Основы объектно-ориентированного программирования

Java: ООП простыми словами

Классы и объекты, инкапсуляция, наследование, полиморфизм, абстракция и композиция

---

1. Класс и объект

Класс — это описание того, какие данные и действия будут у объектов данного типа.
Объект — конкретный экземпляр класса.
Например, можно создать класс `Cat`:

```
public class Cat {
    String name;

    public void meow() {
        System.out.println("Meow!");
    }
}
```

А затем создать объект:

```
Cat cat = new Cat();

cat.name = "Murka";
cat.meow();
```

Здесь `Cat` — класс, а `cat` — объект этого класса.

Главная идея

Объект обычно объединяет:

- данные — его **состояние** ;
- методы — его **поведение** .

2. Инкапсуляция

Инкапсуляция означает, что внутреннее состояние объекта защищено от неправильного изменения.
Рассмотрим игрока, у которого есть здоровье:

```
public class Player {

    private int health = 100;

    public void takeDamage(int damage) {
        if (damage > 0) {
            health = Math.max(0, health - damage);
        }
    }

    public int getHealth() {
        return health;
    }
}
```

Поле `health` объявлено как `private`. Поэтому другой код не может произвольно записать в него любое значение.
Изменение выполняется через понятный метод:

```
player.takeDamage(20);
```

А такой код недоступен:

```
player.health = -500; // Ошибка
```

Инкапсуляция помогает объекту сохранять корректное состояние.

Важно

Инкапсуляция — это не просто `private`.
Главная идея — разрешать изменять состояние только безопасным и понятным способом.

3. Наследование

Наследование позволяет создать новый класс на основе уже существующего. Обычно это отношение:

«является разновидностью».

Например, собака и кошка являются животными.

```
public class Animal {

    public void sound() {
        System.out.println("Some sound");
    }
}
```

Класс `Dog` наследуется от `Animal`:

```
public class Dog extends Animal {

    @Override
    public void sound() {
        System.out.println("Woof!");
    }
}
```

А класс `Cat` может реализовать тот же метод по-своему:

```
public class Cat extends Animal {

    @Override
    public void sound() {
        System.out.println("Meow!");
    }
}
```

Ключевое слово `extends` означает наследование.

4. Полиморфизм

Полиморфизм позволяет работать с разными объектами через общий тип. Например:

```
Animal animal = new Dog();

animal.sound();
```

Переменная имеет тип `Animal`, но внутри находится объект `Dog`.
Поэтому будет вызван метод класса `Dog`:

```
Woof!
```

Можно работать сразу с несколькими животными:

```
Animal[] animals = {
    new Dog(),
    new Cat()
};

for (Animal animal : animals) {
    animal.sound();
}
```

Результат:

```
Woof!
Meow!
```

Один и тот же вызов

```
animal.sound();
```

может выполнять разные действия в зависимости от того, какой объект находится в переменной.

5. Абстракция

Абстракция помогает сосредоточиться на том, **что** объект умеет делать, не вдаваясь в детали того, **как** он это делает.
В Java для этого часто используют интерфейсы.
Например:

```
public interface Switchable {

    void turnOn();

    void turnOff();
}
```

Интерфейс говорит:

- объект можно включить;
- объект можно выключить.

Но он не объясняет, как именно это происходит.
Например, лампа может реализовать этот интерфейс:

```
public class Lamp implements Switchable {

    public void turnOn() {
        System.out.println("Lamp is on");
    }

    public void turnOff() {
        System.out.println("Lamp is off");
    }
}
```

Ключевое слово `implements` означает, что класс реализует интерфейс.

6. Композиция

Композиция означает, что один объект содержит или использует другой объект.
Это отношение:

«имеет».

Например, машина имеет двигатель.

```
public class Engine {

    public void start() {
        System.out.println("Engine started");
    }
}
```

Теперь используем двигатель внутри машины:

```
public class Car {

    private final Engine engine =
        new Engine();

    public void start() {
        engine.start();
    }
}
```

Класс `Car` не является двигателем.
Он **имеет** двигатель и использует его.

7. Наследование или композиция?

Полезно задать простой вопрос.
Объект является другим объектом?
Тогда возможно наследование.
Например:

Dog is an Animal.

Собака является животным.
Объект имеет другой объект?
Тогда обычно подходит композиция.
Например:

Car has an Engine.

Машина имеет двигатель.

Важно

Не стоит использовать наследование только для того, чтобы получить доступ к уже написанному коду.
Сначала проверьте, действительно ли между классами есть отношение «является».

8. Переопределение метода

Переопределение означает, что подкласс даёт свою реализацию метода, который уже был объявлен в родительском классе.
Например, в классе `Animal` есть:

```
public void sound() {
    System.out.println("Some sound");
}
```

А в классе `Dog`:

```
@Override
public void sound() {
    System.out.println("Woof!");
}
```

Аннотация

```
@Override
```

помогает компилятору проверить, что мы действительно переопределяем существующий метод.
Переопределение тесно связано с полиморфизмом.

9. Перегрузка метода

Перегрузка — это несколько методов с одинаковым именем, но разными параметрами.
Например:

```
public void print(String text) {
    System.out.println(text);
}

public void print(int number) {
    System.out.println(number);
}
```

Теперь можно написать:

```
print("Hello");
print(25);
```

В первом случае вызывается метод с параметром `String`, а во втором — с параметром `int`.

Не путайте

Переопределение — новая реализация унаследованного метода.
Перегрузка — одинаковое имя метода, но разные параметры.

10. Как выглядит хороший класс

Хороший класс обычно:

- решает одну понятную задачу;
- хранит только необходимые данные;
- не позволяет случайно испортить своё состояние;
- имеет понятные названия полей и методов;
- предоставляет действия, соответствующие смыслу объекта.

Например, для банковского счёта понятнее метод:

```
account.withdraw(100);
```

чем прямое изменение баланса:

```
account.balance = -100;
```

Первый вариант описывает действие «снять деньги» и позволяет классу проверить, можно ли его выполнить.

11. Четыре основных принципа ООП

Инкапсуляция
Защищает внутреннее состояние объекта и предоставляет безопасные способы работы с ним.
Наследование
Позволяет создавать новые типы на основе существующих, если между ними есть отношение «является».
Полиморфизм
Позволяет использовать разные объекты через общий тип.
Абстракция
Выделяет главное и скрывает ненужные для пользователя детали реализации.

Главное об ООП

Класс описывает объекты.
Объект хранит состояние и выполняет действия.
Инкапсуляция защищает состояние.
Наследование создаёт отношения между родственными типами.
Полиморфизм позволяет одному общему типу работать с разными реализациями.
Абстракция показывает важные возможности объекта и скрывает детали.
Композиция позволяет строить сложные объекты из более простых.

Памятка предназначена для первого знакомства с объектно-ориентированным программированием.
Примеры специально упрощены, чтобы показать основные идеи без лишних деталей.