

Лекция 10. Основы объектно-ориентированного программирования: классы и объекты

Тема:

Инкапсуляция, члены класса, методы, конструкторы и деструкторы.

1. Введение

Объектно-ориентированное программирование (ООП) — это современная парадигма, ориентированная на моделирование реальных сущностей с помощью **объектов и классов**.

Главная идея ООП состоит в объединении данных и функций, которые с ними работают, в единую структуру — **класс**.

ООП повышает **повторное использование кода, масштабируемость и понятность программ**, что делает его основным подходом в языках C++, Java, Python и C#.

Основные принципы ООП:

- **Инкапсуляция** — сокрытие внутренней реализации.
- **Наследование** — использование свойств существующих классов.
- **Полиморфизм** — способность объектов вести себя по-разному.
- **Абстракция** — выделение существенных характеристик объекта.

В данной лекции рассматриваются основы: создание классов, объектов, конструкторов и деструкторов.

2. Классы и объекты

Класс — это шаблон (модель), по которому создаются **объекты**.

Объект — это экземпляр класса, обладающий состоянием (данными) и поведением (методами).

Пример (на C++):

```
class Student {  
public:  
    string name;  
    int age;  
    float gpa;  
};
```

```

void displayInfo() {
    cout << "Имя: " << name << ", Возраст: " << age << ", GPA: " << gpa <<
endl;
}
};

int main() {
    Student s1; // создание объекта
    s1.name = "Айгерим";
    s1.age = 20;
    s1.gpa = 3.8;
    s1.displayInfo();
}

```

В этом примере:

- Student — класс, описывающий студентов;
- s1 — объект этого класса;
- displayInfo() — метод, который выводит данные на экран.

3. Инкапсуляция

Инкапсуляция — это объединение данных и методов, которые с ними работают, в один модуль (класс), а также ограничение доступа к внутренним деталям реализации.

Это достигается с помощью **модификаторов доступа**:

- public — открытый доступ (виден всем);
- private — доступен только внутри класса;
- protected — доступен в классе и его наследниках.

Пример:

```

class BankAccount {
private:
    double balance;

public:
    void deposit(double amount) {
        balance += amount;
    }

    void withdraw(double amount) {

```

```

    if (amount <= balance)
        balance -= amount;
    else
        cout << "Недостаточно средств\n";
}

double getBalance() {
    return balance;
}
};

```

Пользователь класса может работать только через **публичные методы**, не имея прямого доступа к переменной `balance`. Это предотвращает ошибки и защищает данные от некорректного использования.

4. Члены класса

Класс может содержать:

- **Поля (данные)** — переменные, описывающие состояние объекта.
- **Методы (функции)** — описывают поведение.
- **Конструкторы и деструкторы** — специальные методы для создания и удаления объектов.
- **Статические члены** — общие для всех экземпляров.

Пример:

```

class Circle {
private:
    double radius;
    static const double PI;

public:
    Circle(double r) { radius = r; }
    double area() { return PI * radius * radius; }
};

const double Circle::PI = 3.14159;

```

Здесь `PI` — **статическая константа**, общая для всех объектов класса.

5. Методы класса

Методы определяют, **что объект умеет делать**.
Их можно реализовать **внутри** или **вне** класса.

Пример:

```
class Rectangle {  
private:  
    double width, height;  
  
public:  
    void setDimensions(double w, double h);  
    double area();  
};  
  
void Rectangle::setDimensions(double w, double h) {  
    width = w;  
    height = h;  
}  
  
double Rectangle::area() {  
    return width * height;  
}
```

6. Конструкторы

Конструктор — это специальный метод, который автоматически вызывается при создании объекта.
Он инициализирует данные класса.

Пример:

```
class Student {  
public:  
    string name;  
    int age;  
  
    // Конструктор  
    Student(string n, int a) {  
        name = n;  
        age = a;  
    }  
  
    void display() {
```

```

        cout << name << " - " << age << " лет" << endl;
    }
};

int main() {
    Student s1("Айдана", 21);
    s1.display();
}

```

Можно определять несколько конструкторов (**перегрузка конструкторов**):

```
Student() { name = "Неизвестно"; age = 0; }
```

7. Деструкторы

Деструктор вызывается автоматически при уничтожении объекта и используется для освобождения ресурсов (например, памяти, файлов, соединений).

Обозначается символом ~ перед именем класса.

Пример:

```

class FileHandler {
public:
    FileHandler() { cout << "Файл открыт\n"; }
    ~FileHandler() { cout << "Файл закрыт\n"; }
};

int main() {
    FileHandler fh; // При выходе из функции вызовется деструктор
}

```

8. Пример комплексного класса

```

class Car {
private:
    string brand;
    int year;
    double mileage;

public:
    Car(string b, int y, double m) {
        brand = b;
        year = y;
    }
}

```

```

        mileage = m;
    }

    void drive(double km) {
        mileage += km;
        cout << "Машина проехала " << km << " км." << endl;
    }

    void info() {
        cout << brand << " (" << year << ") - пробег: " << mileage << " км." <<
endl;
    }

    ~Car() {
        cout << "Автомобиль " << brand << " удален из памяти.\n";
    }
};

```

9. Принципы хорошего проектирования классов

- **Инкапсулируй всё, что можно** — не открывай внутренние данные напрямую.
 - **Используй конструкторы для корректной инициализации.**
 - **Деструктор** должен освобождать ресурсы.
 - **Методы** должны быть логически связаны с поведением объекта.
 - **Поля** — отражать состояние.
-

10. Пример на Python (для сравнения)

```

class Student:
    def __init__(self, name, gpa):
        self.name = name
        self.gpa = gpa

    def display(self):
        print(f"{self.name} имеет средний балл {self.gpa}")

s1 = Student("Алия", 3.95)
s1.display()

```

Python автоматически вызывает `__init__()` (конструктор) и `__del__()` (деструктор).

11. Заключение

Основные понятия классов и объектов лежат в основе современного программирования.

Они позволяют строить сложные системы, объединяя данные и функции в логические единицы — **объекты**.

ООП делает программы **гибкими, масштабируемыми и удобными для сопровождения**.

Понимание принципов **инкапсуляции**, работы **конструкторов** и **деструкторов** является необходимым этапом для перехода к более сложным темам — наследованию, полиморфизму и абстракции.

Список литературы

1. Лафоре, Р. *Объектно-ориентированное программирование в C++*. — Питер, 2019.
2. Страуструп, Б. *Язык программирования C++*. — Вильямс, 2013.
3. Шилдт, Г. *C++ для начинающих*. — Эксмо, 2020.
4. Eckel, B. *Thinking in C++*. — MindView Press, 2000.
5. Horstmann, C. *Big C++*. — Wiley, 2014.
6. Lippman, S. *C++ Primer*. — Addison-Wesley, 2012.