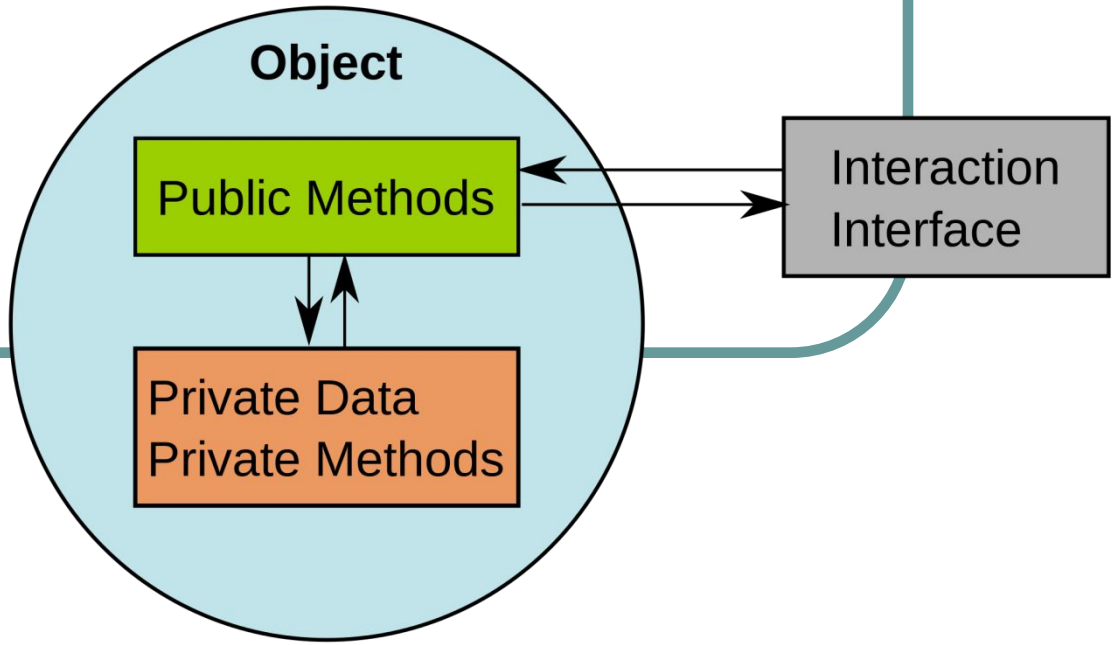
Инкапсуляция (практика)



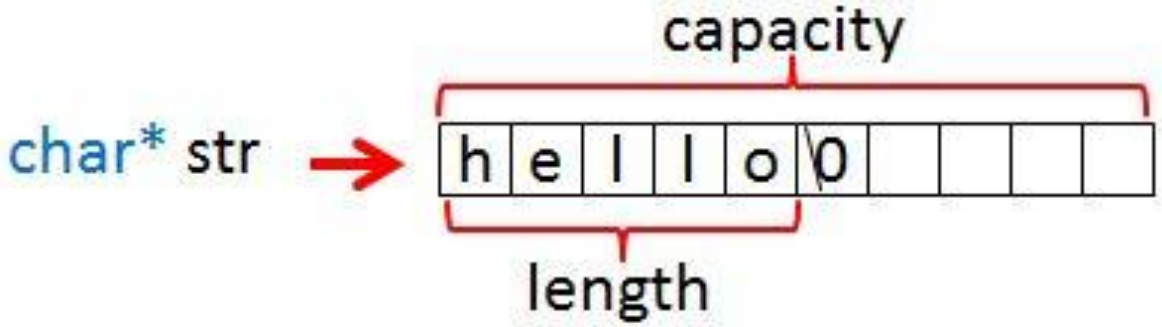
Класс String

Часто приходится работать с классами, объекты которых связаны с блоками данных в динамической памяти. Такая связь чревата «утечкой» памяти. Но классы дают разработчику все средства, чтобы её избежать. Для примера создадим класс, описывающий текстовую строку – благо, есть образец, - стандартный класс string.

