Java: классы и интерфейсы

Основные конструкции простыми словами

Поля, методы, конструкторы, модификаторы доступа, интерфейсы, абстрактные классы, record и enum

1. Из чего состоит класс?

Класс может содержать:

- поля — данные объекта;
- конструкторы — создание объекта;
- методы — действия объекта.

Простой пример:

```
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
        );
    }
}
```

Здесь:

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

Главная идея

Класс описывает, какие данные хранит объект и какие действия он умеет выполнять.

2. Создание объекта

Объект создаётся с помощью ключевого слова `new`:

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

Здесь:

```
Student
```

— тип переменной,

```
student
```

— имя переменной,
а

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

создаёт новый объект.
После этого можно вызвать его метод:

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

3. Конструктор

Конструктор вызывается при создании нового объекта.
Его имя совпадает с именем класса:

```
public class Book {

    private String title;

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

У конструктора нет возвращаемого типа.
То есть мы не пишем:

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

Такой код уже объявляет обычный метод, а не конструктор.
Слово `this` означает **текущий объект**:

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

Слева находится поле объекта, а справа — параметр конструктора.

4. Несколько конструкторов

У класса может быть несколько конструкторов с разными 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;
```

```
    }
}
```

Вызов

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

передаёт работу другому конструктору этого же класса.
Такой вызов должен быть первой командой в конструкторе.

5. Модификаторы доступа

Модификаторы доступа определяют, откуда можно обращаться к полям и методам.

	Свой класс	Пакет	Подкласс	Везде
`private`	да	нет	нет	нет
`без модиф.`	да	да	только в том же пакете	нет
`protected`	да	да	да	нет
`public`	да	да	да	да

Для начала полезно запомнить два основных варианта:
private
используется для внутреннего состояния класса.
public
используется для операций, которые должны быть доступны другому коду.
Например:

```
public class Player {

    private int health;

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

6. static

Ключевое слово **static** означает, что поле или метод относится к самому классу, а не к конкретному объекту.
Например:

```
public class MathHelper {

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

Метод можно вызвать без создания объекта:

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

Результат:

```
25
```

Важно

Обычный метод вызывается у объекта.

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

Статический метод обычно вызывается через имя класса.

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

7. final

Ключевое слово **final** имеет несколько значений в зависимости от места использования.

final-поле

После присваивания ссылку или значение нельзя заменить:

```
private final String name;
```

final-метод

Нельзя переопределить в подклассе:

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

final-класс

От него нельзя наследоваться:

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

Для констант часто используют сразу `static final`:

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

8. Что такое интерфейс?

Интерфейс задаёт контракт:

«что объект должен уметь делать».

Например:

```
public interface Printable {

    void print();
}
```

Интерфейс говорит, что объект должен иметь метод `print()`.
Теперь класс может реализовать этот интерфейс:

```
public class Document
        implements Printable {
```

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

Ключевое слово

```
implements
```

означает, что класс реализует интерфейс.

9. Зачем нужны интерфейсы?

Интерфейс позволяет работать с разными объектами одинаковым способом.
Например:

```
public class Photo
        implements Printable {

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

Теперь и `Document`, и `Photo` можно использовать через общий тип:

```
Printable item = new Photo();

item.print();
```

Можно создать массив разных объектов:

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

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

Именно поэтому интерфейсы хорошо сочетаются с полиморфизмом.

10. Класс может реализовать несколько интерфейсов

Java не позволяет классу наследоваться сразу от нескольких классов.
Но один класс может реализовать несколько интерфейсов.
Например:

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

public interface Savable {
```

```
    void save();
}
```

Один класс может реализовать оба:

```
public class Document
        implements Printable, Savable {

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

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

Объект `Document` теперь выполняет сразу две роли:

- его можно печатать;
- его можно сохранять.

11. Абстрактный класс

Абстрактный класс — класс, объект которого нельзя создать напрямую.
Он может содержать:

- поля;
- конструкторы;
- обычные методы;
- абстрактные методы.

Например:

```
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 имеет реализации.
Её должен предоставить подкласс:

```
public class Dog extends Animal {
```

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

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

Вызов

```
super(name);
```

обращается к конструктору родительского класса.

12. Абстрактный класс или интерфейс?

Упрощённое правило:
Абстрактный класс
подходит, когда родственные классы имеют общие данные и общую реализацию.
Например:

Dog is an Animal.

Интерфейс
подходит, когда нужно описать возможность или роль объекта.
Например:

Document is Printable.

