# Java8

## JS8 Course

### Functional Programming

- Functional Interface — это interface с одним абстрактным методом — **java8/func**
  - используется — для автоматического программирования
  - — указывается только то «что нужно сделать» и источник данных
  - — используется внутренний механизм итерации
  - преимущества — не надо указывать параметры цикла итерации
  - — не надо создавать final переменные
  - — не надо думать о многопоточности
  - package — java.util.function

### Functional Interface

- Functional Interface — это interface с одним абстрактным методом — **java8/func**
  - может — содержать любое число default и static методов

### Lambda

- Lamda expressions — **java8/lambda**
  - shorthand — for functional interface
  - — используется как краткая запись функционального интерфейса с одним методом
  - свойства — не имеет имени метода или класса, полностью анонимна
  - — имеет параметры, тело и возвращаемое значение как метод
  - — может быть параметром или переменной, вставляться прямо в метод
  - — имеет существенно сокращенную запись
  - синтаксис — (parameterList) -< { statements } применение — object.method()
- Пример. реализация — **java8/lambda**
- ```
  private interface ILambda2 {
      int method(int x, int y);
  }
  private static int sum(int[] a, ILambda2 iL2) {
      return a[0] + iL2.method(a[1], a[2]);
  }
  public static void main(String[] args) {
      ILambda2 iL2 = (x, y) -> {
          return 2 * x + y;
      };
      int z = iL2.method(1, 2);
      int z2 = sum(new int[]{1, 2, 3}, (x, y) -> x * y);
  }
  ```
  ```

## Interface Default Methods

- Interface Default Methods **java8/streams/Main07default**
  - свойства тело обязательно должно быть определено в интерфейсе
  - могут вызываться из любого класса реализующего интерфейс
  - могут быть добавлены в интерфейс в любой момент БЕЗ реализации в классах
  - default методы позволяют наследовать интерфейсы как классы но множественно
- ВНИМАНИЕ. При diamond коллизии обязательно переопределить метод в реализующем классе
- Static Methods
  - свойства принадлежат ТОЛЬКО классу интерфейса, вызываются только из класса интерфейса
  - могут иметь одинаковую сигнатуру с методами реализующего класса
- Пример. реализации
- ```
  public class Main07default {
      private interface IA {
          void print();
          static void calc(int x, int y) {
              System.out.println("IA calc plus");
          }
          static void reuse(int x, int y) {
              System.out.println("IA reuse calc");
          }

          default void mul(int x, int y) {
              System.out.println("IA calc mul");
          }
          default void div(int x, int y) {
              System.out.println("IA calc div");
          }
      }

      private interface IB {

          default void reuse(int x, int y) {
              System.out.println("IA reuse calc");
          }

      }

      private interface IC {

          default void count(int x, int y) {
              System.out.println("IC count calc");
          }


          default void average(int x, int y) {
              System.out.println("IC average calc");
          }
      }

      private interface ID  extends IC{

          default void reuse(int x, int y) {
              System.out.println("ID reuse calc");
          }

      }
  ```
  ```

- Пример. реализации продолжение
-
```
    public static void main(String[] args) {
        A a = new A();
        B b = new B();
        C c = new C();
        D d = new D();

        System.out.println("\nInterface IA:");
        IA.calc(1,2);
        IA.reuse(1,2);
//        A.reuse(1,2);
//        B.reuse(1,2);

        System.out.println("\nClass A:");
        A.calc(1,2);

        System.out.println("\nClass B:");
        B.calc(1,2);
        b.print();
        b.mul(1,2);

        System.out.println("\nClass C:");
        c.div(1,2);
        c.print();
        c.calc(1,2);
        c.reuse(1,2);

        System.out.println("\nClass D:");
        d.reuse(1,2);                    // collision
        d.count(1,2);
        d.average(1,2);
    }

    private static class A implements IA {
        static void calc(int x, int y) {
            System.out.println("A calc plus");
        }

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

    private static class B extends  A {
        @Override
        public void mul(int x, int y) {
            System.out.println("B calc mul");
        }
    }

    private static class C extends  A implements IB, IC{

    }

    private static class D implements IB, ID{
        @Override
        public void count(int x, int y) {
            System.out.println("Class D count calc");
        }

        @Override
        public void reuse(int x, int y) {
            System.out.println("Class D reuse calc");
        }
    }

}
```

## Streams

- Streams — это объекты реализующие интерфейс Stream — java8/streams
  - позволяют — выполнять Functional Programming Tasks
  - special — используются для работы с int, long или double значениями

## Stream Pipeline

- Stream Pipeline — потоки для работы с данными
  - прогоняют — массив входных данных через последовательность шагов обработки
  - data source — входные данные для потока
  - — исходные данные для pipeline массив, коллекция или сгенерированные данные
  - intermediate — задачи которые формируют процесс обработки
  - — операции обработки, порождают на выходе такой же stream
  - — по умолчанию lazy — ожидают выполнения terminal операции
  - terminal — запускают промежуточные задачи, затем выполняются сами и выдают результат
  - — eager — выполняются только тогда, когда есть запрос
  - reusage — для pipeline НЕВОЗМОЖЕН в принципе

## Method Reference

- Method Reference — способы ссылок на методы в lambda выражениях
  - instance метод

```
Stream.of(strings)
        .sorted(String::compareToIgnoreCase) );       // instance method
        .collect(Collectors.toList()));
 Stream.of(strings)
         .sorted((v1,v2)-> v1.compareToIgnoreCase(v2)) // instance method specifi
         .collect(Collectors.toList()));
```

  - instance метод конкретного объекта

```
List<Integer> list = new ArrayList<>();
IntStream.of(values).forEach(list::add);
IntStream.of(values).forEach(v->list.add(v));
```

  - static метод

```
Stream.of(strings)
        .sorted(String::compareToIgnoreCase) );       // instance method
        .collect(Collectors.toList()));               // static method
```

