Объектно-ориентированное программирование

Лекция 1: Введение и основы ООП

Преподаватель: Каледин О.Е.

Группа: 203

Семестр: Осень 2025

📚 Цель лекции

Понять основные принципы объектно-ориентированного программирования, изучить историю развития ООП и освоить базовые концепции на примерах.

План лекции

Часть 1: Теоретические основы

1. **Что такое ООП?**
2. **История развития ООП**
3. **Основные принципы ООП**
4. **Преимущества и недостатки ООП**
5. **Сравнение с другими парадигмами**
6. **Почему именно Java?**
7. **Наш проект: Пошаговая стратегия**
8. **Структура курса**
9. **Практические примеры**

Что такое ООП?

Объектно-ориентированное программирование — парадигма программирования, основанная на концепции "объектов", которые содержат данные и код.

Ключевые концепции:

- **Объекты** — экземпляры классов
- **Классы** — шаблоны для создания объектов
- **Наследование** — создание новых классов на основе существующих
- **Инкапсуляция** — объединение данных и методов
- **Полиморфизм** — использование объектов разных типов

🏛️ История развития ООП

Истоки (1960-1970)

- **Simula 67** - первый язык с концепцией классов и объектов
- **Smalltalk** - первый полностью объектно-ориентированный язык
- **Ada** - военный язык с поддержкой ООП

Расцвет (1980-1990)

- **C++** - расширение C с поддержкой ООП
- **Eiffel** - язык с контрактным программированием
- **Java** - платформо-независимый язык ООП

Современность (1990-настоящее время)

🎯 Основные принципы ООП

1. Абстракция (Abstraction)

- Выделение существенных характеристик объекта
- Скрытие сложности реализации

2. Инкапсуляция (Encapsulation)

- Объединение данных и методов в классе
- Контроль доступа к данным

3. Наследование (Inheritance)

- Создание новых классов на основе существующих
- Переиспользование кода

Инкапсуляция (Encapsulation)

Принцип: Объединение данных и методов в единый объект с контролируемым доступом.

```
public class BankAccount {
    private double balance;  // Приватные данные

    public void deposit(double amount) {  // Публичный метод
        if (amount > 0) {
            balance += amount;
        }
    }

    public double getBalance() {  // Контролируемый доступ
        return balance;
    }
}
```

Преимущества:

Наследование (Inheritance)

Принцип: Создание новых классов на основе существующих с наследованием их свойств.

```
// Базовый класс
public class Animal {
    protected String name;
    protected int age;

    public void eat() {
        System.out.println(name + " ест");
    }
}

// Наследник
public class Dog extends Animal {
    private String breed;

    public void bark() {
```

Полиморфизм (Polymorphism)

Принцип: Возможность объектов с одинаковым интерфейсом иметь различную реализацию.

```
// Интерфейс
public interface Shape {
    double getArea();
}

// Реализации
public class Circle implements Shape {
    private double radius;

    @Override
    public double getArea() {
        return Math.PI * radius * radius;
    }
}

public class Rectangle implements Shape {
    private double width, height;

    @Override
```

Полиморфизм (продолжение)

```
// Полиморфное использование
public void printArea(Shape shape) {
    System.out.println("Площадь: " + shape.getArea());
}

// Использование
Circle circle = new Circle(5);
Rectangle rect = new Rectangle(4, 6);

printArea(circle);   // Площадь: 78.54...
printArea(rect);     // Площадь: 24.0
```

Типы полиморфизма:

- **Переопределение методов** (Override)
- **Перегрузка методов** (Overload)
- **Полиморфизм интерфейсов**

Абстракция (Abstraction)

Принцип: Выделение существенных характеристик объекта и игнорирование несущественных.

```
// Абстрактный класс
public abstract class Vehicle {
    protected String brand;
    protected String model;

    public abstract void startEngine();
    public abstract void stopEngine();

    public void displayInfo() {
        System.out.println(brand + " " + model);
    }
}

// Конкретная реализация
public class Car extends Vehicle {
    @Override
    public void startEngine() {
        System.out.println("Двигатель автомобиля запущен");
    }

    @Override
```

