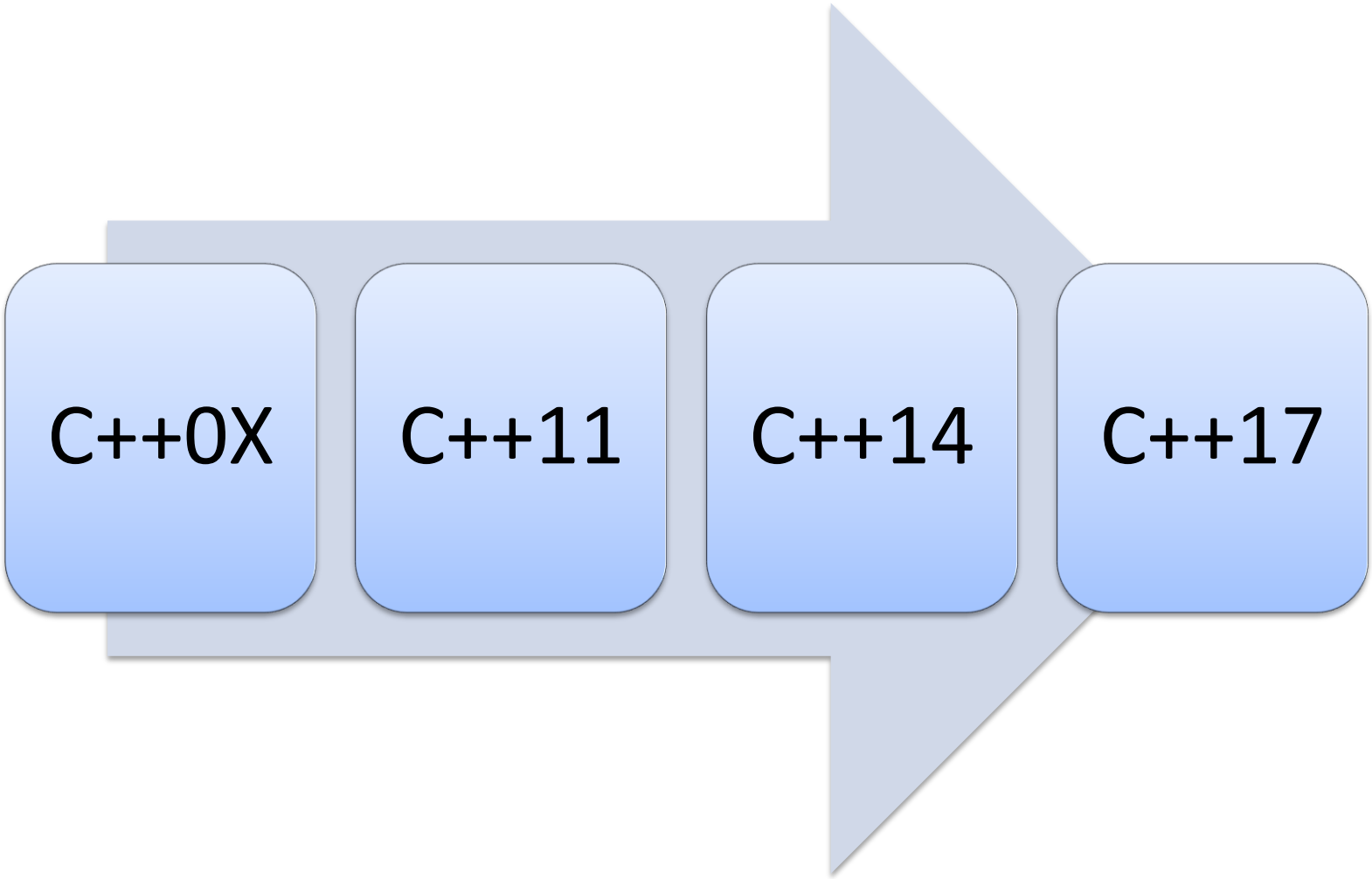
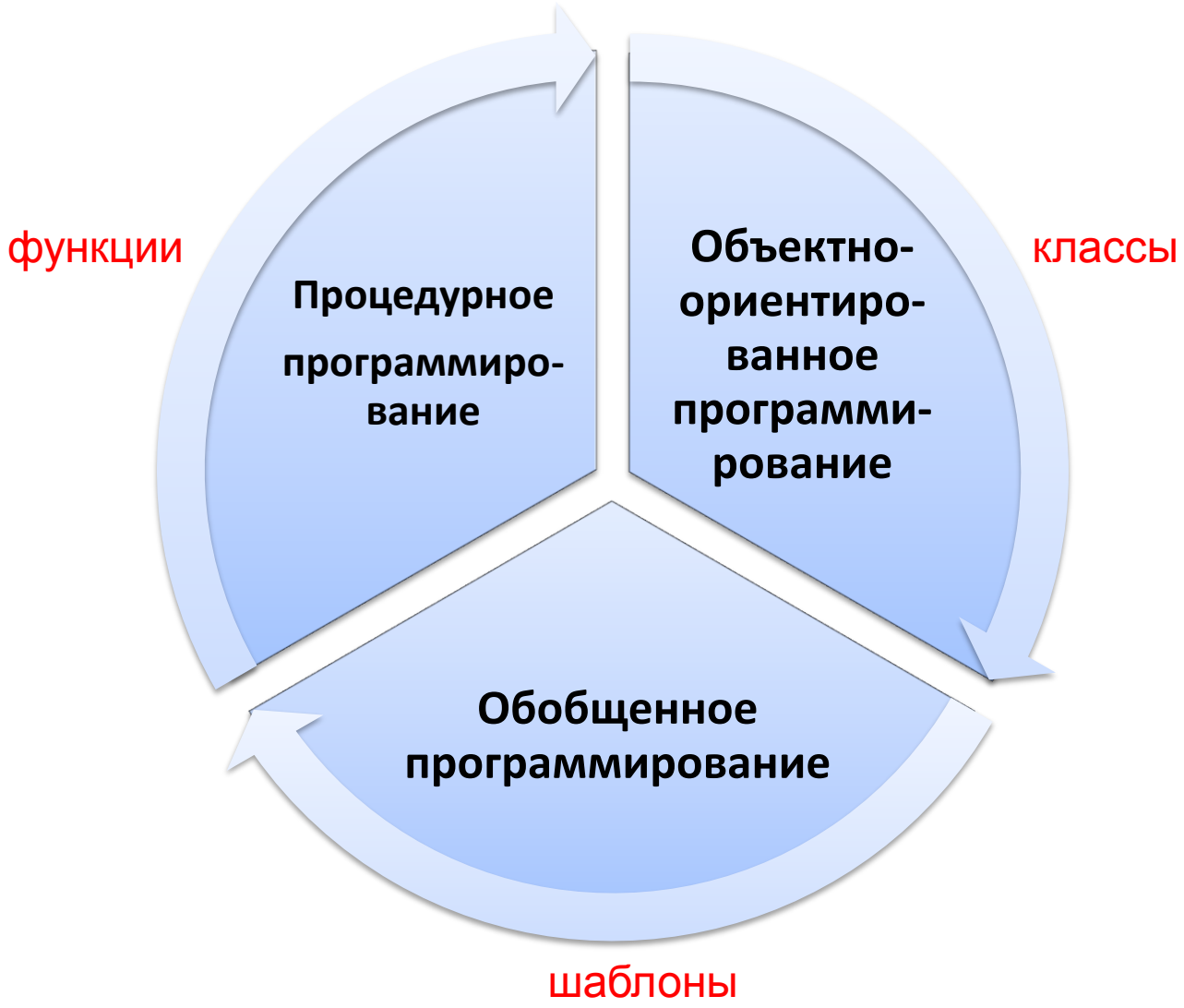
C++: Основы ООП