  - constructor

```
private static Collector<Integer, ?, List<Integer>> toList(Integer[] values) {
    Supplier<List<Integer>> sA = () -> new ArrayList<>();     // constructor List<>
    BiConsumer<List<Integer>, Integer> cT = (v, d) -> v.add(d);
    BinaryOperator<List<Integer>> bA = (v, w) -> {            // добавление группы элементов
        v.addAll(w);                                          //        R apply(T t, U u);
        return v;
    };
    final Set<Collector.Characteristics> CH_ID = Collections.unmodifiableSet(EnumSet.
                                          of(Collector.Characteristics.IDENTITY_FINISH));
    Collector<Integer,?,List<Integer>> c2= new Collector2<>(sA, cT, bA, CH_ID); //constructor
    return c2;
}

private static Collector<Integer, ?, List<Integer>> toList2(Integer[] values) {
   final Set<Collector.Characteristics> CH_ID = Collections.unmodifiableSet(EnumSet.
                                          of(Collector.Characteristics.IDENTITY_FINISH));

    Collector<Integer, ? extends List<Integer>, List<Integer>> c2 = new Collector2<>(
      ArrayList::new, List::add, (v, w) -> { v.addAll(w);  return v; }, CH_ID); //constructor
    return c2;

}
```

## BinaryOperator Basic Functional Interfaces

- BinaryOperator<T> использует два аргумента <T>, выполняет операцию, возвращает <T>
- Пример. реализация
- ```
  public interface BinaryOperator<T> extends BiFunction<T,T,T> {
      public static <T> BinaryOperator<T> minBy(Comparator<? super T> comparator) {
          Objects.requireNonNull(comparator);
          return (a, b) -> comparator.compare(a, b) <= 0 ? a : b;
      }
  }
  ```
- ```
  public interface IntBinaryOperator {
      int applyAsInt(int left, int right);
  }
  ```
- ```
  IntBinaryOperator intBinAdd = (v1, v2) -> v1 + v2;
  System.out.printf("int[] reduce add: %d", IntStream.of(values).reduce(intBinAdd).getAsInt());
  ```
- ```
  BinaryOperator<Integer> binAdd = (v1,v2)-> v1 + v2;
  System.out.printf("List<> add: %d", list.stream().limit(10).reduce(binAdd).orElseGet(()->0));
  ```

- Consumer<T> использует один аргумент <T> и выполняет операцию, возвращает void
- Пример. реализация

```
public interface Consumer<T> {
    void accept(T t);
}

Consumer<Integer> consumer = (value)-> System.out.printf("%d.",value);// Consumer<T>
Predicate<Integer> predicate = v -> v % 2 == 0;
list.stream().filter(v -> v % 2 == 0).sorted().forEach(v -> System.out.printf("%d ", v));
list.stream().filter(predicate).sorted().forEach(consumer);
```

- ```
  IntConsumer intC = (value) -> System.out.printf("%d ", value); // used Consumer<T>
  ```
- ```
  IntStream.of(values)
                  .filter(v -> v % 2 == 0)                          // intermediate
                  .sorted()                                         // intermediate
                  .forEach(v -> System.out.printf("%d ", v));       // terminal
  IntStream.of(values).filter(intP).sorted().forEach(intC);
  ```

- Functional<T,R>использует аргумент <T>, выполняет операцию и возвращает <R>
- Пример. реализация
- ```
  public interface IntFunction<R> {
      R apply(int value);
  }
  ```
- ```
  public interface Function<T, R> {
      R apply(T t);
  ```
- ```
  }
  ```
- ```
  IntFunction<String> intF = v->String.valueOf(v)+"..";
  IntFunction<String> intFS =String::valueOf;
  Function<Integer,String> function = v->String.valueOf(v)+"..";
  Function<Integer,String> functionS = String::valueOf;

  System.out.println("int[]: " + IntStream.of(values)
                                      .mapToObj(String::valueOf)
                                      .collect(Collectors.joining("_")));
  System.out.println("list : " + list.stream()
                                      .map(Object::toString)
                                      .collect(Collectors.joining("_")));
  //function
  System.out.println("int[]: " + IntStream.of(values)
                                      .mapToObj(intF)
                                      .collect(Collectors.joining()));
  System.out.println("int[]: " + IntStream.of(values)
                                      .mapToObj(intFS)
                                      .collect(Collectors.joining("_")));
  System.out.println("list: " + list.stream().map(function).collect(Collectors.joining()));
  System.out.println("list: " + list.stream().map(functionS).collect(Collectors.joining("_")));
  ```

- Predicate<T> использует аргумент<T>, выполняет сравнение и возвращает boolean
  - default методы and, or, negate позволяют объединять Predicate<T> в условия
- Пример. реализация

```
Predicate<Integer> predicate = (value)-> value > 5;
public interface Predicate<T> {
    boolean test(T t);
}
```

```
IntPredicate intP = v -> v % 2 == 0;
IntPredicate intP2 = v -> v % 2 != 0;
IntUnaryOperator intF = v -> v * 10;
System.out.print("\nint[] even sorted  : ");
IntStream.of(values).filter(intP).sorted().forEach(intC);
System.out.print("\nint[] all mul sorted: ");
IntStream.of(values).filter(intP2.or(intP)).sorted().map(intF).forEach(intC);
```

- Supplier<T> без аргументов, выполняет операцию и вовзращает результат <T>
  - используется для заполнения коллекций
- UnaryOperator<T> использует аргумент<T> , выполняет операцию и возвращает результат <T>
  - расширяет Functional<T,T>

```
public interface IntUnaryOperator {
    int applyAsInt(int operand);
}
IntPredicate intP = v -> v % 2 == 0;
IntUnaryOperator intF = v -> v * 10;
IntConsumer intC = (value) -> System.out.printf("%d ", value); // used Consumer<T>
System.out.print("\nint[] odd mul sorted: ");
IntStream.of(values).filter(v -> v % 2 != 0)
                .sorted()           //
                .map(v -> v * 10)
                .forEach(v -> System.out.printf("%d ", v));
IntStream.of(values).filter(intP2).sorted().map(intF).forEach(intC); //predicate, function
```