✅ Преимущества ООП

1. Переиспользование кода

- Наследование позволяет создавать новые классы на основе существующих
- Композиция позволяет включать объекты в другие объекты

2. Упрощение разработки

- Модульность и структурированность кода
- Четкое разделение ответственности

3. Легкость сопровождения

- Изменения в одном месте не влияют на другие части
- Четкие интерфейсы между компонентами

❌ Недостатки ООП

1. Сложность для простых задач

- Избыточность для простых программ
- Накладные расходы на создание объектов

2. Производительность

- Дополнительные вызовы методов
- Использование памяти для объектов

3. Сложность понимания

- Множество концепций для изучения
- Возможность создания сложных иерархий

🔄 Сравнение с другими парадигмами

ООП vs Процедурное программирование

Аспект	ООП	Процедурное
Организация	По объектам	По функциям
Данные	Инкапсулированы	Глобальные/локальные
Переиспользование	Наследование/композиция	Функции/библиотеки
Сложность	Высокая	Низкая
Масштабируемость	Хорошая	Ограниченная

🔄 Сравнение с другими парадигмами

ООП vs Функциональное программирование

Аспект	ООП	Функциональное
Состояние	Изменяемое	Неизменяемое
Функции	Методы объектов	Первоклассные функции
Данные	Объекты	Структуры данных
Побочные эффекты	Разрешены	Минимизированы
Параллелизм	Сложный	Простой

Почему именно Java?

Преимущества для изучения ООП:

✅ **Чистый ООП** — все является объектом
✅ **Строгая типизация** — безопасность типов
✅ **Автоматическое управление памятью**
✅ **Кроссплатформенность**
✅ **Богатая стандартная библиотека**
✅ **Активное сообщество**

Наш проект: Пошаговая стратегия

Концепция игры:

- **Жанр:** Пошаговая стратегия
- **Сеттинг:** Фэнтези мир
- **Механика:** Управление армией, ресурсами, территориями

Что мы будем разрабатывать:

- Система юнитов и их характеристик
- Игровое поле и механика движения
- Система боя и тактики
- Управление ресурсами
- ИИ противника

Архитектура игры

```
Game (Основной класс)
├── GameBoard (Игровое поле)
├── Player (Игрок)
│   ├── Army (Армия)
│   │   ├── Unit (Юнит)
│   │   └── Commander (Командир)
├── ResourceManager (Управление ресурсами)
└── GameEngine (Игровой движок)
```

Структура курса

ООП блок (12 лекций + 18 лабораторных):

Лекции 1-6: Основы ООП

- Введение в ООП
- Классы и объекты
- Наследование и полиморфизм
- Интерфейсы и абстрактные классы
- Исключения и обработка ошибок

Лекции 7-12: Продвинутые темы

- Коллекции и generics
- Потоки и файлы

Лабораторные работы

Блок 1 (1-6): Основы ООП

- Создание базовых классов
- Реализация наследования
- Работа с интерфейсами

Блок 2 (7-12): Игровая логика

- Система юнитов
- Игровое поле
- Механика движения

Блок 3 (13-18): Продвинутые функции

💻 Практические примеры

Пример 1: Система управления библиотекой

```
public class Book {
    private String title;
    private String author;
    private String isbn;
    private boolean isAvailable;

    public Book(String title, String author, String isbn) {
        this.title = title;
        this.author = author;
        this.isbn = isbn;
        this.isAvailable = true;
    }

    public void borrow() {
        if (isAvailable) {
            isAvailable = false;
            System.out.println("Книга '" + title + "' выдана");
        } else {
            System.out.println("Книга '" + title + "' уже выдана");
        }
    }

    public void returnBook() {
```

Система управления библиотекой (продолжение)

```
public class Library {
    private List<Book> books;

    public Library() {
        this.books = new ArrayList<>();
    }

    public void addBook(Book book) {
        books.add(book);
        System.out.println("Книга '" + book.getTitle() + "' добавлена в библиотеку");
    }

    public Book findBook(String title) {
        for (Book book : books) {
            if (book.getTitle().equals(title)) {
                return book;
            }
        }
        return null;
    }

    public void displayAllBooks() {
        System.out.println("Книги в библиотеке:");
        for (Book book : books) {
            String status = book.isAvailable() ? "доступна" : "выдана";
            System.out.println("- " + book.getTitle() + " (" + book.getAuthor() + ") - " + status);
        }
    }
}
```