22 мая 2017 г.

Диалекты C++



C++ поддерживает:



Объектно-ориентированное программирование

подход, при котором функции и переменные, относящиеся к какому-то конкретному объекту, **объединены в коде** определенным образом и **тесно связаны** между собой

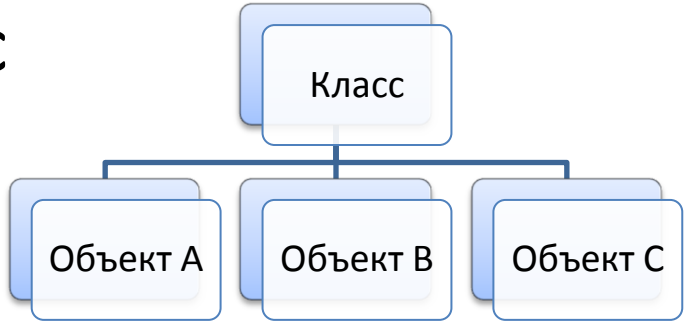
Класс

- Поля (свойства)
- Методы

Объект класса

Класс

- пользовательский тип данных
- содержит поля (данные) и методы (функции)
- класс должен удовлетворять принципу единственной ответственности
- может быть создано много экземпляров (объектов) одного класса
- структура – тоже класс



Задание

Какие поля и методы следует реализовать в следующих классах:

1) хомячок
2) книга
3) арифметическая прогрессия?

