Java Учебник — Подсказки с кодом и объяснениями

Содержит примеры по всем изученным темам: синтаксис, ООП (части 1–3).

# Оглавление

## Синтаксис

* • Variables.java
* • Conditionals.java
* • Loops.java

## ООП часть 1

* • PersonDemo.java
* • InheritanceDemo.java
* • PolymorphismDemo.java
* • ConstructorDemo.java

## ООП часть 2

* • ObjectClassDemo.java
* • StaticDemo.java
* • EnumDemo.java
* • AbstractClassDemo.java
* • InterfaceDemo.java

## ООП часть 3

* • RecordDemo.java
* • SealedDemo.java
* • NestedClasses.java
* • AnonymousClassDemo.java
* • LocalClassDemo.java

# Синтаксис

## Variables.java

public class Variables {  
 public static void main(String[] args) {  
 int age = 25;  
 double price = 19.99;  
 boolean isStudent = true;  
 char grade = 'A';  
 String name = "Иван";  
  
 System.out.println("Возраст: " + age);  
 System.out.println("Цена: " + price);  
 System.out.println("Студент? " + isStudent);  
 System.out.println("Оценка: " + grade);  
 System.out.println("Имя: " + name);  
 }  
}

Этот файл демонстрирует переменные разных типов (int, double, boolean, char, String). Переменные — это основа программирования: они хранят данные, которые мы используем в программах. На практике переменные нужны для всего — от подсчёта денег до хранения текста сообщений.

## Conditionals.java

public class Conditionals {  
 public static void main(String[] args) {  
 int number = 10;  
  
 if (number > 0) {  
 System.out.println("Число положительное");  
 }  
  
 if (number % 2 == 0) {  
 System.out.println("Число чётное");  
 } else {  
 System.out.println("Число нечётное");  
 }  
  
 if (number < 0) {  
 System.out.println("Число отрицательное");  
 } else if (number == 0) {  
 System.out.println("Число равно нулю");  
 } else {  
 System.out.println("Число положительное");  
 }  
  
 int day = 3;  
 switch (day) {  
 case 1 -> System.out.println("Понедельник");  
 case 2 -> System.out.println("Вторник");  
 case 3 -> System.out.println("Среда");  
 default -> System.out.println("Другой день");  
 }  
 }  
}

Файл демонстрирует условные операторы if, if-else, if-else if и switch. С их помощью программа принимает решения. Применяется, например, при проверке пароля, определении выходных или будних дней.

## Loops.java

public class Loops {  
 public static void main(String[] args) {  
 for (int i = 1; i <= 5; i++) {  
 System.out.println("for: i = " + i);  
 }  
  
 int j = 1;  
 while (j <= 5) {  
 System.out.println("while: j = " + j);  
 j++;  
 }  
  
 int k = 1;  
 do {  
 System.out.println("do-while: k = " + k);  
 k++;  
 } while (k <= 5);  
  
 int[] numbers = {10, 20, 30, 40, 50};  
 for (int num : numbers) {  
 System.out.println("foreach: num = " + num);  
 }  
 }  
}

Файл демонстрирует циклы: for, while, do-while, foreach. Циклы нужны для многократного выполнения кода, например перебора массива товаров или сообщений в чате.

# ООП часть 1

## PersonDemo.java

class Person {  
 private String name;  
 private int age;  
  
 public Person(String name, int age) {  
 this.name = name;  
 this.age = age;  
 }  
  
 public String getName() { return name; }  
 public int getAge() { return age; }  
  
 public void setAge(int age) { this.age = age; }  
  
 public void introduce() {  
 System.out.println("Меня зовут " + name + ", мне " + age + " лет.");  
 }  
}  
  
public class PersonDemo {  
 public static void main(String[] args) {  
 Person p = new Person("Иван", 25);  
 p.introduce();  
 p.setAge(26);  
 p.introduce();  
 }  
}

Пример показывает \*\*инкапсуляцию\*\*: поля `name` и `age` скрыты (private), доступ к ним осуществляется через методы-геттеры/сеттеры. Это нужно для контроля доступа и защиты данных. В реальных проектах так пишутся модели пользователей, товаров, заказов.

