(); // вызов родительского метода print()

cout << "; Height= " << height;

}

private:

double height;

};

**Абстрактный класс (метод)**  в [объектно-ориентированном программировании](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) — базовый [класс](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), который не предполагает создания экземпляров

class CA { // Абстрактный класс

public:

CA(void) { std::cout << "This object of the class "; }

virtual void Abstr(void) = 0; // Чистая (пустая) виртуальная функция.

};

class CB : public CA {

public:

void Abstr(void) { std::cout << " call function cb.Abstr();"; } //Подменяющая функция.

};

**Модификатор доступа**

**Структура класса**

**Состояние объекта -** Состояние объекта характеризуется перечнем (обычно статическим) всех свойств данного объекта и текущими (обычно динамическими) значениями каждого из этих свойств.

**Поведение объекта** — это то, как объект действует и реагирует; поведение выражается в терминах состояния объекта и передачи сообщений.

**Статические члены -** Статические член-данные (или поля) класса можно рассматривать как глобальную переменную класса. Но в отличие от обычных глобальных переменных на статические члены распространяются правила видимости private и public.

class X  
{  
  static int n;  
};  
int X::n = 0;

**Коллекции -** программный объект, содержащий в себе, тем или иным образом, набор значений одного или различных типов, и позволяющий обращаться к этим значениям.

**Раннее и позднее связывание.** Связывание — это сопоставление вызова функции с вызовом. В C++ все функции по умолчанию имеют раннее связывание, то есть компилятор и компоновщик решают, какая именно функция должна быть вызвана, до запуска программы. Виртуальные функции имеют позднее связывание, то есть при вызове функции нужное тело выбирается на этапе выполнения программы. Встретив ключевое слово virtual, компилятор помечает, что для этого метода должно использоваться позднее связывание