«Три кита» ООП

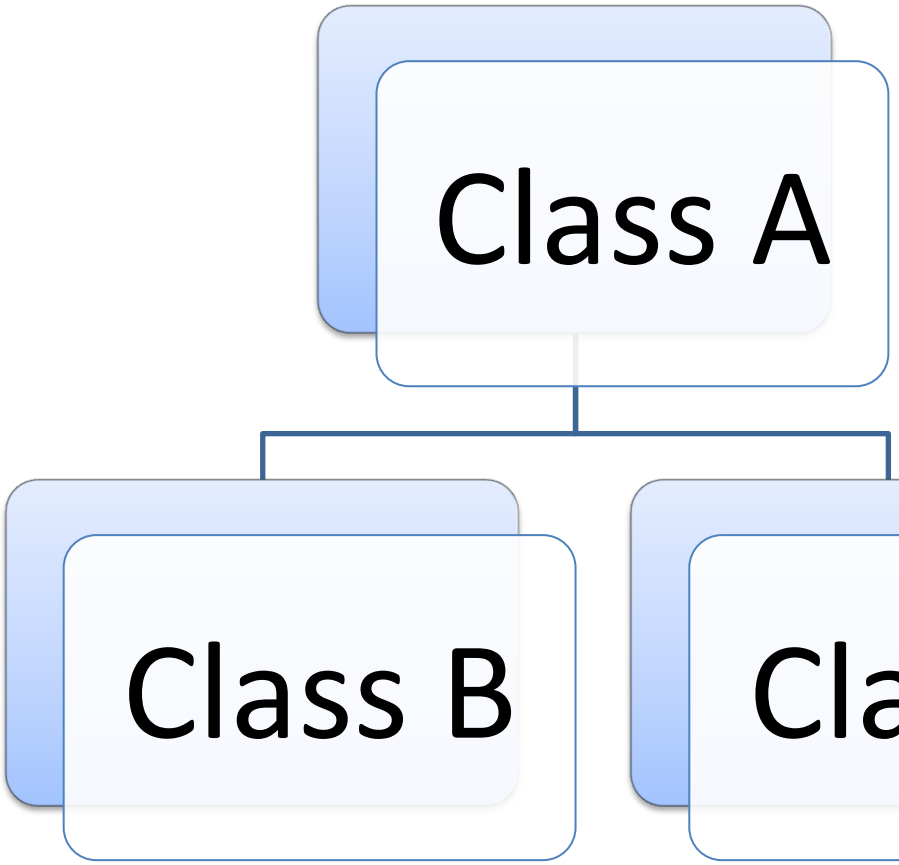
- Инкапсуляция
- Наследование
- Полиморфизм

Инкапсуляция

- сокрытие в классе данных, чтобы они были недоступны снаружи
- объединение данных и методов, с помощью которых эти данные обрабатываются

Принцип «открытости / закрытости»

