C++. Базовый уровень

# Классы и объекты. Принципы ООП.

КОД будущего

Минцифры России

инжиниум МГТУ им. Н.Э. Баумана

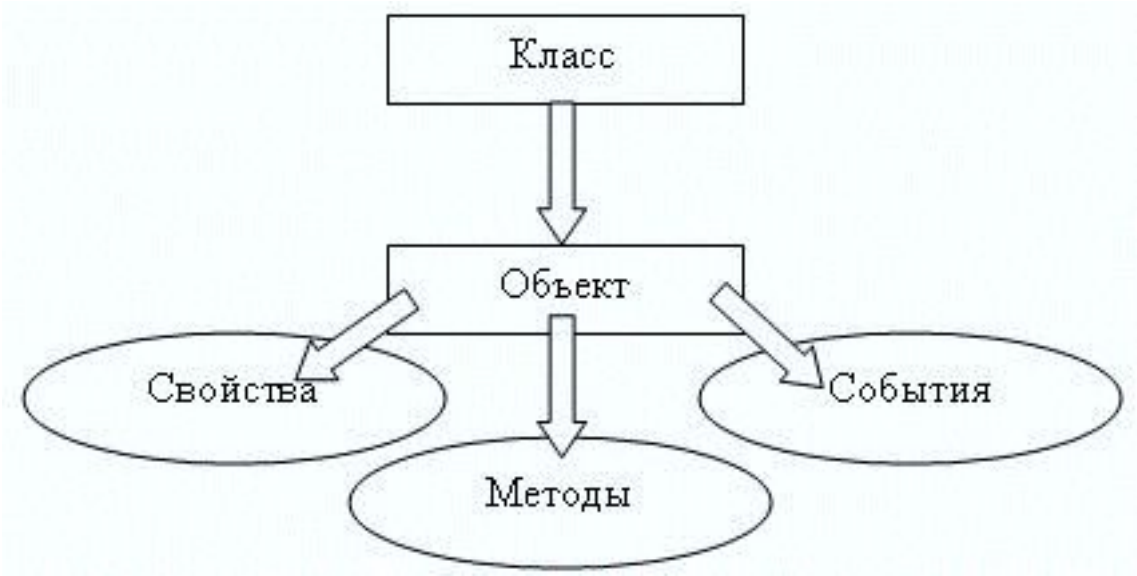
Национальные проекты России

ЦИФРОВАЯ ЭКОНОМИКА

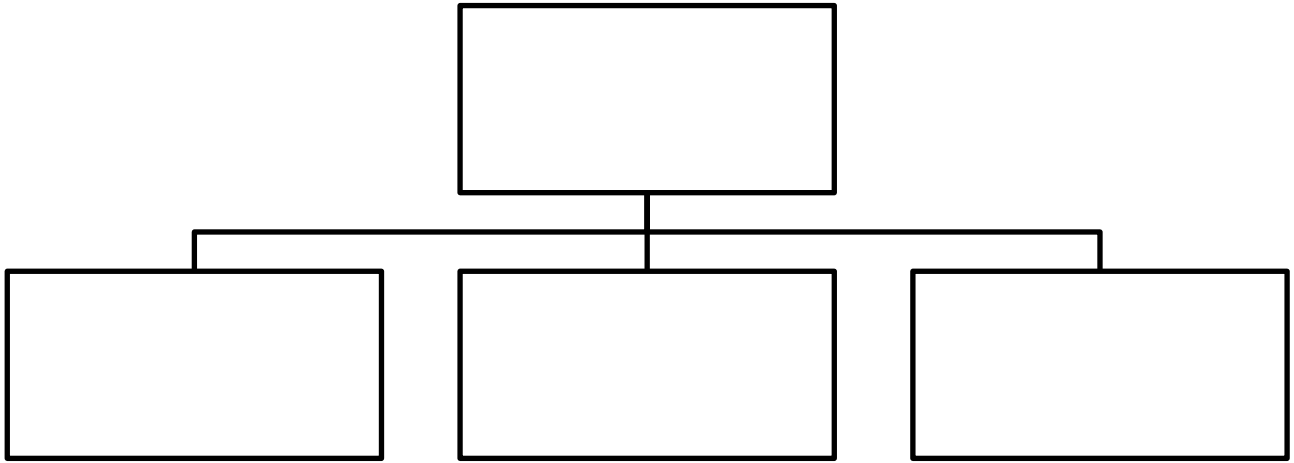
20.35 УНИВЕРСИТЕТ

2023

# Классы и объекты. Принципы ООП.



# Классы и объекты. Принципы ООП.



# Классы и объекты. Принципы ООП.

```
class имя_класса
{
    // компоненты класса
};
```

```
class Person { };

int main()
{

}
```

```
class Person
{

};
int main()
{
    Person person;
}
```

# Классы и объекты. Принципы ООП.

```
class Person
{
public:
    std::string name;
    unsigned age;
    void print()
    {
        std::cout << "Name: " << name << "\tAge: " << age << std::endl;
    }
};
int main()
{
    Person person;
    // устанавливаем значения полей класса
    person.name = "Tom";
    person.age = 38;
    // вызываем функцию класса
    person.print();
}
```