	Абстрактный класс	Интерфейс
Поля объекта	да	обычно нет
Конструктор	да	нет
Готовые методы	да	да, через `default`
Сколько можно использовать	наследуется один класс	можно реализовать несколько
Основная идея	общая база	общий контракт

13. default-методы интерфейса

Интерфейс может содержать метод с готовой реализацией.
Для этого используется слово `default`:

```
public interface Greeting {

    void sayHello();

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

Класс обязан реализовать:

```
sayHello()
```

но уже получает готовый метод:

```
sayBye()
```

Важно

Для первого знакомства достаточно помнить:
обычный метод интерфейса задаёт обязательный контракт, а `default`-метод уже содержит готовое поведение.

14. record

record удобен для небольших объектов, главная задача которых — хранить данные.
Например:

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

Создание объекта:

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

Получить данные:

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

Java автоматически создаёт для `record`:

- конструктор;
- методы доступа;
- `equals()`;
- `hashCode()`;
- `toString()`.

Важно

Обычный класс хорошо подходит для объектов с поведением и изменяемым состоянием.
`record` особенно удобен для простых объектов-значений.

15. enum

enum используется, когда существует заранее известный ограниченный набор вариантов.
Например, дни недели:

```
public enum Day {
```

```
    MONDAY,
    TUESDAY,
    WEDNESDAY,
    THURSDAY,
    FRIDAY,
    SATURDAY,
    SUNDAY
}
```

Или состояние заказа:

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

Использование:

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

Преимущество `enum` состоит в том, что значение можно выбрать только из разрешённых вариантов.

16. Generics: тип в угловых скобках

Generics позволяют классу или интерфейсу работать с разными типами данных. Мы уже встречали их в коллекциях:

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

Можно создать и собственный класс:

```
public class Box<T> {

    private final T value;

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

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

Теперь один и тот же класс можно использовать для разных типов:

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

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

Буква `T` означает тип, который будет указан позже.

Важно

Generics позволяют писать один общий класс, сохраняя проверку типов компилятором.

17. sealed-типы

Это более современная возможность Java.
Ключевое слово **sealed** позволяет явно ограничить, какие классы могут реализовать интерфейс или наследоваться от класса.
Например:

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

Разрешены только указанные варианты:

```
public final class Success
        implements Result {
}

public final class Failure
        implements Result {
}
```

Это удобно, когда набор возможных типов заранее известен и ограничен.
Например:

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

Для первых программ `sealed` не является обязательной темой, но полезно знать, что такая возможность в современной Java существует.

18. equals(), hashCode() и toString()

Все классы Java наследуют основные методы от класса `Object`.
Среди них:
equals()
сравнивает объекты.
hashCode()
возвращает хеш-код объекта.
toString()
создаёт текстовое представление объекта.
Например:

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

Для `record` эти методы автоматически создаются на основе его компонентов.
Для обычных классов их иногда нужно переопределять самостоятельно.

19. Частые ошибки

Ошибка 1.
Делать все поля `public`:

```
public int health;
```

Обычно состояние объекта лучше защищать с помощью `private`.

Ошибка 2.
Путать конструктор и метод.
Конструктор:

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

Метод:

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

Ошибка 3.
Пытаться создать объект абстрактного класса:

```
// Нельзя
Animal animal = new Animal("Bob");
```

Ошибка 4.
Объявить реализацию интерфейса, но забыть реализовать его обязательный метод.

Ошибка 5.
Использовать наследование только потому, что хочется повторно использовать код.
Наследование должно отражать реальное отношение между типами.

20. Короткая памятка

class
Описывает состояние и поведение объектов.
constructor
Создаёт и подготавливает объект.
private
Скрывает часть класса от внешнего кода.
public
Делает поле, метод или класс доступным другому коду.
static
Относится к классу, а не к отдельному объекту.
final
Ограничивает дальнейшее изменение или наследование.
interface
Задаёт общий контракт или роль.
abstract class
Создаёт общую базу для родственных классов.
record
Удобно хранит набор связанных значений.
enum
Описывает фиксированный набор вариантов.

generics
Позволяют одному коду работать с разными типами безопасным способом.

Главный принцип

Используйте **класс**, когда объект должен хранить состояние и выполнять действия.
Используйте **интерфейс**, когда нужно описать общую возможность или роль:

«что объект умеет делать».

Используйте **абстрактный класс**, когда родственные классы имеют общую основу.
Используйте **record** для простых объектов-значений.
Используйте **enum**, когда набор допустимых вариантов заранее известен.
Главное — не использовать конструкцию потому, что она сложнее или современнее. Выбирайте ту, которая яснее всего описывает задачу.

Памятка предназначена для первого знакомства с классами и интерфейсами Java. Примеры специально упрощены, чтобы показать основные идеи без лишних деталей.