3. Java_MIDDLE (JavaStreamApi)

- Обработка коллекций

- Работа с библиотекой Stream

- Использование lambda выражений

- Интерфейсы Comparator (Comparable), Iterator (Iterable)

- Контракт hashCode() - equals()

- Метод reference()

Обработка коллекций (использование стримов):

```
public class Streams {
    public static void main(String[] args) throws IOException {
        List<Employee> emps = List.of(
                new Employee("Michael", "Smith", 243, 43, "CHEF"),
                new Employee("Jane", "Smith", 523, 40, "MANAGER"),
                new Employee("Jury", "Gagarin", 6423, 26, "MANAGER"),
                new Employee("Jack", "London", 5543, 53, "WORKER"),
                new Employee("Eric", "Jackson", 2534, 22, "WORKER")
        );

        // Создание стримов из объектов файловой системы
        Stream<String> lines = Files.lines(Paths.get("some.txt"));
        Stream<Path> list = Files.list(Paths.get("./"));
        Stream<Path> walk = Files.walk(Paths.get("./"), 3);

        // Создание из примитивов
        IntStream intStream = IntStream.of(1, 2, 3, 4);
        DoubleStream doubleStream = DoubleStream.of(1.2, 3.4);
        IntStream range = IntStream.range(10, 100); // 10 .. 99
        IntStream intStream1 = IntStream.rangeClosed(10, 100); // 10 .. 100

        int[] ints = {1, 2, 3, 4};
        IntStream stream = Arrays.stream(ints);

        Stream<String> stringStream = Stream.of("1", "2", "3");
        Stream<? extends Serializable> stream1 = Stream.of(1, "2", "3");

        // Stream builder
        Stream<String> build = Stream.<String>builder()
                .add("Mike")
                .add("joe")
                .build();

        // Стримы из объектов
        Stream<Employee> stream2 = emps.stream();
        Stream<Employee> employeeStream = emps.parallelStream();

        // Преобразования
        Stream<Integer> iterate = Stream.iterate(1950, val -> val + 3);
        Stream<String> concat = Stream.concat(stringStream, build);
        IntStream intStream2 = IntStream.of(100, 200, 300, 400);
        intStream.reduce((left, right) -> left + right).orElse(0);
        IntStream.of(100, 200, 300, 400).average();
        IntStream.of(100, 200, 300, 400).max();
        IntStream.of(100, 200, 300, 400).min();
        IntStream.of(100, 200, 300, 400).sum();
        IntStream.of(100, 200, 300, 400).summaryStatistics();

        // Операции над объектами
        Stream<Employee> empStream = emps.stream();
        empStream.count();
        emps.stream().forEach(employee ->
System.out.println(employee.getAge()));
        emps.forEach(employee -> System.out.println(employee.getAge()));
        emps.stream().forEachOrdered(employee ->
System.out.println(employee.getAge()));
        emps.stream().collect(Collectors.toList());
        emps.stream().toArray();
        Map<Integer, String> collect = emps.stream().collect(Collectors.toMap(
                Employee::getId,
                emp -> String.format("%s %s", emp.getLastName(),
emp.getFirstName())
        ));
        emps.stream().max(Comparator.comparingInt(Employee::getAge));
        emps.stream().findAny();
        emps.stream().findFirst();
        emps.stream().noneMatch(employee -> employee.getAge() > 60); // true
        emps.stream().anyMatch(employee -> employee.getRole() == "CHEF"); //
true
        emps.stream().allMatch(employee -> employee.getAge() > 18); // true

        // Трансформация
        LongStream longStream = IntStream.of(100, 200, 300,
400).mapToLong(Long::valueOf);
        IntStream.of(100, 200, 300, 400, 100, 200).distinct(); // 100, 200, 300,
400
        Stream<Employee> employeeStream2 = emps.stream().filter(employee ->
employee.getRole() != "CHEF");

        emps.stream()
                .skip(3)
                .limit(5);

        emps.stream()
                .sorted(Comparator.comparingInt(Employee::getAge))
                .peek(emp -> emp.setAge(18))
                .map(emp -> String.format("%s %s", emp.getLastName(),
emp.getFirstName()));

        emps.stream().takeWhile(employee -> employee.getAge() >
30).forEach(System.out::println);
        emps.stream().dropWhile(employee -> employee.getAge() >
30).forEach(System.out::println);

        IntStream.of(100, 200, 300, 400)
                .flatMap(value -> IntStream.of(value - 50, value))
                .forEach(System.out::println);

        Stream<Employee> empl = emps.stream()
                .filter(employee ->
                        employee.getAge() <= 30 && employee.getRole() !=
"WORKER"
                )
                .sorted(Comparator.comparing(Employee::getLastName));

        print(empl);

        Stream<Employee> sorted = emps.stream()
                .filter(employee -> employee.getAge() > 40)
                .sorted((o1, o2) -> o2.getAge() - o1.getAge())
                .limit(4);
        print(sorted);

        IntSummaryStatistics statistics = emps.stream()
                .mapToInt(Employee::getAge)
                .summaryStatistics();

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

Варианты сортировки:

```
public class SortColection {
    public static void main(String[] args) {
        List<String> list = new ArrayList<>();
        list.add("Hello");
        list.add("Hey");
        list.add("a");
        list.add("ab");

        // Переопределение метода compare()
        list.sort(new Comparator<String>() {
            @Override
            public int compare(String o1, String o2) {
                if (o1.length() > o2.length()) {
                    return 1;
                } else if (o1.length() < o2.length()) {
                    return -1;
                } else {
                    return 0;
                }
            }
        });

        // Описание паттерна сортировки лямбда функцией
        list.sort((s1, s2) -> {
            if (s1.length() > s2.length()) return -1;
            else if (s1.length() < s2.length()) return 1;
            else return 0;
        });

        // Создание компаратора для передачи в качестве аргумента метода
        Comparator<String> comparator = ((s1, s2) -> {
            if (s1.length() > s2.length()) return 1;
            else if (s1.length() < s2.length()) return -1;
            else return 0;
        });

        System.out.println(list);
        list.sort(comparator);
        System.out.println(list);

        // Бинарный поиск
        Collections.sort(list);
        System.out.println(Collections.binarySearch(list, "Hello"));
    }
}
```

Сравнение объектов с помощью hashCode() и equals():

```
@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    Person person = (Person) o;
    return id == person.id && Objects.equals(name, person.name);
}

@Override
public int hashCode() {
    return Objects.hash(id, name);
}
```

Итератор:

```
Iterator <Integer> iterator = list.iterator();
while (iterator.hasNext()) {
    System.out.println(iterator.next());
    if (idx == 1){
        iterator.remove();
    }
    idx ++;
}
System.out.println(list);
```