Начальная версия класса

```
class String
{
    char* str;
    int length;
    int capacity;

public:
    int GetLength() const
    {
        return length;
    }
};
```



Строка – это массив символов, завершенный нуль-символом. Capacity обозначает фактический размер массива, а length – собственно размер строки. Первый параметр желательно делать больше второго, чтобы было место для расширения строки без необходимости постоянного пересоздания массива.

Добавляем методы

```
class String
{
public:
    String(char* str, int capacity = 80) {
        Initialize(str, capacity);
    }
    String() { Initialize("", 80); }
    String(int capacity) { Initialize("", capacity); }
    String(const String& obj) {
        Initialize(obj.str , obj.capacity);
    }
    ~String() { delete [] str; }
    const char* GetStr() { return str; }
private:
    void Initialize(char* str, int capacity) {
        length = strlen(str);
        capacity = (capacity > length) ?
            capacity : length + 20 ;
        str = new char[capacity];
        strcpy_s(this->str, length + 1, str);
    }
    ...
```

Логика выделения памяти под строку и её заполнение вынесены в приватный метод, который вызывают все конструкторы. Также предоставляется деструктор и геттер, который даст пользователю доступ к внутреннему массиву в режиме readonly.

```
String a;
String b("hello world");
String c = b;
cout << a.getStr() << endl;
cout << b.getStr() << endl;
cout << c.getStr() << endl;
// a.getStr()[5] = 'a'; ERROR COMPILE
```

Геттеры / сеттеры

```
class String
{
public:
    int GetCapacity() const { return capacity; }
    void SetString(const char* str) {
        length = strlen(str);
        if (length >= capacity) {
            delete [] this->str;
            capacity = length + 20;
            this->str = new char[capacity];
        }
        strcpy_s(this->str, length + 1, str);
    }
    void Clear() { str[0] = '\0'; length = 0; }
    void ShrinkToFit() {
        if (length + 1 == capacity) return;
        capacity = length + 1;
        char* temp = new char[capacity];
        strcpy_s(temp, capacity, str);
        delete [] str;
        str = temp;
```

Сеттер не обязательно должен менять только одно поле, и для одного поля может быть несколько сеттеров. Если метод меняет состояние объекта в целом, его называют модификатором.

```
String a("hello world");
cout << a.GetStr() << endl;
cout << a.GetLength() << endl;
cout << a.GetCapacity() << endl;
a.SetString("nice weather");
cout << a.GetStr() << endl;
cout << a.GetLength() << endl;
cout << a.GetCapacity() << endl;
a.Clear();
cout << a.GetStr() << endl;
cout << a.GetLength() << endl;
cout << a.GetCapacity() << endl;
a.ShrinkToFit();
cout << a.GetLength() << endl;
cout << a.GetCapacity() << endl;
```

Пример кода

https://git.io/vo29j

Explicit-конструкторы

Подумайте, как компилятор отреагирует на подобный код:

```
void f(Fraction a)
{
    ...
}

f(3);
```

Компилятор в таком случае прибегнет к неявному преобразованию типов, т.е. в данной ситуации при вызове функции будет вызван конструктор с одним параметром типа `int`, и уже полученный объект будет передан в функцию.

Explicit-конструкторы

Аналогичная ситуация:

```
Fraction f()
{
    ...
    return 3;
}


Fraction a;
a = f();
```

Здесь в операторе `return` компилятор также прибегнет к неявному преобразованию типов, т.е. данной ситуации перед возвратом значения из функции также будет вызван конструктор с одним параметром типа `int` и уже полученный объект будет использован в качестве возвращаемого значения функции.

Explicit-конструкторы

Для предотвращения в таких ситуациях неявных преобразований типов вместе с конструкторами преобразования используется ключевое слово **explicit**.

Если оно присутствует, то вместо неявного преобразования типов компилятор выдаст ошибку несоответствия типов.

Пример на explicit

https://git.io/vo2Q2

Что дальше?

Задание на класс String:

https://yadi.sk/i/RzBDOgMjsbef5

Практика