## InheritanceDemo.java

class Animal {  
 public void eat() {  
 System.out.println("Животное ест");  
 }  
}  
  
class Dog extends Animal {  
 public void bark() {  
 System.out.println("Собака лает");  
 }  
}  
  
public class InheritanceDemo {  
 public static void main(String[] args) {  
 Dog d = new Dog();  
 d.eat();  
 d.bark();  
 }  
}

Пример наследования: `Dog` расширяет `Animal`. Класс-наследник получает все методы и может добавлять новые. Используется для иерархий: например, `Транспорт` → `Автомобиль` → `Грузовик`.

## PolymorphismDemo.java

class Shape {  
 public void draw() {  
 System.out.println("Фигура рисуется");  
 }  
}  
  
class Circle extends Shape {  
 @Override  
 public void draw() {  
 System.out.println("Рисуется круг");  
 }  
}  
  
class Square extends Shape {  
 @Override  
 public void draw() {  
 System.out.println("Рисуется квадрат");  
 }  
}  
  
public class PolymorphismDemo {  
 public static void main(String[] args) {  
 Shape s1 = new Circle();  
 Shape s2 = new Square();  
  
 s1.draw();  
 s2.draw();  
 }  
}

Пример \*\*полиморфизма\*\*: переменная типа `Shape` может ссылаться на разные объекты (`Circle`, `Square`). Вызов `draw()` работает по-разному. Это позволяет писать гибкий код, например обработку разных документов или разных форматов файлов.

## ConstructorDemo.java

class Car {  
 private String brand;  
 private int year;  
  
 public Car(String brand, int year) {  
 this.brand = brand;  
 this.year = year;  
 }  
  
 public void info() {  
 System.out.println(brand + " выпуска " + year + " года");  
 }  
}  
  
public class ConstructorDemo {  
 public static void main(String[] args) {  
 Car c = new Car("Toyota", 2020);  
 c.info();  
 }  
}

Пример работы \*\*конструкторов\*\*. Конструктор задаёт начальные значения полей объекта. На практике используется при создании любых сущностей: заказов, пользователей, транзакций.

# ООП часть 2

## ObjectClassDemo.java

class Person {  
 private String name;  
 private int age;  
  
 public Person(String name, int age) {  
 this.name = name;  
 this.age = age;  
 }  
  
 @Override  
 public String toString() {  
 return "Person{name='" + name + "', age=" + age + "}";  
 }  
  
 @Override  
 public boolean equals(Object obj) {  
 if (this == obj) return true;  
 if (obj == null || getClass() != obj.getClass()) return false;  
 Person p = (Person) obj;  
 return age == p.age && name.equals(p.name);  
 }  
  
 @Override  
 public int hashCode() {  
 return name.hashCode() + age;  
 }  
}  
  
public class ObjectClassDemo {  
 public static void main(String[] args) {  
 Person p1 = new Person("Иван", 25);  
 Person p2 = new Person("Иван", 25);  
  
 System.out.println(p1);  
 System.out.println("p1 == p2? " + p1.equals(p2));  
 System.out.println("hashCode p1: " + p1.hashCode());  
 System.out.println("hashCode p2: " + p2.hashCode());  
 }  
}

Демонстрация методов `Object`: `toString`, `equals`, `hashCode`. Используется для сравнения объектов, вывода их в логах, работы с коллекциями.

## StaticDemo.java

class MathUtils {  
 public static final double PI = 3.14159;  
 public static int square(int x) {  
 return x \* x;  
 }  
}  
  
public class StaticDemo {  
 public static void main(String[] args) {  
 System.out.println("PI = " + MathUtils.PI);  
 System.out.println("Квадрат числа 5 = " + MathUtils.square(5));  
 }  
}

Пример использования `static`. Поля и методы `static` принадлежат классу, а не объекту. На практике: математические функции, утилитарные классы.

## EnumDemo.java

enum Day {  
 MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY  
}  
  
public class EnumDemo {  
 public static void main(String[] args) {  
 Day today = Day.FRIDAY;  
  
 switch (today) {  
 case MONDAY -> System.out.println("Начало недели");  
 case FRIDAY -> System.out.println("Пятница! Скоро выходные!");  
 case SATURDAY, SUNDAY -> System.out.println("Выходной день");  
 default -> System.out.println("Будний день");  
 }  
 }  
}

Пример перечислений (enum). Удобно для хранения фиксированных значений (дни недели, состояния заказа).