- Collector<T,A,R> `public interface Collector<T, A, R>`
  - интерфейс для создания объектов Mutable Reduction
- Пример. реализация toList() создание List<T> из входного массива T[]

```
public interface Collector<T, A, R> {
    Supplier<A> supplier();
    BiConsumer<A, T> accumulator();
    BinaryOperator<A> combiner();
    Function<A, R> finisher();
}
```

```
public static <T>
Collector<T, ?, List<T>> toList() {
    return new CollectorImpl<>((Supplier<List<T>>) ArrayList::new, List::add,
                               (left, right) -> { left.addAll(right); return left; },
                               CH_ID);
}
```

```
        System.out.println("collect sorted:"+ Arrays.stream(values).sorted()
                                                .collect(Collectors.toList()));
         System.out.println("collect filter: " + Stream.of(values).filter(v -> v > 4)
                                                .sorted().collect(Collectors.toList()));

         List<Integer> g4 = Stream.of(values).filter(v -> v > 4).collect(Collectors.toList());
         System.out.println("List<> g4 unsorted : " + g4);
         System.out.println("g4 sorted:"+ g4.stream().sorted().collect(Collectors.toList()));
```

- Пример. реализация на базе Custom Collector
- ```
  private static class Collector2<T, A, R> implements Collector<T, A, R> {
      private final Supplier<A> supplier;
      private final BiConsumer<A, T> accumulator;
      private final BinaryOperator<A> combiner;
      private final Function<A, R> finisher;
      private final Set<Characteristics> characteristics;

      Collector2(Supplier<A> supplier, BiConsumer<A, T> accumulator,
                 BinaryOperator<A> combiner,Function<A, R> finisher,
                 Set<Characteristics> characteristics) {
          this.supplier = supplier;
          this.accumulator = accumulator;
          this.combiner = combiner;
          this.finisher = finisher;
          this.characteristics = characteristics;
      }
      Collector2(Supplier<A> supplier, BiConsumer<A, T> accumulator,
                 BinaryOperator<A> combiner,Set<Characteristics> characteristics) {
          this(supplier, accumulator, combiner, castingIdentity(), characteristics);
      }
      private static <I, R> Function<I, R> castingIdentity() {
          return i -> (R) i;
      }
      public BiConsumer<A, T> accumulator() {
          return accumulator;
      }
      public Supplier<A> supplier() {
          return supplier;
      }

      public BinaryOperator<A> combiner() {
          return combiner;
      }
      public Function<A, R> finisher() {
          return finisher;
      }
      public Set<Characteristics> characteristics() {
          return characteristics;
      }
  ```

- Пример. реализация на базе Custom Collector продолжение
- ```
     private static Collector<Integer, ?, List<Integer>> toList(Integer[] values) {
          Supplier<List<Integer>> sA = () -> new ArrayList<Integer>(); // создание List<>
          BiConsumer<List<Integer>, Integer> cT = (v, d) -> v.add(d);  // добавление элемента
          BinaryOperator<List<Integer>> bA = (v, w) -> {               // добавление элементов
              v.addAll(w);                                             // R apply(T t, U u);
              return v;
          };
          final Set<Collector.Characteristics> CH_ID = Collections.unmodifiableSet(EnumSet
                                                  .of(Collector.Characteristics.IDENTITY_FINISH));
          Collector<Integer, ?, List<Integer>> c2 = new Collector2<>(sA, cT, bA, CH_ID);
          return c2;
      }
  }
  ```
- ```
          Collector<Integer, ?, List<Integer>> c2 = Collector2.toList(values);  // ? any type
          System.out.println("List<> user collect2 : " + Stream.of(values)
                  .filter(v -> v < 4)
                  .sorted()
                  .collect(c2));  // returns List<Integer>
  ```

- f
  ```

## Stream Intermediate Operations

- Применение. **java8/streams/Main02i**
- filter() `Stream<T> filter(Predicate<? super T> predicate);` **java8/streams/Main01**
    - возвращает поток, Predicate<T> отбирает элементы в новый поток по условию
- Пример. реализация filter() и Predicate<T>

```
Consumer<Integer> consumer = (value)-> System.out.printf("%d.",value); // used Consumer<T>
Predicate<Integer> predicate = (value)-> value > 5;
list.stream().filter(predicate).forEach(consumer);
list.stream().filter((value)-> value > 5).forEach((value)-> System.out.printf("%d,",value));
```

- distinct() `Stream<T> distinct();` **java8/streams/Main01**
    - возвращает поток, отбирает уникальные элементы в новый поток
- Пример. реализация distinct()

```
list.stream().distinct().forEach(consumer);
```

- limit() `Stream<T> limit(long maxSize);` **java8/streams/Main01**
    - возвращает поток, отбирает в новый поток заданное число элементов
- Пример. реализация limit()
- `list.stream().limit(7).forEach(consumer);`

- map() `<R> Stream<R> map(Function<? super T, ? extends R> mapper);` **java8/streams/Main01**
    - возвращает поток, в новый поток попадают обработанные значения того же или другого типа
- ВНИМАНИЕ. Для потоков IntStream, LongStream и так далее новый поток ТОГО ЖЕ типа
- Пример. реализация map() и Function<T,R>
- 

```
Function<Integer,String> functionS = (value)-> String.valueOf(value);
Consumer<String> consumerS = (value)-> listS.add(value);
list.stream().map(functionS).forEach(consumerS);
list.stream().map(String::valueOf).forEach(listS::add);
```

- sorted() `Stream<T> sorted();` **java8/streams/Main01**
- `Stream<T> sorted(Comparator<? super T> comparator);`
    - возвращает поток, в новый поток сортированные значения стандартно или Comparator
- Пример. реализация sorted() и Comparator<T>
- 

```
Comparator<Integer> cInt = Integer::compare;
Comparator<Integer> rInt = (i1,i2)->Integer.compare(i2,i1);
Comparator<String> cStr = String::compareTo;
Comparator<String> cStrN = (s1,s2)->Integer.compare(Integer.valueOf(s1),Integer.valueOf(s2));
Comparator<String> cStrNS = Comparator.comparingInt(Integer::valueOf);
Comparator<String> rStr = (s1,s2)-> s2.compareTo(s1);
Comparator<String> rStrN = (s1,s2)->Integer.compare(Integer.valueOf(s2),Integer.valueOf(s1));