Наследование



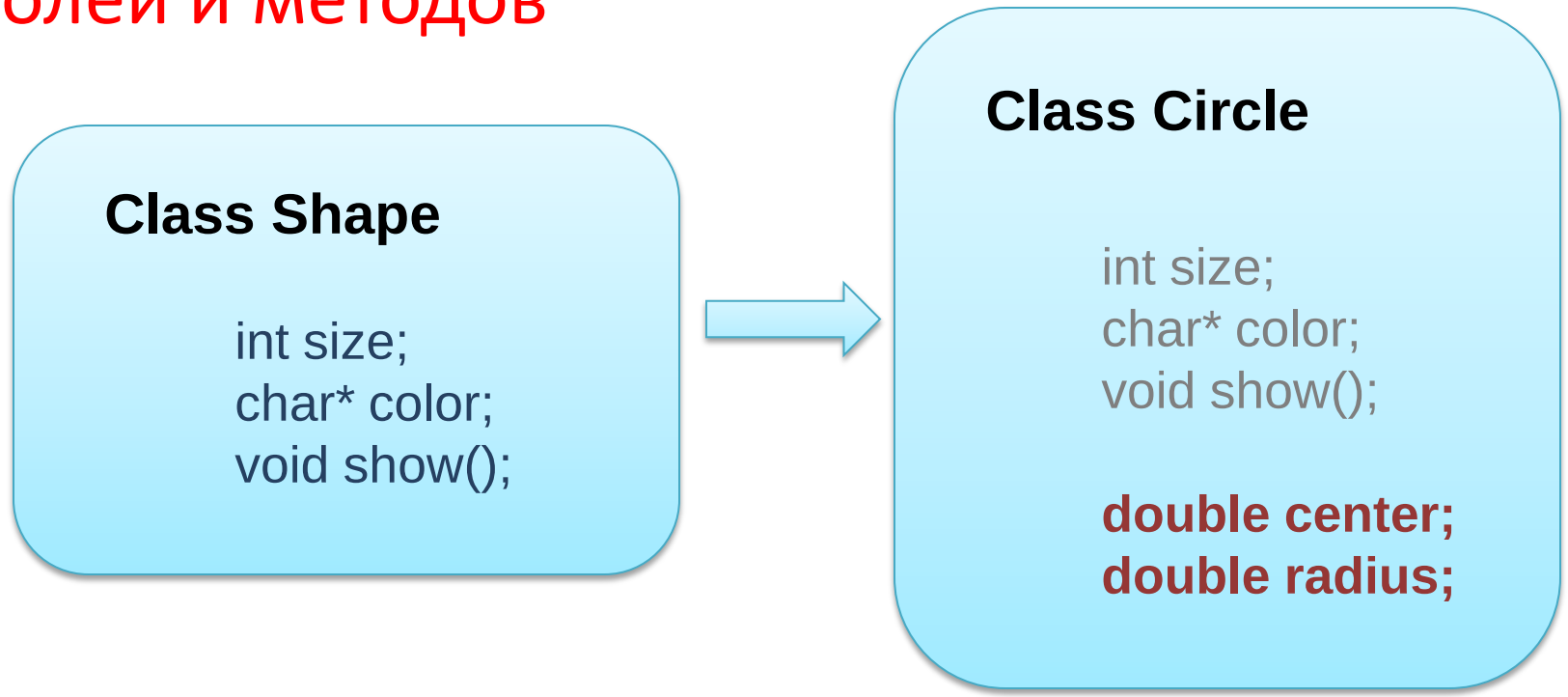
Родительский (базовый) класс

Дочерние (производные) классы

Иерархия классов

Наследование

Производные классы получают «по наследству» данные и методы своих базовых классов и расширяют их функциональность за счет своих полей и методов



Полиморфизм

1) способность объекта вести себя по-разному в зависимости от ситуации;
2) способность объекта использовать методы производного класса

«Один интерфейс – много реализаций»

Класс: структура

- имя класса
- поля
- конструктор
- деструктор
- аксессоры
- методы

```
class MyFirstClass {
    int code;
public:
    MyFirstClass() { }
    MyFirstClass(int x) : code(x) { }
    ~MyFirstClass() { }
    int getCode() const { return code; }
    void setCode(int x) { code = x; }
    void print()
        { std::cout << code << std::endl; }
};
```

Модификаторы доступа

public	• поля и методы доступны везде
protected	• в собственных методах класса • в производных классах
private	• в собственных методах класса

Конструкторы

1. Конструктор по умолчанию
2. Конструктор с параметрами

```
class MyClass
{
public:
    MyClass();
    MyClass(int a);
    MyClass(int a, int b);
};
```

Конструктор по умолчанию

- не имеет параметров
- если в классе нет конструктора, создается автоматически (создает объект по умолчанию)
- если в классе объявлен конструктор, то конструктор по умолчанию не генерируется

```
class MyClass
{
public:
        MyClass() { };
};
```

Конструктор с параметрами

- принимает набор аргументов
- некоторые аргументы могут иметь значения по умолчанию
- нужен для инициализации полей в классе

```
class MyClass {
        int a, b;
public:
        MyClass(int x, int y = 1) {
                a = x;
                b = y;
        }
};
```

Деструктор

- не имеет параметров
- не возвращает значение
- если в классе нет деструктора, создается автоматически
- нужно реализовывать, чтобы освободить ресурсы

```
class MyClass
{
public:
        ~MyClass() { };
};
```