Пример 2: Иерархия животных

```
public abstract class Animal {
    protected String name;
    protected int age;

    public Animal(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public abstract void makeSound();
    public abstract void move();

    public void sleep() {
        System.out.println(name + " спит");
    }

    public void eat() {
        System.out.println(name + " ест");
    }
}
```

Иерархия животных (продолжение)

```
public class Dog extends Animal {
    private String breed;

    public Dog(String name, int age, String breed) {
        super(name, age);
        this.breed = breed;
    }

    @Override
    public void makeSound() {
        System.out.println(name + " лает: Гав-гав!");
    }

    @Override
    public void move() {
        System.out.println(name + " бежит на четырех лапах");
    }

    public void fetch() {
        System.out.println(name + " приносит мячик");
    }
}
```

Иерархия животных (продолжение)

```
public class Bird extends Animal {
    private boolean canFly;

    public Bird(String name, int age, boolean canFly) {
        super(name, age);
        this.canFly = canFly;
    }

    @Override
    public void makeSound() {
        System.out.println(name + " поет: Чик-чирик!");
    }

    @Override
    public void move() {
        if (canFly) {
            System.out.println(name + " летит");
        } else {
            System.out.println(name + " ходит на двух лапках");
        }
    }
}
```

🧠 Вопросы для самопроверки

Базовые вопросы

1. **Что такое объект в ООП?**
    - Объект - это экземпляр класса, содержащий данные и методы для работы с ними.
2. **В чем разница между классом и объектом?**
    - Класс - это шаблон/чертеж, объект - это конкретная реализация класса.
3. **Что означает принцип инкапсуляции?**
    - Объединение данных и методов в единый объект с контролируемым доступом.

Вопросы для самопроверки (продолжение)

Продвинутые вопросы

4. **Когда лучше использовать наследование, а когда композицию?**
    - Наследование для "является" отношений, композиция для "содержит" отношений.
5. **Что такое полиморфизм и какие его виды вы знаете?**
    - Полиморфизм - способность объектов с одинаковым интерфейсом иметь разную реализацию.
6. **Какие проблемы может вызвать неправильное использование ООП?**
    - Сложные иерархии, избыточное наследование, нарушение принципов SOLID.

📖 Дополнительные материалы

📚 Основная литература на русском языке

Учебники по ООП

- **"Объектно-ориентированное программирование"** - А.А. Абрамов, М.А. Трифонов, С.А. Трифонова
- **"Объектно-ориентированное программирование"** - В.В. Подбельский, С.С. Фомин
- **"Объектно-ориентированное программирование в C++"** - Р. Лафоре
- **"Объектно-ориентированное программирование"** - Е.А. Роганов, Н.А. Роганова

Учебники по Java

🎯 Заключение

Объектно-ориентированное программирование представляет собой мощную парадигму, которая:

✅ **Упрощает моделирование** сложных систем
✅ **Повышает переиспользование** кода
✅ **Улучшает сопровождение** программного обеспечения
✅ **Обеспечивает масштабируемость** решений

Ключевые моменты для запоминания

- **4 основных принципа**: Инкапсуляция, Наследование, Полиморфизм, Абстракция
- **Объект** - это экземпляр класса с данными и методами
- **Класс** - это шаблон для создания объектов
- **ООП** подходит для сложных систем, но может быть избыточным для простых задач

Следующая лекция:

"Наследование и полиморфизм" - изучение наследования, абстрактных классов и переопределения методов.

🎯 Переход к практическим основам

Теперь, когда мы изучили теоретические основы ООП, давайте перейдем к практическому применению этих принципов в Java.