System.out.println("Integer:");
list.stream().limit(10).sorted().forEach(consumer);
list.stream().limit(10).sorted(rInt).forEach(consumer);
list.stream().limit(10).sorted(Comparator.reverseOrder()).forEach(consumer);
```

- 

```
System.out.println("String:");
Consumer<String> consumerP = (s)->System.out.print(s+" ");
listS.stream().limit(10).sorted().forEach(consumerP);
listS.stream().limit(10).sorted(rStr).forEach(consumerP);
listS.stream().limit(10).sorted(Comparator.reverseOrder()).forEach(consumerP);
System.out.println("String as Integer:");
listS.stream().limit(10).sorted(cStrN).forEach(consumerP);
listS.stream().limit(10).sorted(cStrNS).forEach(consumerP);
listS.stream().limit(10).sorted(rStrN).forEach(consumerP);
```

- range() `static IntStream range(int startInclusive, int endExclusive)` **java8/streams/Main02i**
  - создает поток последовательности значений
  - можно использовать как индексы
- Пример. реализация

```
// range
System.out.println("\n range :"+IntStream.range(1,10).sum());        // сумма элементов
System.out.println("\n range :"+IntStream.rangeClosed(1,10).sum()); // сумма элементов
long[] longs = new long[values.length];
IntStream.range(1,values.length).forEach(i->longs[i] = values[i]);
LongStream.of(longs).forEach(v-> System.out.println(v+" "));
// range as index
long[] longs = new long[values.length];
IntStream.range(0,values.length).forEach(i->longs[i] = values[i]);
LongStream.of(longs).forEach(v-> System.out.print(v+" "));
IntStream.range(0,values.length).forEach(i-> System.out.printf("%d ",values[i]));
```

## Stream Terminal Operations

- forEach() `void forEach(IntConsumer action);` **java8/streams/Main02t**

```
List<Integer> list = new ArrayList<>();
IntConsumer consumerA = (value)-> list.add(value);
Consumer<Integer> consumerP = (value)-> System.out.printf("%d.",value);
IntStream.of(values).forEach(consumerA);            // add value to list
list.stream().sorted().forEach(consumerP);           // print every value
```

## Stream Terminal Reduction Operations

- average(), count(), max(), min()
  - методы обработки элементов потока и выдача результата
- Optional<T> используется для обработки пустых массивов
  - getAsDouble() метод возвращает double, если пусто NullPointerException
  - orElse(value) метод, возвращает <value> если пусто, допускает null
  - orGetElse() тоже самое, но используется Supplier<T>
  - ifPresent() метод принимает Consumer<Optional<T>> который выполняет операцию
  - isPresent() определяет валидно ли Optional<T> или нет
- Пример. реализации

```
//integer
System.out.printf("  count: %d \n",IntStream.of(values).count());
System.out.printf("    min: %d \n",IntStream.of(values).min().getAsInt());
System.out.printf("    max:%d \n", IntStream.of(values).max().getAsInt());
System.out.printf("average:%.1f\n",IntStream.of(values).average().getAsDouble()); //no check
System.out.printf("average:%.1f\n",IntStream.of(values).average().orElse(0));      //0 empty
System.out.printf("average:%.1f\n",IntStream.of(values).average().orElseGet(()->0));//0 empty
IntStream.of(values).average().ifPresent((e)->System.out.println("optional ifPresented:"+e));
OptionalDouble oD = IntStream.of(values).average();
if(oD.isPresent()) System.out.println("optioonal isPresented:"+oD.getAsDouble());
//average mapToInt
list.stream().mapToInt(value -> value).average().orElse(0));
```

- summarystatistics()
  - возвращает объект который имеет методы average(), count(), max(), min(), sum
- Пример. реализации

