Начало с 5 лаб.

**List**

**List** — это **список**, в котором можно хранить **много значений** (например, много объектов, строк, чисел и т.д.).

List<String> imiona = new ArrayList<>(); создаём список

imiona.add("Ola");

imiona.add("Jan");

for (String i : imiona) {

System.out.println(i);

}

Не забывай, что в основном у меня не просто String , а List< Person>.

**Set**

**Set — это множество (коллекция), где:**

* ❗ **нет повторяющихся элементов**
* ❗ **нет гарантии порядка** (как элементы будут располагаться — Java сама решает)

**List — это список, а Set — это множество.**

Set<String> imiona = new HashSet<>();

imiona.add("Ola");

imiona.add("Jan");

imiona.add("Ola"); // не добавится второй раз

Мы используем TreeSet, чтобы:

* дети автоматически хранились **в порядке по дате рождения**,
* избежать дубликатов,
* удобно находить младших/старших.

### Что такое Collections?

Collections — это **класс-помощник** в Java (java.util.Collections), который содержит **статические методы** для работы с коллекциями (List, Set, Map и др.).

**Примеры того, что можно делать с помощью Collections:**

* Сортировать списки.
* Найти минимум или максимум в коллекции.
* Перемешать элементы случайным образом.
* Сделать коллекцию неизменяемой (чтобы нельзя было менять).
* Копировать элементы из одной коллекции в другую.

## Чтобы использовать Collections, нужно:

1. **Импортировать его**:

import java.util.Collections;

2.

public class Person implements Comparable<Person>

"Я, класс Person, **умею сравнивать** себя с другим объектом Person."

3.

@Override

public int compareTo(Person other) {

return this.birthDate.compareTo(other.birthDate);

}

4. Используем Collections

// Метод для поиска самого младшего ребёнка  
public Person getYoungestChild() {  
 if (children.isEmpty()) {  
 return null;  
 }  
 return Collections.*max*(children); // Работает благодаря Comparable  
}

public List<Person> getChildren() {

List<Person> sortedChildren = new ArrayList<>(children); // копируем

Collections.sort(sortedChildren); // сортируем от старшего к младшему

return sortedChildren;

}

| **Что** | **Что делает** |
| --- | --- |
| implements Comparable<Person> | Позволяет сравнивать объекты Person. |
| compareTo() | Определяет правило сравнения — здесь по дате рождения. |
| Collections.max(children) | Выбирает самого младшего ребёнка, потому что младший — это "больше" по дате рождения. |

**Что такое Map?**

Map — это такая структура данных, которая хранит пару **ключ-значение**. Ты можешь думать о ней как о наборе **словарь**: ключом будет слово, а значением — то, что этому слову соответствует.

import java.util.HashMap;

import java.util.Map;

В данном случае:

* **Ключ** — это **полное имя** человека (например, "Anna Kowalska").
* **Значение** — это сам **объект Person**, который представляет человека.

Map<String, String> capitals = new HashMap<>();

Здесь ты создаешь объект типа Map для хранения людей. Он будет хранить ключи типа String (имя + фамилия) и значения типа Person (объект, представляющий человека).

1️⃣ put(key, value) — добавить или заменить

capitals.put("Spain", "Madrid"); // добавляет новую пару

capitals.put("Germany", "Bonn"); // заменяет старую столицу

2️⃣ get(key) — получить значение

System.out.println(capitals.get("France")); // Paris

System.out.println(capitals.get("Italy")); // null (если ключа нет)

3️⃣ containsKey(key) — проверить наличие ключа

System.out.println(capitals.containsKey("Germany")); // true

System.out.println(capitals.containsKey("Italy")); // false

4️⃣ remove(key) — удалить по ключу

capitals.remove("Germany"); // теперь "Germany" нет в карте

System.out.println(capitals.get("Germany")); // null

5️⃣ keySet() — все ключи (страны)  
for (String country : capitals.keySet()) {

System.out.println("Country: " + country);

}

Что значат слова throws, extends, implements?

**🔹 throws — может выбросить исключение**

Это предупреждение, что метод **может выбросить ошибку**, и нужно быть готовым её обработать.

public static Person fromCsvLine(String line) throws NegativeLifespanException

(· static → метод или переменная **принадлежит классу**, не объекту.

· Можно вызывать **без создания объекта**.

· В твоём случае fromCsvLine(...) — идеально подходит как static.)

🔹 extends — наследование

public class NegativeLifespanException extends Exception;

То есть, extends = "унаследован от...".