## AbstractClassDemo.java

abstract class Animal {  
 public abstract void makeSound();  
 public void sleep() {  
 System.out.println("Животное спит");  
 }  
}  
  
class Dog extends Animal {  
 @Override  
 public void makeSound() {  
 System.out.println("Гав-гав!");  
 }  
}  
  
public class AbstractClassDemo {  
 public static void main(String[] args) {  
 Dog d = new Dog();  
 d.makeSound();  
 d.sleep();  
 }  
}

Абстрактные классы задают общий шаблон. Методы могут быть абстрактными или с реализацией. Применяются в случаях, когда нужен базовый функционал, но детали уточняются в наследниках.

## InterfaceDemo.java

interface Flyable {  
 void fly();  
}  
  
class Bird implements Flyable {  
 @Override  
 public void fly() {  
 System.out.println("Птица летит!");  
 }  
}  
  
class Airplane implements Flyable {  
 @Override  
 public void fly() {  
 System.out.println("Самолёт взлетает!");  
 }  
}  
  
public class InterfaceDemo {  
 public static void main(String[] args) {  
 Flyable f1 = new Bird();  
 Flyable f2 = new Airplane();  
 f1.fly();  
 f2.fly();  
 }  
}

Интерфейсы описывают поведение (контракты). Классы реализуют их. Используются для задания общих правил, например: всё, что может 'летать', должно реализовать метод fly().

# ООП часть 3

## RecordDemo.java

record Person(String name, int age) {}  
  
public class RecordDemo {  
 public static void main(String[] args) {  
 Person p1 = new Person("Иван", 25);  
 System.out.println(p1.name());  
 System.out.println(p1.age());  
 System.out.println(p1);  
 }  
}

Record — это компактный способ создания неизменяемых классов данных. Идеально для DTO (передачи данных между слоями программы).

## SealedDemo.java

sealed class Shape permits Circle, Square {}  
  
final class Circle extends Shape {}  
final class Square extends Shape {}  
  
public class SealedDemo {  
 public static void main(String[] args) {  
 Shape s1 = new Circle();  
 Shape s2 = new Square();  
 System.out.println("Sealed классы ограничивают наследование");  
 }  
}

Sealed-классы ограничивают круг наследников. Используется для безопасности и предсказуемости иерархий.

## NestedClasses.java

class Outer {  
 private String message = "Привет из Outer!";  
  
 static class StaticNested {  
 public void show() {  
 System.out.println("Это статический вложенный класс");  
 }  
 }  
  
 class Inner {  
 public void show() {  
 System.out.println("Inner класс: " + message);  
 }  
 }  
}  
  
public class NestedClasses {  
 public static void main(String[] args) {  
 Outer.StaticNested sn = new Outer.StaticNested();  
 sn.show();  
 Outer outer = new Outer();  
 Outer.Inner in = outer.new Inner();  
 in.show();  
 }  
}

Вложенные классы: статические и inner. Inner имеет доступ к полям внешнего. Используются для организации кода и вспомогательных сущностей.

## AnonymousClassDemo.java

interface Greeting {  
 void sayHello();  
}  
  
public class AnonymousClassDemo {  
 public static void main(String[] args) {  
 Greeting g = new Greeting() {  
 @Override  
 public void sayHello() {  
 System.out.println("Привет из анонимного класса!");  
 }  
 };  
 g.sayHello();  
 }  
}

Анонимные классы создаются 'на лету'. Используются для одноразовых реализаций интерфейсов, например в обработчиках событий.

## LocalClassDemo.java

public class LocalClassDemo {  
 public void display() {  
 class Local {  
 public void show() {  
 System.out.println("Привет из локального класса!");  
 }  
 }  
 Local local = new Local();  
 local.show();  
 }  
  
 public static void main(String[] args) {  
 LocalClassDemo demo = new LocalClassDemo();  
 demo.display();  
 }  
}

Локальные классы определяются внутри методов. Используются, когда класс нужен только в ограниченном контексте метода.