```
IntSummaryStatistics ss =  IntStream.of(values).summaryStatistics();
System.out.println(ss.getAverage()+" "+ss.getCount()+" "+ss.getMax()+" "+ss.getMin()+"
"+ss.getSum());
IntSummaryStatistics ss = list.stream().mapToInt(value -> value).summaryStatistics();
System.out.printf("count: %d min: %d max: %d sum: %d  average: %.1f %n",
                  ss.getCount(),ss.getMin(),ss.getMax(),ss.getSum(),ss.getAverage());
```

- ВНИМАНИЕ. Метод list.stream().mapToInt(v->v) КОНВЕРТИРУЕТ поток Stream<Integer> в IntStream

- reduce()  `T reduce(T identity, BinaryOperator<T> accumulator)`
  `OptionalInt reduce(IntBinaryOperator op)`
  - использует обработку всех элементов массива с использованием BinaryOperator
  - identity начальное значение, для BinaryOperator использует с первым элементом потока
  - InteStream, Stream<T> используется например 1 для умножения
  - Stream<T> используется вместо orElse() для пустых коллекций
  -
- Пример. реализация
- `//reduce`
- ```
  System.out.printf("int[]: %d \n", IntStream.of(values).reduce((v1,v2)-> v1+v2).getAsInt());
  System.out.printf("int[]: %d \n", IntStream.of(values).reduce(0,(v1,v2)-> v1+v2));
  System.out.printf("int[]: %d \n", IntStream.of(values).reduce((v1,v2)-> v1* v2).getAsInt());
  System.out.printf("int[] reduce mul: %d \n", IntStream.of(values).reduce(1,(v1,v2)-> v1*v2));
  ```

- Пример. реализация продолжение
- `//reduce int[] empty`
- ```
  System.out.printf("int[]empty: %d\n", IntStream.of(empty).reduce((v1,v2)-> v1+v2).orElse(0));
  System.out.printf("int[]empty add: %d \n", IntStream.of(empty).reduce(0,(v1,v2)-> v1 + v2));
  System.out.printf("int[]empty: %d\n", IntStream.of(empty).reduce((v1,v2)-> v1*v2).orElse(1));
  System.out.printf("int[]empty mul: %d \n", IntStream.of(empty).reduce(1,(v1,v2)-> v1*v2));
  ```
- `//reduce List<Integer>`
- ```
  System.out.printf("List<> reduce: %d \n",list.stream().limit(10).reduce(0,(v1,v2)->v1+v2));
  System.out.printf("List<> : %d \n",list.stream().limit(10).reduce((v1,v2)->v1+v2).get());
  //reduce List<Integer> empty
  ```
- ```
  System.out.printf("List<> empty reduce: %d \n",listEmpty.stream().reduce(0,(v1,v2)->v1+v2));
  System.out.printf("List<> %d \n",listEmpty.stream().reduce((v1,v2)->v1+v2).orElse(0));
  System.out.printf("List<> %d \n",listEmpty.stream().reduce((v1,v2)->v1+v2).orElseGet(()->0));
  ```

## Stream Terminal Mutable Reduction Operations

- Terminal Mutable Reduction Operation **java8/streams/Main02tList**
  - это операция которая создает один объект, который содержит все элементы потока
  - не сокращает все элементы до одного, как reduce, а создает один объект
  - вместо нескольких, который содержит все элементы потока
- collect()
  - аргумент интерфейс Collector<T, A,R>
  - Collectors.toList() используется для создания List<> из массива
  - Collectors.goupingBy() используется для создания Map<>
- Пример. реализация List<>
- ```
  System.out.println("sorted:"+ Arrays.stream(values).sorted().collect(Collectors.toList()));
  System.out.println("collect filter: " + Stream.of(values).filter(v -> v > 4)
                                                  .sorted().collect(Collectors.toList()));
  List<Integer> g4 = Stream.of(values).filter(v -> v > 4).collect(Collectors.toList());
  System.out.println("g4 sorted:"+ g4.stream().sorted().collect(Collectors.toList()));
  ```
- Пример. реализация Map<> **java8/streams/Main03Employee**
- ```
  // distinct map
          System.out.printf("%nEmployees distinct map asc: %n");
          Map<String, List<Employee>> fMap =
                  list.stream().collect(Collectors.groupingBy(e -> e.getLastName()));
          list.stream().map(Employee::getLastName)
                  .distinct()
                  .sorted()
                  .forEach(s-> {
                      System.out.println(fMap.get(s).stream().findFirst().get());
                  });
  ```

- Пример. реализация Map<> java8/streams/Main03Employee
- ```
  // distinct map
  Map<String, List<Employee>> fMap =
                  list.stream().collect(Collectors.groupingBy(e -> e.getLastName()));
  list.stream().map(Employee::getLastName)
                  .distinct()
                  .sorted()
                  .forEach(s-> {
                      System.out.println(fMap.get(s).stream().findFirst().get());
                  });
  ```
- ```
  // group
  Map<String, List<Employee>> dMap =
                  list.stream().collect(Collectors.groupingBy(e -> e.getDepartment()));
  dMap.forEach((key, value) -> {
              System.out.println(key);
              value.forEach(e -> System.out.printf("     %s%n", e));
          });
  ```
- ВНИМАНИЕ. Метод groupingBy() КОНВЕРТИРУЕТ поток Stream<Integer> в Map(key,value)
- 
- ```
  // group sorted by key
          System.out.printf("%nEmployees sorted by department then salary: %n");
          Map<String, List<Employee>> dMapS =
                  list.stream().collect(Collectors.groupingBy(
                          Employee::getDepartment,
                          TreeMap::new,
                          Collectors.toList()));
          dMapS.forEach((key, value) -> {
              System.out.println(key);
              value.stream().sorted(Comparator.comparing(Employee::getSalary))
                      .forEach(e->System.out.printf("     %s%n", e));
          });
  ```
- ВНИМАНИЕ. Метод groupingBy(K,new,toList) КОНВЕРТИРУЕТ поток Stream<T> в TreeMap(K,List<>)
- ```
  // group and count
          System.out.printf("%nEmployees count by department: %n");
          Map<String, Long> cMap =
                  list.stream().collect(
                          Collectors.groupingBy(
                                  e -> e.getDepartment(),
                                  () -> new TreeMap<String, Long>(),
                                  Collectors.counting()));
          cMap.forEach((key, value) -> System.out.println(key + ":" + value));
  ```
- ```
  // group and count
          System.out.printf("%nEmployees count by department: %n");
          Map<String, Long> eMap = list.stream().collect(
          Collectors.groupingBy(Employee::getDepartment, TreeMap::new, Collectors.counting()));
          eMap.forEach((key, value) -> {
              System.out.printf("%-10s: %d%n",key, value);
          });
  ```
- ВНИМАНИЕ. Метод groupingBy(F,new,int) КОНВЕРТИРУЕТ поток Stream<Integer> в TreeMap(key,value)
- ```
  // group sum, average
  ```
- ```
  System.out.printf("%nEmployees sum salaries: %.1f",
      list.stream().mapToDouble(e -> e.getSalary()).sum());
  ```
- ```
  System.out.printf("%nEmployees sum salaries: %.1f",
      list.stream().mapToDouble(Employee::getSalary).reduce(0, (v1, v2) -> v1 + v2));
  ```
- ```
  System.out.printf("%nEmployees average salary: %.1f%n",
      list.stream().mapToDouble(Employee::getSalary).average().orElse(0));
  ```
- ```
  System.out.printf("%nEmployees average salary by dept%n");
      list.stream().collect(Collectors.groupingBy(Employee::getDepartment, TreeMap::new,
  ```
- ```
      Collectors.toList())).forEach((k, v) -> System.out.printf("%-10s: %.1f%n", k,
  ```
- ```
       v.stream()
  ```
- ```
            .map(Employee::getSalary)
  ```
- ```
            .mapToDouble(s->s).average().orElse(0)));
  ```
- ВНИМАНИЕ. ОЧЕНЬ КРУТЫЕ ПРЕОБРАЗОВАНИЯ average salary by department
- toArray()

## Stream Terminal Search Operations

- findFirst() **java8/streams/Main03Employee**
  - находит первый элемент оригинального потка в отфильтрованном потоке
- Пример. реализация
- ```
  Predicate<Employee> fTSP = e -> e.getSalary() >= 4000 && e.getSalary() <= 6000;
// findFirst()
System.out.printf("%nEmployees 4000..6000 first: %n%s %n",
        list.stream()
                .filter(fTSP)
                .findFirst().orElse(null));
  ```
- findAny() **java8/streams/Main03Employee**
  - находит любой элемент из выборки, значение выбирается прозвольно
- Пример. реализация
- ```
  System.out.printf("%nEmployees 4000..6000 any : %n%s %n",
        list.stream()
                .filter(fTSP)
                .findAny().orElse(null));
  ```
- 
- anyMatch() **java8/streams/Main03Employee**
  - выдает boolean результат проверки на совпадение хотя бы одного элемента
- Пример. реализация
- ```
  System.out.printf("%nEmployees 4000..6000 match: %n%s %n",
        list.stream()
                .filter(fTSP)
                .anyMatch(e->e.getLastName().startsWith("I")));
  ```
- 
- allMatch() **java8/streams/Main03Employee**
  - выдает boolean результат проверки на совпадение всех элементов
- Пример. реализация