📚 Часть 2: Классы и объекты в Java

План второй части:

1. **Синтаксис Java** и основные правила
2. **Структура класса** и его компоненты
3. **Конструкторы** и их типы
4. **Модификаторы доступа** и их применение
5. **Практические примеры** создания классов

Синтаксис Java

Основные правила:

- **Регистрозависимость** — `Unit` ≠ `unit`
- **Точка с запятой** обязательна в конце выражений
- **Фигурные скобки** для блоков кода
- **Имена классов** начинаются с заглавной буквы
- **Имена методов** начинаются с маленькой буквы

Структура класса

```
[модификаторы] class ИмяКласса {
    // Поля (переменные)
    [модификаторы] тип имяПоля;

    // Конструкторы
    [модификаторы] ИмяКласса(параметры) {
        // инициализация
    }

    // Методы
    [модификаторы] возвращаемыйТип имяМетода(параметры) {
        // тело метода
        return значение;
    }
}
```

Пример простого класса

```
public class Unit {
    // Поля класса
    private String name;
    private int health;
    private int attack;

    // Конструктор
    public Unit(String name, int health, int attack) {
        this.name = name;
        this.health = health;
        this.attack = attack;
    }

    // Методы
    public void takeDamage(int damage) {
        this.health -= damage;
        if (this.health < 0) {
            this.health = 0;
        }
    }

    public boolean isAlive() {
        return this.health > 0;
    }
}
```

Конструкторы

Назначение:

- **Инициализация объекта** при создании
- **Установка начальных значений** полей
- **Выполнение подготовительных операций**

Особенности:

- Имя конструктора = имя класса
- Нет возвращаемого значения
- Может быть несколько конструкторов (перегрузка)
- Автоматически вызывается при `new`

Типы конструкторов

```
public class Unit {
    private String name;
    private int health;
    private int attack;

    // Конструктор по умолчанию
    public Unit() {
        this.name = "Unknown";
        this.health = 100;
        this.attack = 10;
    }

    // Параметризованный конструктор
    public Unit(String name, int health, int attack) {
        this.name = name;
        this.health = health;
        this.attack = attack;
    }

    // Конструктор копирования
    public Unit(Unit other) {
        this.name = other.name;
        this.health = other.health;
        this.attack = other.attack;
    }
}
```

Модификаторы доступа

`public` — доступ везде

```
public String name;          // Поле доступно из любого места
public void attack() { }     // Метод доступен из любого места
```

`private` — доступ только внутри класса

```
private int health;          // Поле доступно только внутри класса
private void heal() { }      // Метод доступен только внутри класса
```

`protected` — доступ в пакете и наследниках

```
protected int experience;  // Доступ в пакете и наследниках
```

Ключевое слово `this`

Использование:

- **Обращение к полям** текущего объекта
- **Вызов конструкторов** текущего класса
- **Передача ссылки** на текущий объект

```
public class Unit {
    private String name;
    private int health;

    public Unit(String name, int health) {
        this.name = name;       // this.name – поле класса
        this.health = health;  // name – параметр метода
    }

    public Unit copy() {
```

Создание объектов

```
public class Main {
    public static void main(String[] args) {
        // Создание объекта с помощью new
        Unit warrior = new Unit("Warrior", 150, 25);
        Unit archer = new Unit("Archer", 100, 30);

        // Вызов методов
        warrior.takeDamage(50);
        System.out.println("Warrior alive: " + warrior.isAlive());

        // Создание копии
        Unit warriorCopy = new Unit(warrior);
    }
}
```

Практический пример: Иерархия юнитов

```
// Базовый класс для всех юнитов
public abstract class Unit {
    protected String name;
    protected int health;
    protected int maxHealth;
    protected int attack;
    protected int defense;
    protected Position position;

    public abstract void performAction();
    public abstract String getUnitType();
}

// Конкретные типы юнитов
public class Warrior extends Unit { }
public class Archer extends Unit { }
public class Mage extends Unit { }
```

Что дальше?

На следующей лекции:

- **Наследование и полиморфизм**
- **Абстрактные классы**
- **Переопределение методов**

Подготовка:

- Установить Java JDK 11+
- Установить IDE (IntelliJ IDEA или Eclipse)
- Прочитать главу 3-4 из учебника
- Практиковаться в создании классов

Вопросы?

Контакты:

- **Email:** [ваш.email@university.edu]
- **Telegram:** [@username]
- **Офис:** [номер кабинета]

Следующая лекция: Наследование и полиморфизм