# Классы и объекты. Принципы ООП.

объект.компонент

```
person.name = "Tom";
person.age = 38;

person.print()
```

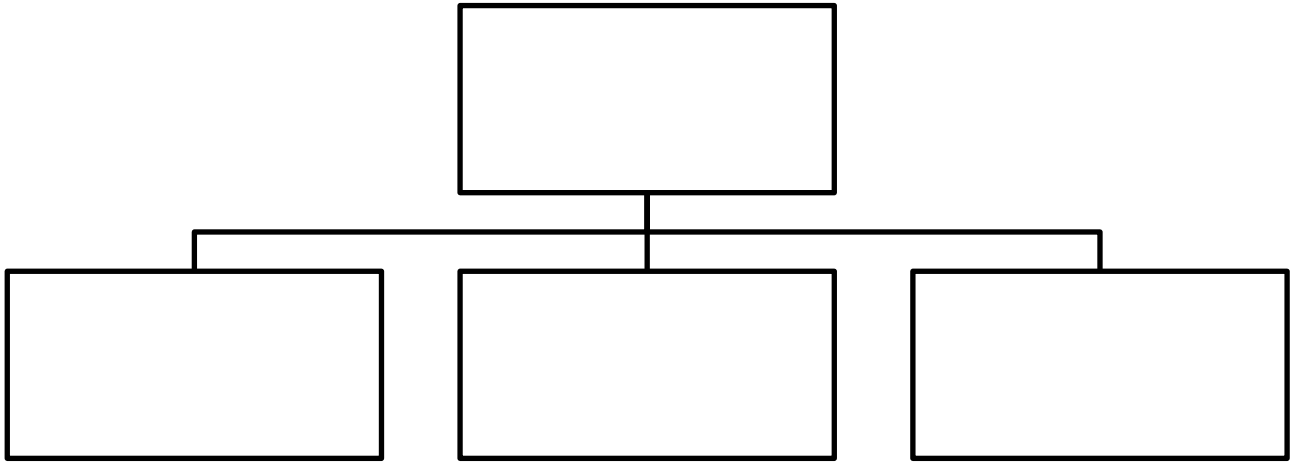
# Классы и объекты. Принципы ООП.

Конструкторы - это специальные функции, которые вызываются при создании объектов класса.
Конструкторы имеют ту же самую имя, что и класс, и не имеют типа возвращаемого значения.
Конструкторы используются для инициализации атрибутов объекта заданными значениями или для выполнения других действий, необходимых при создании объекта.

```
// Класс Point, который представляет точку на плоскости
class Point {
    // атрибуты x и y
    int x;
    int y;
    // Конструктор по умолчанию, который инициализирует x и y нулями
    Point() {
        x = 0;
        y = 0;
    }
};
```

```
// Класс Point, который представляет точку на плоскости
class Point {
    // атрибуты x и y
public:
    int x;
    int y;
    // Конструктор с параметрами, который принимает координаты точки и инициализирует x и y ими
    Point(int x, int y) {
        this->x = x;
        this->y = y;
    }
};
int main() {
    Point p1(10, -10);
    return 0;
}
```

# Классы и объекты. Принципы ООП.



# Классы и объекты. Принципы ООП.

```
class Point {
    int x, y;
};

class Line : Point {
    int len;
};

class Triangle : Line {
    int angel;
};
```

```
class Animal {
    int count;
};

class Cat : Animal {
    string name;
};

class Dog : Animal {
    string name;
};
```

# Классы и объекты. Принципы ООП.

```cpp
#include <iostream>
// Функция add, которая складывает два целых числа
int add(int a, int b) {
    return a + b;
}
// Функция add, которая складывает два вещественных числа
double add(double a, double b) {
    return a + b;
}
// Функция add, которая складывает две строки
std::string add(std::string a, std::string b) {
    return a + b;
}

// Главная функция программы
int main() {
    // Вызов функции add для разных типов данных
    std::cout << add(2, 3) << "\n"; // 5
    std::cout << add(2.5, 3.5) << "\n"; // 6
    std::cout << add("Hello", "World") << "\n"; // HelloWorld
    return 0;
}
```

# Классы и объекты. Принципы ООП.

```
#include <iostream>
using namespace std;
class Animal {
public:
    int count;
    int walk() {
        cout << "walk 10 steps";
    }
};
class Cat : public Animal {
public:
    string name;
    int walk() {
        cout << "walk 5 steps";
    }
};
int main() {
    Animal a1;
    Cat barsik;
    a1.walk();
    barsik.walk();
}
```

# Классы и объекты. Принципы ООП.

Инкапсуляция - это свойство класса, которое ограничивает доступ к его данным и функциям извне, используя модификаторы доступа public, private и protected. Инкапсуляция позволяет скрыть детали реализации класса и обеспечить безопасность и целостность его данных. Инкапсуляция также позволяет изменять внутреннюю логику класса без влияния на другие части программы, которые используют этот класс.

Public означает, что члены класса доступны для всех,

Private означает, что члены класса доступны только для самого класса,

Protected означает, что члены класса доступны для самого класса и его наследников.

```
#include <iostream>
// Базовый класс Animal
class Animal {
private:
    std::string name; // Закрытый атрибут name
protected:
    void make_sound(); // Защищенная функция make_sound
};

// Производный класс Dog
class Dog : public Animal {
public:
    void bark(); // Открытая функция bark
};
```