```
System.out.printf("%nEmployees 4000..6000 match: %n%s %n",
        list.stream()
                .filter(fTSP)
                .allMatch(e->e.getLastName().startsWith("I")));
```
```

## Stream Pipeline Code Examples

- of() `IntStream of(int... values)`
  - получает список значений, возвращает поток значений массива

```
public static IntStream of(int... values) {
    return Arrays.stream(values);
}
```

- forEach `forEach(IntConsumer action)`
  - каждое значение передается как аргумент объекту IntConsumer

```
void forEach(IntConsumer action);
public interface IntConsumer {
    void accept(int value);
}
IntStream.of(values).forEach(value -> System.out.printf("%d ",value)); // consumer implicit
IntConsumer intConsumer = (value)-> System.out.printf("%d.",value);    // consumer explicit
IntStream.of(values).forEach(intConsumer);
```

## Stream File Code Example

- Stream File Code Example **java8/streams/Main04File**
  - Files.lines(Path) поток строк текста из файла
  - Paths.get(String) путь до файла
  - можно текстовый файл разместить в папке проекта и обращаться по имени
  - map() применяет regex к каждой строке текста
  - flatMap() порождает новые элементы потока, разбивает строки текста на слова
  - collect() создает Map<>
- Пример. реализация

```
Files.lines(Paths.get("data/Ch2P.txt"))  // file to stream of lines
.map(line -> line.replaceAll(regex, ""))
.flatMap(line -> pattern.splitAsStream(line))
.collect(Collectors.groupingBy(String::toLowerCase,
        TreeMap::new, Collectors.counting()));      // map of words and count
```

- Stream Map via EntrySet
  - map.entrySet() создает коллекцию Set, которая может создать stream
  - collect() создает новую Map<> из Set, по новому ключу, размещая в value Map.Entry
  - .forEach() отрабатывает новую Map<>

```
Map<Character, List<Map.Entry<String, Long>>> eMap =
        map.entrySet()
                .stream()
        .collect(Collectors.groupingBy(e -> e.getKey().charAt(0),
                TreeMap::new, Collectors.toList()));
 eMap.forEach((letter, list)-> {

     System.out.printf("%n%C%n",letter);
     list.stream().forEach(e->
             System.out.printf("%12s: %d%n",e.getKey(),e.getValue()));

 });
```

## Stream Random Multithreading Code Example

- Stream Random Multithreading Code Example **java8/streams/Main05Random**
  - stream — создание случайных чисел в многопоточном режиме
  - ints() — SecureRandom криптографически сильный генератор rnd
  - `ints(100,1,10)` генерирует 100 чисел в пределах от 1 до 9 включительно
  - boxed() — переводит из stream<int> в stream<Integer>
  - обратный метод `mapToInt(v->v)`
  - collect() — создает Map<Integer,Long>
  - Function.identity() — аналог v->v для Integer
  - now() — Instant.now() — создание временной метки
  - between() — Duration.between() — вычисление разности между метками
- Пример. реализация Sequential and Parallel stream()
- ```
  // stream multithreading
  // immutable
  // lambda
  // internal iteration
          Random random = new Random();
          int[]values = random.ints(60_000_000, 1, 7).toArray();
  // sequential
          System.out.printf("%-6s%s%n", "Face", "Frequency");
          Instant start = Instant.now();
          Map<Integer, Long> map =
                  IntStream.of(values)          // 1..6 (linclusive 7 exclusive)
                           .boxed()             // to stream<Integer>
                           .collect(Collectors.groupingBy(Function.identity(),
                                                Collectors.counting())); //Map<Integer,Long>
          map.forEach((key, value) -> {
              System.out.printf("%-6d%d  %.2f%% %n", key, value, 100 * (value - 1e7) / 1e7);
          });
          Instant middle = Instant.now();

  // parallel
          System.out.printf("%-6s%s%n", "Face", "Frequency");
          Map<Integer, Long> map2 =
                  IntStream.of(values).parallel()
                           .boxed()              // to stream<Integer>
                           .collect(Collectors.groupingBy(Function.identity(),
                                                Collectors.counting())); //Map<Integer,Long>
          map2.forEach((key, value) -> {
              System.out.printf("%-6d%d  %.2f%% %n", key, value, 100 * (value - 1e7) / 1e7);
          });
          Instant end = Instant.now();
          System.out.printf("Sequential: %d%n",Duration.between(start,middle).toMillis());
          System.out.printf("Parallel  : %d%n",Duration.between(middle,end).toMillis());
      }
  ```
-
  ```

## Stream Pipeline Code Examples

- Simple lambda
- Пример. реализация просто вывод на печать содержимое массива

```
int[] values = { 3,10,6,1,4,8,2,5,9,7 };
IntStream.of(values).forEach(value -> System.out.printf("%d ",value));  // consumer used
IntConsumer intConsumer = (value)-> System.out.printf("%d.",value);    // consumer explicit
IntStream.of(values).forEach(intConsumer);
```

- Class method lambda
  - используется сокращенная форма метода { Class::method }
  - означает аргумент типа Class подается в метод method(Class) и метод возвращает результат
  - аналог { (e) -> method(e) }
- Пример. реализация просто вывод на печать содержимое массива
- 

