Java: класи та інтерфейси

Основні конструкції простими словами

Поля, методи, конструктори, модифікатори доступу, інтерфейси, абстрактні класи, record та enum

1. З чого складається клас?

Клас може містити:

- поля — дані об’єкта;
- конструктори — створення об’єкта;
- методи — дії об’єкта.

Простий приклад:

```
public class Student {

    private String name;
    private int age;

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

    public void introduce() {
        System.out.println(
            "My name is " + name
        );
    }
}
```

Тут:

- `name` і `age` — поля;
- `Student(...)` — конструктор;
- `introduce()` — метод.

Головна ідея

Клас описує, які дані зберігає об’єкт і які дії він уміє виконувати.

2. Створення об’єкта

Об’єкт створюється за допомогою ключового слова `new`:

```
Student student =
    new Student("Anna", 14);
```

Тут:

```
Student
```

— тип змінної,

```
student
```

— ім’я змінної,
а

```
new Student("Anna", 14)
```

створює новий об’єкт.
Після цього можна викликати його метод:

```
student.introduce();
```

3. Конструктор

Конструктор викликається під час створення нового об’єкта.
Його ім’я збігається з ім’ям класу:

```
public class Book {

    private String title;

    public Book(String title) {
        this.title = title;
    }
}
```

Конструктор не має типу значення, що повертається.
Тобто ми не пишемо:

```
// Неправильно
public void Book(String title) {
}
```

Такий код уже оголошує звичайний метод, а не конструктор.
Слово `this` означає **поточний об’єкт**:

```
this.title = title;
```

Ліворуч знаходиться поле об’єкта, а праворуч — параметр конструктора.

4. Кілька конструкторів

Клас може мати кілька конструкторів з різними параметрами.

```
public class User {

    private String name;
    private boolean active;

    public User(String name) {
        this(name, true);
    }

    public User(
            String name,
            boolean active) {

        this.name = name;
        this.active = active;
```

```
    }
}
```

Виклик

```
this(name, true);
```

передає роботу іншому конструктору цього самого класу.
Такий виклик має бути першою командою в конструкторі.

5. Модифікатори доступу

Модифікатори доступу визначають, звідки можна звертатися до полів і методів.

	Свій клас	Пакет	Підклас	Усюди
`private`	так	ні	ні	ні
`без модиф.`	так	так	лише в тому самому пакеті	ні
`protected`	так	так	так	ні
`public`	так	так	так	так

Для початку корисно запам’ятати два основні варіанти:
private
використовується для внутрішнього стану класу.
public
використовується для операцій, які мають бути доступними іншому коду.
Наприклад:

```
public class Player {

    private int health;

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

6. static

Ключове слово **static** означає, що поле або метод належить самому класу, а не конкретному об’єкту.
Наприклад:

```
public class MathHelper {

    public static int square(int x) {
        return x * x;
    }
}
```

Метод можна викликати без створення об’єкта:

```
int result = MathHelper.square(5);
```

Результат:

```
25
```

Важливо

Звичайний метод викликається в об’єкта.

```
student.introduce();
```

Статичний метод зазвичай викликається через ім’я класу.

```
MathHelper.square(5);
```

7. final

Ключове слово **final** має кілька значень залежно від місця використання.
final-поле
Після присвоєння посилання або значення не можна замінити:

```
private final String name;
```

final-метод
Не можна перевизначити в підкласі:

```
public final void print() {
    System.out.println("Hello");
}
```

final-клас
Від нього не можна успадковуватися:

```
public final class Calculator {
}
```

Для констант часто використовують одразу `static final`:

```
public static final int MAX_SIZE = 100;
```

8. Що таке інтерфейс?

Інтерфейс задає контракт:

«що об’єкт повинен уміти робити».

Наприклад:

```
public interface Printable {

    void print();
}
```

Інтерфейс говорить, що об’єкт повинен мати метод `print()`.
Тепер клас може реалізувати цей інтерфейс:

```
public class Document
        implements Printable {

    @Override
    public void print() {
        System.out.println(
            "Printing document"
```