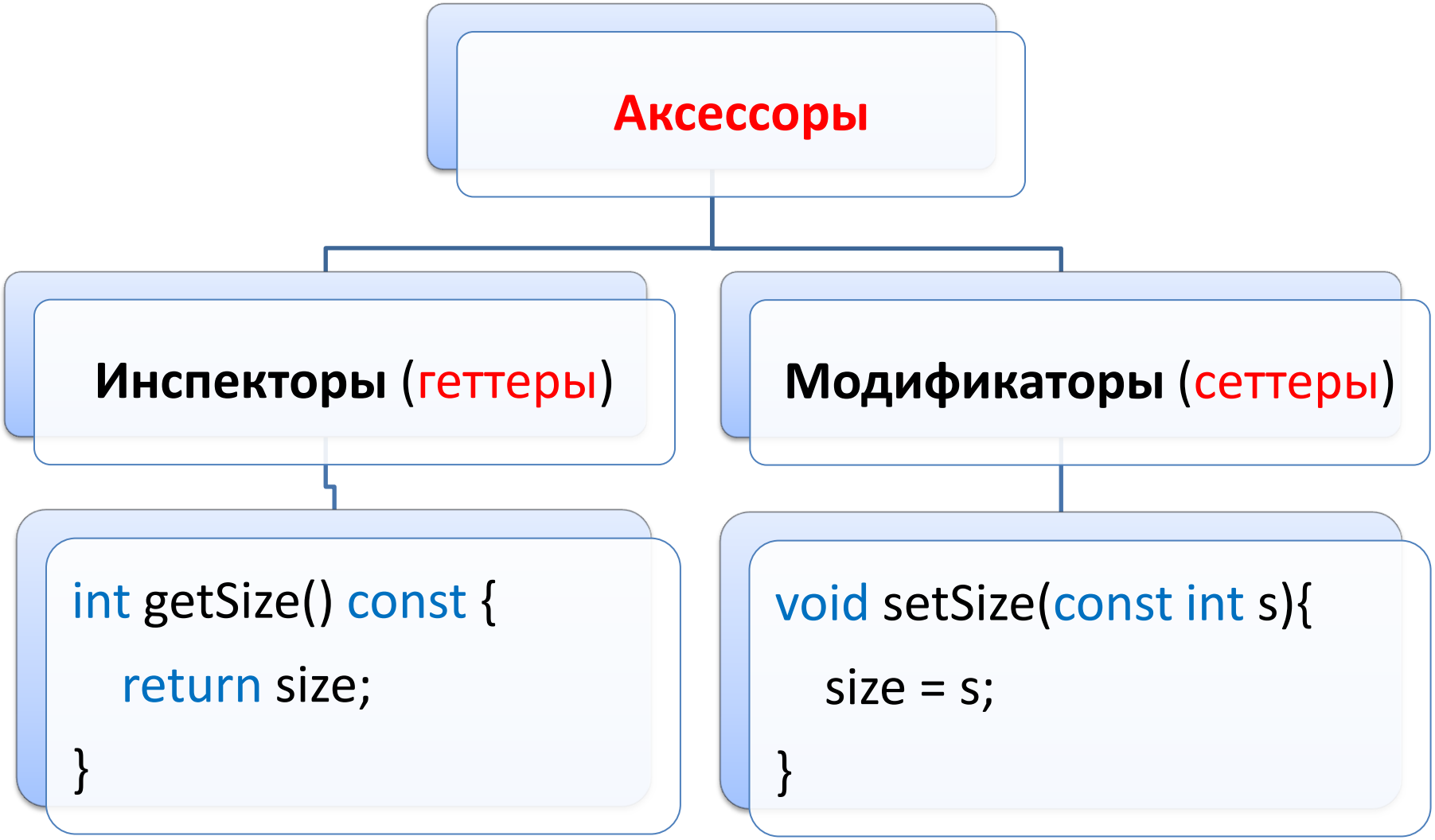
Конструкторы и деструкторы

В каком порядке будут вызваны конструкторы и деструкторы объектов?

```
class Point { };
class Rectangle { };
class Circle { };

void init() {
    Point p;
    Rectangle r;
    Circle c;
}
…
init();
```

Работа с полями



Реализация методов вне объявления класса

```
class MyFirstClass {
    int code;
public:
    MyFirstClass(int x) : code(x) { }
    int getCode() const;
    void setCode(int);
    void print();
};

 int MyFirstClass::getCode() const { return code; }
 void MyFirstClass::setCode(int x) { code = x; }
 void MyFirstClass::print() { std::cout << code << std::endl; }
```

Вопросы?