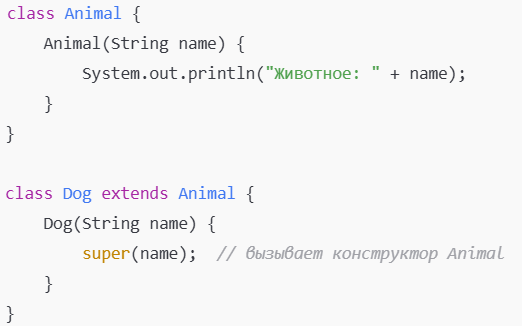
🔹 implements — реализация интерфейса

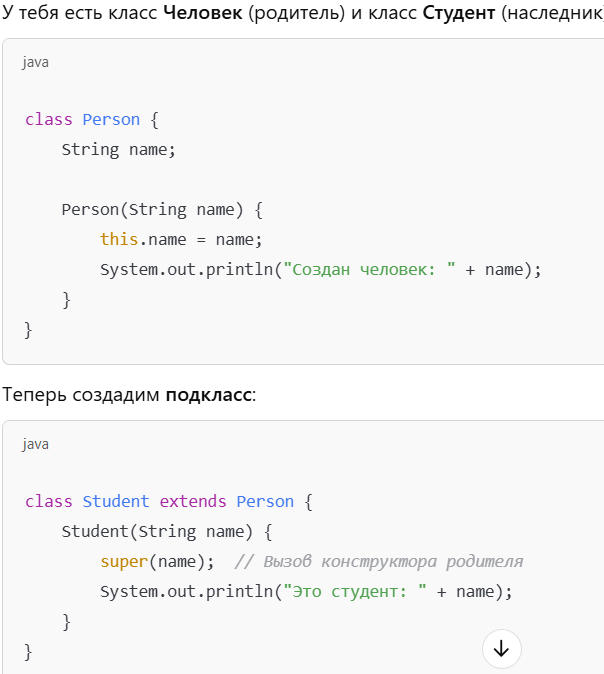
Это ты пока можешь просто запомнить как:

Класс **обещает реализовать** определённые методы.

### SUPER

**super** — это ключевое слово в Java. Оно используется, чтобы обратиться к **родительскому классу** (от которого ты наследуешься).

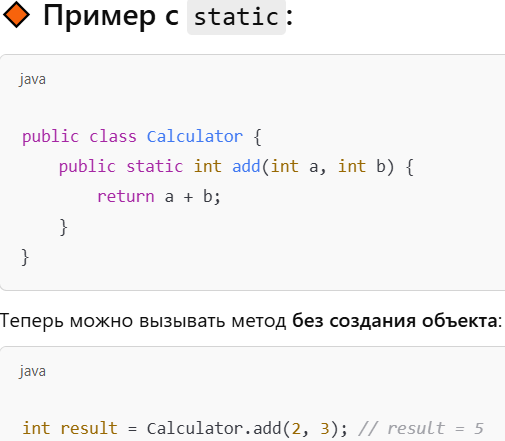




🔹 Что такое static?

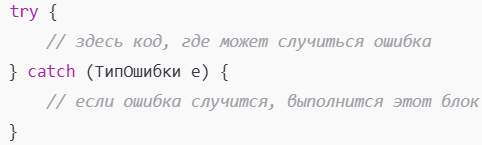
Слово static в Java означает, что **что-то принадлежит классу, а не объекту**.





🧠 Что такое try-catch?

Это способ **ловить ошибки**, чтобы программа **не упала**, а **выполнила запасной план** (например, показала сообщение пользователю).



🔹 Используй protected, **если подклассы должны видеть поле/метод**.  
🔹 Используй private, **если хочешь скрыть полностью и контролировать через get/set**.

✅ Что такое LocalTime, LocalTime.now()?

🔹 LocalTime.now() — возвращает **текущее локальное время** (например, 14:23:10).  
🔹 Методы:

* getHour() → часы (0–23)
* getMinute() → минуты (0–59)
* getSecond() → секунды (0–59)

**📘 Пример:**

**LocalTime now = LocalTime.now();**

**int h = now.getHour(); // например, 14**

**int m = now.getMinute(); // например, 23**

**int s = now.getSecond(); // например, 10**

✅ Как работает String.format

String.format — это способ **вставить значения в строку по шаблону**.

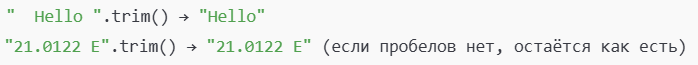
String s = String.format("Привет, %s! Тебе %d лет.", "Аня", 20);

System.out.println(s); // Привет, Аня! Тебе 20 лет.

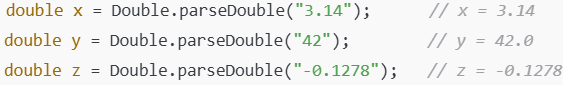
### 🔧 Что делает trim()?

Метод trim() убирает пробелы **в начале и в конце** строки.

Примеры:



🔹 **Double.parseDouble(...)** — это **стандартный способ в Java** преобразовать строку в число с плавающей точкой (double).



int timezone = Integer.parseInt(parts[1].trim());

if (direction.equals("S") || direction.equals("W")) {  
 value = -value;  
}

Важно запомнить equals

**List<String> lines = Files.readAllLines(Paths.get(filePath));**

Paths.get(filePath) — превращает путь "cities.csv" (или любой другой, переданный в filePath) в Path-объект.

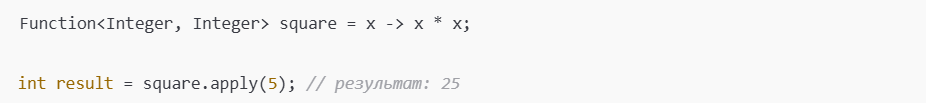
Files.readAllLines(...) — читает **все строки файла** по указанному пути.

Function<T, R>





🔶Метод apply(...) **запускает функцию**, передавая в неё аргумент.



stream()

List<Integer> numbers = new ArrayList<>();

List<Integer> doubledNumbers = numbers.stream() // создаём поток данных

.filter(number -> number % 2 == 0)

.map(number -> number \* 2) // умножаем каждое число на 2

.collect(Collectors.toList()); // собираем результат обратно в список

AbstractList<T>

AbstractList<T> — это абстрактный класс, который реализует интерфейс List<T>.