```
        );
    }
}
```

Ключове слово

```
implements
```

означає, що клас реалізує інтерфейс.

9. Навіщо потрібні інтерфейси?

Інтерфейс дозволяє працювати з різними об’єктами однаковим способом.
Наприклад:

```
public class Photo
        implements Printable {

    @Override
    public void print() {
        System.out.println(
            "Printing photo"
        );
    }
}
```

Тепер і `Document`, і `Photo` можна використовувати через спільний тип:

```
Printable item = new Photo();

item.print();
```

Можна створити масив різних об’єктів:

```
Printable[] items = {
    new Document(),
    new Photo()
};

for (Printable item : items) {
    item.print();
}
```

Саме тому інтерфейси добре поєднуються з поліморфізмом.

10. Клас може реалізувати кілька інтерфейсів

Java не дозволяє класу успадковуватися одночасно від кількох класів.
Але один клас може реалізувати кілька інтерфейсів.
Наприклад:

```
public interface Printable {
    void print();
}

public interface Savable {
    void save();
}
```

Один клас може реалізувати обидва:

```
public class Document
        implements Printable, Savable {

    @Override
    public void print() {
        System.out.println("Print");
    }

    @Override
    public void save() {
        System.out.println("Save");
    }
}
```

Об’єкт `Document` тепер виконує одразу дві ролі:

- його можна друкувати;
- його можна зберігати.

11. Абстрактний клас

Абстрактний клас — це клас, об’єкт якого не можна створити безпосередньо. Він може містити:

- поля;
- конструктори;
- звичайні методи;
- абстрактні методи.

Наприклад:

```
public abstract class Animal {

    private final String name;

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

    public String getName() {
        return name;
    }

    public abstract void sound();
}
```

Метод

```
public abstract void sound();
```

не має реалізації.
Її має надати підклас:

```
public class Dog extends Animal {

    public Dog(String name) {
        super(name);
    }
```

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

Виклик

```
super(name);
```

звертається до конструктора батьківського класу.

12. Абстрактний клас чи інтерфейс?

Спрощене правило:
Абстрактний клас
підходить, коли споріднені класи мають спільні дані та спільну реалізацію.
Наприклад:

Dog is an Animal.

Інтерфейс
підходить, коли потрібно описати можливість або роль об’єкта.
Наприклад:

Document is Printable.

	Абстрактний клас	Інтерфейс
Поля об’єкта	так	зазвичай ні
Конструктор	так	ні
Готові методи	так	так, через `default`
Скільки можна використати	успадковується один клас	можна реалізувати кілька
Основна ідея	спільна основа	спільний контракт

13. default-методи інтерфейсу

Інтерфейс може містити метод з готовою реалізацією.
Для цього використовується слово `default`:

```
public interface Greeting {

    void sayHello();

    default void sayBye() {
        System.out.println("Bye!");
    }
}
```

Клас повинен реалізувати:

```
sayHello()
```

але вже отримує готовий метод:

```
sayBye()
```

Важливо

Для першого знайомства достатньо пам’ятати:
звичайний метод інтерфейсу задає обов’язковий контракт, а `default`-метод уже містить готову поведінку.

14. record

record зручний для невеликих об’єктів, головне завдання яких — зберігати дані.
Наприклад:

```
public record Student(
    long id,
    String name
) {
}
```

Створення об’єкта:

```
Student student =
    new Student(1, "Anna");
```

Отримати дані:

```
long id = student.id();
String name = student.name();
```

Java автоматично створює для `record`:

- конструктор;
- методи доступу;
- `equals()`;
- `hashCode()`;
- `toString()`.

Важливо

Звичайний клас добре підходить для об’єктів із поведінкою та змінюваним станом.
`record` особливо зручний для простих об’єктів-значень.

15. enum

enum використовується, коли існує заздалегідь відомий обмежений набір варіантів.
Наприклад, дні тижня:

```
public enum Day {
    MONDAY,
    TUESDAY,
    WEDNESDAY,
    THURSDAY,
    FRIDAY,
    SATURDAY,
    SUNDAY
```

```
}
```

Або стан замовлення:

```
public enum OrderStatus {
    NEW,
    PAID,
    SHIPPED,
    CANCELLED
}
```

Використання:

```
OrderStatus status =
    OrderStatus.PAID;
```

Перевага enum полягає в тому, що значення можна вибрати лише з дозволених варіантів.

16. Generics: тип у кутових дужках

Generics дозволяють класу або інтерфейсу працювати з різними типами даних. Ми вже зустрічали їх у колекціях:

```
List<String>
List<Integer>
```

Можна створити й власний клас:

```
public class Box<T> {

    private final T value;

    public Box(T value) {
        this.value = value;
    }

    public T getValue() {
        return value;
    }
}
```

Тепер один і той самий клас можна використовувати для різних типів:

```
Box<String> textBox =
    new Box<>("Hello");

Box<Integer> numberBox =
    new Box<>(42);
```

Літера T означає тип, який буде вказано пізніше.

Важливо

Generics дозволяють писати один спільний клас, зберігаючи перевірку типів компілятором.

17. sealed-типи

Це сучасніша можливість Java.
Ключове слово **sealed** дозволяє явно обмежити, які класи можуть реалізувати інтерфейс або успадковуватися від класу.
Наприклад:

```
public sealed interface Result
        permits Success, Failure {
}
```

Дозволені лише вказані варіанти:

```
public final class Success
        implements Result {
}

public final class Failure
        implements Result {
}
```

Це зручно, коли набір можливих типів заздалегідь відомий і обмежений.
Наприклад:

$$\text{Result} \longrightarrow \begin{cases} \text{Success}, \\ \text{Failure}. \end{cases}$$

Для перших програм `sealed` не є обов'язковою темою, але корисно знати, що така можливість у сучасній Java існує.

18. equals(), hashCode() і toString()

Усі класи Java успадковують основні методи від класу `Object`.
Серед них:
equals()
порівнює об'єкти.
hashCode()
повертає хеш-код об'єкта.
toString()
створює текстове представлення об'єкта.
Наприклад:

```
System.out.println(student);
```

Для `record` ці методи автоматично створюються на основі його компонентів.
Для звичайних класів їх іноді потрібно перевизначати самостійно.

19. Поширені помилки

Помилка 1.
Робити всі поля `public`:

```
public int health;
```

Зазвичай стан об'єкта краще захищати за допомогою `private`.

Помилка 2.
Плутати конструктор і метод.
Конструктор:

```
public Student(String name) {
}
```

Метод:

```
public void print() {
}
```

Помилка 3.
Намагатися створити об’єкт абстрактного класу:

```
// Не можна
Animal animal = new Animal("Bob");
```

Помилка 4.
Оголосити реалізацію інтерфейсу, але забути реалізувати його обов’язковий метод.

Помилка 5.
Використовувати успадкування лише тому, що хочеться повторно використати код.
Успадкування має відображати реальне відношення між типами.

20. Коротка пам’ятка

class
Описує стан і поведінку об’єктів.
constructor
Створює та готує об’єкт до роботи.
private
Приховує частину класу від зовнішнього коду.
public
Робить поле, метод або клас доступним іншому коду.
static
Належить класу, а не окремому об’єкту.
final
Обмежує подальшу зміну або успадкування.
interface
Задає спільний контракт або роль.
abstract class
Створює спільну основу для споріднених класів.
record
Зручно зберігає набір пов’язаних значень.
enum
Описує фіксований набір варіантів.
generics
Дозволяють одному коду безпечно працювати з різними типами.

Головний принцип

Використовуйте **клас**, коли об’єкт має зберігати стан і виконувати дії.
Використовуйте **інтерфейс**, коли потрібно описати спільну можливість або роль:

«що об’єкт уміє робити».

Використовуйте **абстрактний клас**, коли споріднені класи мають спільну основу.
Використовуйте **record** для простих об’єктів-значень.
Використовуйте **enum**, коли набір допустимих варіантів відомий заздалегідь.
Головне — не використовувати конструкцію лише тому, що вона складніша або сучасніша. Обирайте ту, яка найзрозуміліше описує задачу.

Пам’ятка призначена для першого знайомства з класами та інтерфейсами Java. Приклади спеціально спрощено, щоб показати основні ідеї без зайвих деталей.