```
List<Integer> list = new ArrayList<>();
IntStream.of(values).forEach((e)->list.add(e));     // standard
IntStream.of(values).forEach(list::add);            // shortcut
```

- 

## Stream Lambda Event Handlers Code Example

- Lambda Event Handlers **java8/streams/Main06Event**
  - применение lambda в оформлении Listeners
- Пример. реализация
- 

```
public class Main6Event {
    public static void main(String[] args) {
        JFrame jFrame = new ComboBoxFrame();
        jFrame.setSize(800, 600);
        jFrame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        jFrame.setLocationRelativeTo(null);
        jFrame.setVisible(true);
    }

    private static class ComboBoxFrame extends JFrame {
        private static final String[] NAMES = {
                "bug1.gif", "bug2.gif", "travelbug.gif", "buganim.gif"
        };
        private final Icon[] ICONS;
        private final JComboBox<String> mComboBox;
        private final JLabel mLabel;

        public ComboBoxFrame() {
            super("Testing JComboBox");
            setLayout(new FlowLayout());

            ICONS = new Icon[NAMES.length];
            Path path = Paths.get("./data");
            for (int i = 0; i < ICONS.length; i++) {
                ICONS[i] = new ImageIcon(path.toString() + "\\" + NAMES[i]);
            }
            mComboBox = new JComboBox<String>(NAMES);
            mComboBox.setMaximumRowCount(3);
            add(mComboBox);
            mLabel = new JLabel(ICONS[0]);
            add(mLabel);

            mComboBox.addItemListener(item -> {
                if (item.getStateChange() == ItemEvent.SELECTED) {
                    mLabel.setIcon(ICONS[mComboBox.getSelectedIndex()]);
                };
            });
        }
    }
```

## Predicate With HashMap

- Predicate With HashMap **java8/examples/Main01L**
  - map заполняется <Class, Predicate>
  - Generic использовать не получится, т.к. у объекта(T o) берется класс o.getClass()
  - этот метод никак не ограничен, а значит надо переопределять Object >> unchecked
  - Predicate для каждого класса свой Predicate, и переопределяется тип входного значения
  - ofNullable Optional.ofNullable() использует Predicate как входное значение
  - выдает null или сам Predicate, по сути потоковая проверка значения
  - orElseThrow Optional.orElseThrow использует Predicate как входное значение
  - выдает заданный тип Exception для null или сам Predicate,тоже потоковая проверка
  -
- Пример. реализация
- ```
  private static final Map<Class, Predicate> map;
  static {
      map = new HashMap<>();
      map.put(Integer.class, (Predicate<Integer>) v -> v > 10);
      map.put(String.class, (Predicate<String>) v -> v.length() > 10);
      map.put(Double.class, (Predicate<Double>) v -> Double.compare((Double) v, 10) > 0);
  }

  private static Predicate p = v ->
          Optional.ofNullable(map.get(v.getClass()))
                  .orElseThrow(()-> new IllegalArgumentException())
                  .test(v);
  ```
- Predicate без HashMap **java8/examples/Main01P**
  - Predicate создается один Predicate и для каждого класса своя ветка реализации
  - и также переопределяется тип входного значения
- Пример. реализация
- ```
  private static Predicate validator = v -> {
      if (v instanceof Integer)
          return (Integer) v > 10;
      if (v instanceof String)
          return ((String) v).length() > 10;
      throw new IllegalArgumentException();
  };
  ```
-

## Functional Interface String Regex Example

- Functional Interface String Regex Example **java8/lambda**
  - задача перекодировать строку chmod в двоичный и затем десятичный код
  - `"-rwxr-xr-x",  // 755`
  - варианты реализации три штуки оригинальный с форума, через массив и быстрый
- Оригинальный вариант
  - regex `(".*([r-][w-][x-]){3}$")` распаковывается так
  - .* любой символ ноль или несколько
  - (abc){3} повторение группы abc 3 раза, то есть abcabcabc
  - [ab] любой из символов a или b
  - ([r-][w-][x-]){3} подойдут комбинации rwx-w-r-x, -wxrw---x и так далее
  - $ конец строки
  - таким образом это полный шаблон строки chmod на 9 символов с любым началом
  - s.matches() использует regex чтобы определить подходит ли строка под шаблон
  - Function(T,R) интерфейс использует входную переменную rwx, как строку
  - rwx.replaceAll() заменяет любой из символов r,w,x на "1"
  - rwx.replaceAll() заменяет любой из символов - на "0"
  - Integer.parseInt() перекодирует строку из двоичного в десятичный, 111 в 7
  - String.valueOf() перекодирует число 7 в строку "7"
  - regex `("(?<=\\G...)"` это особая конструкция, распаковывается так
  - (?<=x) lookupAhead встать на позицию с символа x и далее
  - \\G... найти границу \\G за предыдущим совпадением (...)
  - в результате находит границу через каждые (...) три любых символа
  - .split() метод .split(**"(?<=\\G...)"**)) разбивает строку на подстроки в три символа
  - \\G это boundary matcher как это точно работает
  - stream() создает поток строк из массива строк в три символа каждая
  - mapper запускает Function интерфейс и перекодирует "rwx" >> "111" >> "7"
  - collect joining() объединяет все элементы строк в одну как concatenate()
  - в результате "rwxr-xr-x" >> "rwx","r-x","r-x" >> "111" > 7 > "7","101" > 5 > "5"... >> "755"
  -
- Пример. реализация
- ```
  private static String getDigits(String s) {
      if (!s.matches(".*([r-][w-][x-]){3}$")) {
          throw new IllegalArgumentException("Achtung!");
      }

      Function<String, String> mapper = rwx -> String.valueOf(
              Integer.parseInt(rwx
                      .replaceAll("[rwx]", "1")
                      .replaceAll("-", "0"), 2));

      return  Arrays.stream(s.substring(s.length() - 9)
              .split("(?<=\\G...)"))
              .map(mapper)
              .collect(joining());

