Collectors is **one of the utility class in JDK which contains a lot of utility functions**. It is mostly used with Stream API as a final step.

ToList:

**public** City(String name, **double** temperature)

    {

**this**.name = name;

**this**.temperature = temperature;

    }

System.out.println(prepareTemperature().stream()

                 .filter(f -> f.getTemperature() > 10)

                 .map(f -> f.getName())

                 .collect(Collectors.toList()));

ToSet: No Duplicate

System.out.println(prepareTemperature()

                .stream()

                .filter(f -> f.getTemperature() > 10)

                .map(f -> f.getName())

                .collect(Collectors.toSet()));

        }

ToCollection:

.collect(Collectors.toCollection(List::**new**)));

ToMap: toMap() methods ask for following arguments:

K - Key function; U - Value Function;

BinaryOperator(optional); Supplier(Optional; merge function?)

// Here, the name and the temperature

        // are defined as the type City

        System.out.println(prepareTemperature()

        .stream().filter(city -> city.getTemperature() > 10)

        .collect(Collectors.toMap(

         City::getName, City::getTemperature,

         (key, identicalKey) -> key)));

Collectors.groupingBy .. Collectors.collectingAndThen

System.out.println(prepareTemperature()

     .stream()

     .collect(Collectors.groupingBy(

                            City::getName,

                            Collectors.collectingAndThen(

                                    Collectors.counting(),

                                    f -> f.intValue()))));

Word counting:

System.***out***.println(

Arrays.*stream*(*COUNTRY\_NAMES*)

.collect(Collectors.*groupingBy*(s->s,

Collectors.*counting*())));

**Collector mapping(Function mapper, Collector downstream):**

**// ToList or ToSet**

System.out.println(prepareTemperature()

                .stream()

                .collect(Collectors.groupingBy(

                            City::getName,

                            Collectors.mapping(

                                City::getTemperature,

                                Collectors.toList()))));

Map<BlogPostType, String> postsPerType = posts.stream() .collect(Collectors.groupingBy(BlogPost::getType, Collectors.mapping(BlogPost::getTitle, joining(", ", "Post titles: [", "]"))));

Collectors.partitioningBy:

System.out.println(prepareTemperature()

                .stream()

                .collect(Collectors.partitioningBy(

                  city -> city.getTemperature() > 15)));

[Collector averagingDouble(ToDoubleFunction mapper)](https://www.geeksforgeeks.org/java-collectors-averagingdouble-with-examples/)**:**

[Collector averagingInt(ToIntFunction mapper)](https://www.geeksforgeeks.org/java-8-collectors-averagingint-in-java-with-examples/)**:**

[Collector averagingLong(ToLongFunction mapper):](https://www.geeksforgeeks.org/java-collectors-averaginglong-tolongfunction-mapper-with-examples/)

**static** List<Employee> employeeList

      = Arrays.asList(**new** Employee("Tom Jones", 45, 15000.00,190),

**new** Employee("Tom Jones", 45, 7000.00,220),

**new** Employee("Ethan Hardy", 65, 8000.00,1008),

**new** Employee("Nancy Smith", 22, 10000.00,5),

**new** Employee("Deborah Sprightly", 29, 9000.00,45));

//Using Collectors.averagingInt()

    Double avgAge = employeeList

        .stream()

        .collect(Collectors.averagingInt(Employee::getAge));

     System.out.println("Average age using Collectors.averagingInt: " + avgAge);

    //Using Collectors.averagingLong()

    Double avgLeaves = employeeList

        .stream()

        .collect(Collectors.averagingLong(Employee::getLeaves));

    System.out.println("Average leaves using Collectors.averagingLong: " + avgLeaves);

    //Using Collectors.averagingDouble()

    Double avgSalary = employeeList

        .stream()

        .collect(Collectors.averagingDouble(Employee::getSalary));

    System.out.println("Average salary using Collectors.averagingDouble: " + avgSalary)

**Integer** sum = items.stream()

.map(x -> x.getPrice())

.reduce(0, ArithmeticUtils::add);

**Integer** sum = items.stream()

.map(x -> x.getPrice())

.reduce(0, Integer::sum);

**Integer** sum = items.stream()

.map(item -> item.getPrice())

.reduce(0, (a, b) -> a + b);

**Integer** sum = items.stream()

.map(x -> x.getPrice())

.collect(Collectors.summingInt(Integer::intValue));

items.stream()

.mapToInt(x -> x.getPrice())

.sum();

If it’s string format:

**String** string = "Item1 10 Item2 25 Item3 30 Item4 45"; **Integer** sum = Arrays.stream(string.split(" ")) .filter((s) -> s.matches("\\d+")) .mapToInt(Integer::valueOf) .sum();

[Collector<T, ?, Optional> maxBy(Comparator comparator)](https://www.geeksforgeeks.org/java-collectors-maxbycomparator-comparator-with-examples/)**:**

**Collector reducing(BinaryOperator op):**

**Collector reducing(T identity, BinaryOperator op)**

**FlatMap():**

List<Detail> details = **new** **ArrayList**<>();

details.add(**new** **Detail**());

Stream<String> stream = details.stream()

.flatMap(detail -> detail.getParts().stream());

Map(): more complicated example:

Map from one list of custom class to another custom class.

List<StaffPublic> result = staff.stream().map(temp -> {

StaffPublic obj = new StaffPublic();

obj.setName(temp.getName());

obj.setAge(temp.getAge());

if ("mkyong".equals(temp.getName())) {

obj.setExtra("this field is for mkyong only!");

}

return obj;

}).collect(Collectors.toList());

//Compare by Id

Comparator<Employee> compareById\_1 = Comparator.comparing(e -> e.getId());

Comparator<Employee> compareById\_2 = (Employee o1, Employee o2) -> o1.getId().compareTo( o2.getId() );

//Compare by firstname

Comparator<Employee> compareByFirstName = Comparator.comparing(e -> e.getFirstName());

//how to use comparator

Collections.sort(employees, compareById);