Основи об’єктно-орієнтованого програмування

Java: ООП простими словами

Класи й об’єкти, інкапсуляція, успадкування, поліморфізм, абстракція та композиція

---

1. Клас і об’єкт

Клас — це опис того, які дані та дії матимуть об’єкти цього типу.
Об’єкт — конкретний екземпляр класу.
Наприклад, можна створити клас `Cat`:

```
public class Cat {
    String name;

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

А потім створити об’єкт:

```
Cat cat = new Cat();

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

Тут `Cat` — клас, а `cat` — об’єкт цього класу.

Головна ідея

Об’єкт зазвичай поєднує:

- дані — його **стан** ;
- методи — його **поведінку** .

2. Інкапсуляція

Інкапсуляція означає, що внутрішній стан об’єкта захищений від неправильного змінювання.
Розглянемо гравця, який має запас здоров’я:

```
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` оголошено як `private`. Тому інший код не може довільно записати в нього будь-яке значення.
Зміна виконується через зрозумілий метод:

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

А такий код недоступний:

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

Інкапсуляція допомагає об’єкту зберігати коректний стан.

Важливо

Інкапсуляція — це не просто `private`.
Головна ідея — дозволяти змінювати стан лише безпечним і зрозумілим способом.

3. Успадкування

Успадкування дозволяє створити новий клас на основі вже наявного. Зазвичай це відношення:

«є різновидом».

Наприклад, собака й кіт є тваринами.

```
public class Animal {

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

Клас `Dog` успадковується від `Animal`:

```
public class Dog extends Animal {

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

А клас `Cat` може реалізувати той самий метод по-своєму:

```
public class Cat extends Animal {

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

Ключове слово `extends` означає успадкування.

4. Поліморфізм

Поліморфізм дозволяє працювати з різними об’єктами через спільний тип.
Наприклад:

```
Animal animal = new Dog();

animal.sound();
```

Змінна має тип `Animal`, але всередині знаходиться об’єкт `Dog`.
Тому буде викликано метод класу `Dog`:

```
Woof!
```

Можна працювати одразу з кількома тваринами:

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

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

Результат:

```
Woof!
Meow!
```

Один і той самий виклик

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

може виконувати різні дії залежно від того, який саме об’єкт міститься у змінній.

5. Абстракція

Абстракція допомагає зосередитися на тому, **що** об’єкт уміє робити, не заглиблюючись у подробиці того, **як** він це робить.
У Java для цього часто використовують інтерфейси.
Наприклад:

```
public interface Switchable {

    void turnOn();

    void turnOff();
}
```

Інтерфейс говорить:

- об’єкт можна ввімкнути;
- об’єкт можна вимкнути.

Але він не пояснює, як саме це відбувається.
Наприклад, лампа може реалізувати цей інтерфейс:

```
public class Lamp implements Switchable {

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

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

Ключове слово `implements` означає, що клас реалізує інтерфейс.

6. Композиція

Композиція означає, що один об’єкт містить або використовує інший об’єкт.
Це відношення:

«має».

Наприклад, автомобіль має двигун.

```
public class Engine {

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

Тепер використаємо двигун усередині автомобіля:

```
public class Car {

    private final Engine engine =
        new Engine();

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

Клас `Car` не є двигуном.
Він **має** двигун і використовує його.

7. Успадкування чи композиція?

Корисно поставити просте запитання.
Об’єкт є іншим об’єктом?
Тоді можливе успадкування.
Наприклад:

Dog is an Animal.

Собака є твариною.
Об’єкт має інший об’єкт?
Тоді зазвичай підходить композиція.
Наприклад:

Car has an Engine.

Автомобіль має двигун.

Важливо

Не варто використовувати успадкування лише для того, щоб отримати доступ до вже написаного коду.
Спочатку перевірте, чи справді між класами є відношення «є різновидом».

8. Перевизначення методу

Перевизначення означає, що підклас надає власну реалізацію методу, який уже був оголошений у батьківському класі.
Наприклад, у класі `Animal` є:

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

А в класі `Dog`:

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

Анотація

```
@Override
```

допомагає компілятору перевірити, що ми дійсно перевизначаємо наявний метод.
Перевизначення тісно пов’язане з поліморфізмом.

9. Перевантаження методу

Перевантаження — це кілька методів з однаковим ім’ям, але різними параметрами.
Наприклад:

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

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

Тепер можна написати:

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

У першому випадку викликається метод з параметром `String`, а в другому — з параметром `int`.

Не плутайте

Перевизначення — нова реалізація успадкованого методу.
Перевантаження — однакове ім’я методу, але різні параметри.

10. Який вигляд має хороший клас

Хороший клас зазвичай:

- розв’язує одне зрозуміле завдання;
- зберігає лише необхідні дані;
- не дозволяє випадково зіпсувати свій стан;
- має зрозумілі назви полів і методів;
- надає дії, які відповідають змісту об’єкта.

Наприклад, для банківського рахунку зрозумілішим є метод:

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

ніж пряме змінювання балансу:

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

Перший варіант описує дію «зняти гроші» та дозволяє класу перевірити, чи можна її виконати.

11. Чотири основні принципи ООП

Інкапсуляція
Захищає внутрішній стан об’єкта та надає безпечні способи роботи з ним.
Успадкування
Дозволяє створювати нові типи на основі наявних, якщо між ними є відношення «є різновидом».
Поліморфізм
Дозволяє використовувати різні об’єкти через спільний тип.
Абстракція
Виділяє головне та приховує непотрібні для користувача деталі реалізації.

Головне про ООП

Клас описує об’єкти.
Об’єкт зберігає стан і виконує дії.
Інкапсуляція захищає стан.
Успадкування створює зв’язки між спорідненими типами.
Поліморфізм дозволяє одному спільному типу працювати з різними реалізаціями.
Абстракція показує важливі можливості об’єкта та приховує деталі.
Композиція дозволяє будувати складні об’єкти з простіших.

Пам’ятка призначена для першого знайомства з об’єктно-орієнтованим програмуванням. Приклади спеціально спрощено, щоб показати основні ідеї без зайвих деталей.