  }
  ```
  -
  ```

- Вариант с массивом
  - regex `("."*([r-][w-][x-]){3}$")` распаковывается так
  - .* любой символ ноль или несколько
  - (abc){3} повторение группы abc 3 раза, то есть abcabcabc
  - [ab] любой из символов a или b
  - ([r-][w-][x-]){3} подойдут комбинации rwx-w-r-x, -wxrw---x и так далее
  - $ конец строки
  - таким образом это полный шаблон строки chmod на 9 символов с любым началом
  - s.matches() использует regex чтобы определить подходит ли строка под шаблон
  - skip() пропускает заданное число элементов потока и оставляет только 9
  - map() использует Function(int,int) интерфейс как аргумент
  - в данном случчае идет перекодировка символов char[] потока в целые числа 0 и 1
  - mapToObj() переделывает поток из целых чисел в строки, по одной на каждое число "1", "0"
  - collect() joininig() объединяет строки чисел в одну строку,на выходе строка вида "111101101"
  - regex `("(?<=\\G...)"` это особая конструкция, распаковывается так
  - (?<=x) lookupAhead встать на позицию с символа x и далее
  - \\G... найти границу \\G за предыдущим совпадением (...)
  - в результате находит границу через каждые (...) три любых символа
  - .split() метод .split(**"(?<=\\G...)"**)) разбивает строку на подстроки в три символа
  - \\G это boundary matcher как это точно работает
  - на выходе поток из трех строк "111","101","101"
  - map() использует Function(String,String) интерфейс как аргумент
  - перекодирует тройки двоичных чисел в десятичные и затем в строки "111">"7"
  - collect() joininig() объединяет строки чисел в одну строку, "7","5","5" > "755"
  - в результате "rwxr-xr-x">> char[] ‘r’,’w’.. >> int[] 1,1.. >>"1","1".. >>"111","101"... >>"7","5".. >>"755"
- Пример. реализация
- ```
      private static String getDigits3(String s) {
  // array n
          String s2 = Arrays.stream(s.chars()
                  .skip(s.length() - 9)
                  .map(n -> {
                      if (n == '-') return 0;
                      if (n == 'r' || n == 'w' | n == 'x') return 1;
                      return n;
                  })
                  .mapToObj(Integer::toString)
                  .collect(joining())
                  .split("(?<=\\G...)"))
                  .map(v -> Integer.parseInt(v, 2) + "")
                  .collect(Collectors.joining());

          return s2;
      }
  ```
  ```

- Вариант последовательный
  - regex `(".*([r-][w-][x-]){3}$")` распаковывается так
  - .* любой символ ноль или несколько
  - (abc){3} повторение группы abc 3 раза, то есть abcabcabc
  - [ab] любой из символов a или b
  - ([r-][w-][x-]){3} подойдут комбинации rwx-w-r-x, -wxrw---x и так далее
  - $ конец строки
  - таким образом это полный шаблон строки chmod на 9 символов с любым началом
  - s.matches() использует regex чтобы определить подходит ли строка под шаблон
  - substring() вытаскивает строку длиной 9 символов с конца строки, остальное отбрасывает
  - rwx.replaceAll() заменяет любой из символов r,w,x на "1"
  - rwx.replaceAll() заменяет любой из символов - на "0"
  - tocharArray() создает массив символов из строки char[]
  - Integer.parseInt() перекодирует строку из двоичного в десятичный, 111 в 7
  - String.valueOf() перекодирует число 7 в строку "7"
  - range() IntStream.range()
  - skip() пропускает заданное число элементов потока и оставляет только 9
  - map() использует Function(int,int) интерфейс как аргумент
  - в данном случчае идет перекодировка символов char[] потока в целые числа 0 и 1
  - mapToObj() переделывает поток из целых чисел в строки, по одной на каждое число "1", "0"
  - collect() joininig() объединяет строки чисел в одну строку,на выходе строка вида "111101101"
  - regex `("(?<=\\G...)"` это особая конструкция, распаковывается так
  - (?<=x) lookupAhead встать на позицию с символа x и далее
  - \\G... найти границу \\G за предыдущим совпадением (...)
  - в результате находит границу через каждые (...) три любых символа
  - .split() метод .split(**"(?<=\\G...)"**)) разбивает строку на подстроки в три символа
  - \\G это boundary matcher как это точно работает
  - на выходе поток из трех строк "111","101","101"
  - map() использует Function(String,String) интерфейс как аргумент
  - перекодирует тройки двоичных чисел в десятичные и затем в строки "111">"7"
  - collect() joininig() объединяет строки чисел в одну строку, "7","5","5" > "755"
  - в результате "rwxr-xr-x">> char[] ‘r’,’w’.. >> int[] 1,1.. >>"1","1".. >>"111","101"... >>"7","5".. >>"755"
- Пример. реализация
- ```
  private static String getDigits2(String s) {
      if (!s.matches(".*([r-][w-][x-]){3}$")) {
          throw new IllegalArgumentException("Achtung!");
      }

      char[] chars = s.substring(s.length() - 9)
              .replaceAll("[rwx]", "1")
              .replaceAll("-", "0").toCharArray();

      return IntStream.range(0, 3)
              .map(i -> (chars[i * 3] - '0') * 4 + (chars[i * 3 + 1] - '0') * 2 +
  ```
- ```
                                                      chars[i * 3 + 2] - '0')
              .mapToObj(String::valueOf)
              .collect(Collectors.joining());
  }
  ```
  - 

## Programming Assignment

- Programming Assignment 2 задания **m55/src**
  - реализовать Example Implementation
  - реализовать Example Implementation

## Example Implementation

- Example Implementation
  - реализовать класс Example
- Описание алгоритма
  - описание алгоритма
- Пример. реализация Example Implementation
  - описание деталей
- ВНИМАНИЕ. Example Implementation ОПИСАНИЕ пункта
-

# TRICKS

## GIT

- GIT
  - клонировать с github — git clone https://github.com/v777779/jup
  - удалить папку — git rm –r stage_one_temp
  - git commit –m "removed stage_one_temp folder"
  - git push
  - удалить физически — rmdir /s /q stage_one_temp
  - добавить файл — git add stage_one/_exam_code/*
  - git commit -m "added stage_one final code"
  - git push
  - восстановление всего — git checkout *
  - git checkout stage_one/v3
- Если не добавляет файл *.jar
  - значит работает .gitignore — git add * -f
  - форсировать добавление
  -