Learning Goals

Click to Continue

This session will help you to understand about,

- What is Java Streams?
- Concepts of streams.
- The aggregate functions used in streams.

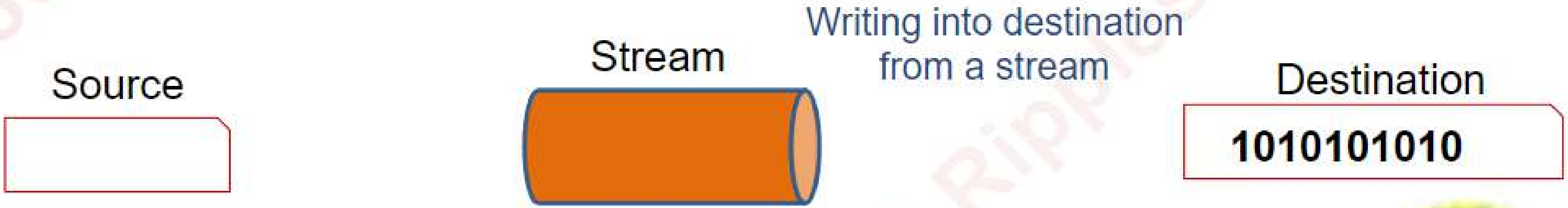
What are streams?

Click to Continue

Stream represents a sequence of data.

- Streams are used to read data from a source , perform a aggregate operation (or) some transformation and provide output.
- Streams can be created from sources such as file, arrays or collections.
- Output can again be a collection, array or file object.

Stream Illustration:



Stream API

Click to Continue

Source	Method	Description
Collections/ Array/IO	stream()	Returns the data of the source object as a stream.
Stream	filter()	Filters a sequence of data and returns a stream.
Stream	collect()	Converts a stream to a list, string ,map or set
Stream	map()	Transforms each data of the stream and returns a stream of data.
Stream	count()	Returns the number of elements in a stream.
Stream	reduce()	Reduces the data of stream into one element based on a accumulation function.

Try it out : Stream Demo

Click to Continue

Assume there is a list of fruits,

```
List<String> l = new ArrayList<>();
l.add("Apple");
l.add("Coconut");
l.add("Grapes");
l.add("Cherry");
l.add("Wat");
```

Map Example:

The below code snip charac form a new list.

This converts the list to stream.

The map function to which we pass a lambda function.

```
List<Character> l1 = l.stream().map((s) ->
s.charAt(0)).collect(Collectors.toList());
l1.forEach(System.out::println);
```

Use For each to print the list.

The lambda function extracts the first char of each fruit.

The collectors interface is used to convert the stream back to a list.

Try it out : Stream Demo

Click to Continue

Map Example: The below code illustrates how to specify two statements in lambda function and pass it to map function. This takes the first character of each fruit and increments it by 1 and adds it to a new list.

```
l1 = l.stream().map((s) -> {char
var = s.charAt(0);return
++var;}).collect(Collectors.toList
());
l1.forEach(System.out::println);
```

The lambda function with more than one statement. Which increments the first character of each string.

Pipelining Example: Here the output of one stream method will be streamlined to another stream method. In the below example, the map output is streamed to filter method and then the final output is converted to a list.

```
List<String> l2 = l.stream().map((s)-
>s.toUpperCase()).filter((c) -> c.contains("C"))
.collect(Collectors.toList());
l2.forEach(System.out::println);
```

Here the map output which is a stream is passed to filter method. Which filters all the fruits which does not have the letter 'C'.

Try it out : Stream Demo

Click to Continue

Limit Example: This limits the stream to a specified number of elements.

```
List<String> l3 =
l.stream().limit(3).collect(Collectors.toList());
l3.forEach(System.out::println);
```

This limits the stream to the first 3 elements and converts it to a list.

Skip Example: Here it skips the first N elements and picks the rest and process.

```
l3 = l.stream().skip(2).collect(Collectors.toList());
l3.forEach(System.out::println);
```

Skips the first 2 elements.

Sort Example: This sorts the stream element based on the natural ascending order.

```
l4 = l.stream().sorted().collect(Collectors.toList());
l4.forEach(System.out::println);
```

Sorts in ascending order.

Try it out : Stream Demo

Click to Continue

Sorts Example: This sorts the stream elements in descending order.

```
List<String> l4 =
l.stream().sorted(Comparator.reverseOrder()).collect(Collecto
rs.toList());
l4.forEach(System.out::println);
```

Comparator interface is used to reverse the order (Descending) of elements.

Map conversion: This converts the list to a Map.

```
Map<String, String> m =
l.stream().collect(Collectors.toMap((c) -> c +
"Key", (y) -> y + "value"));
System.out.println("Map ->" + m);
```

```
OUTPUT:
{CoconutKey=Coconutvalue,
AppleKey=Applevalue,
WatKey=Watvalue,
GrapesKey=Grapesvalue,
CherryKey=Cherryvalue}
```

This converts the list to a Map.

Try it out : Stream Demo

Click to Continue

Assume there is a map of countries,

```
Map<String, String> m1 = new
HashMap<>();
m1.put("India", "New Delhi");
m1.put("US", "Washington");
m1.put("Russia", "Moscow");
m1.put("Srilanka", "Colombo");
```

Takes each entry set of the map. Entry set is a row of a map.

The below code snippet creates a map from a new map based on some rules.

```
Map<String, String> m2 = m1.entrySet().stream().filter((x) ->
x.getKey().contains("a"))
.collect(Collectors.toMap(i -> i.getKey(), i ->
i.getValue()));
System.out.println("Country Map ->" + m2);
```

The filtered map then is converted to a map with key as original map's key. Value as original maps value.

Filters the map based on the rule, if key contains a character "a"

```
OUTPUT:
Country Map ->{Srilanka=Colombo, Russia=Moscow, India=New Delhi}
```

Try it out : Stream Demo

Click to Continue

Map to String: This converts the map to a string by concatenating the key and value with – and finally combining all rows delimited with '|'.

```
String s = m1.entrySet().stream().map((k) ->
k.getKey().concat("-").concat(k.getValue()))
.collect(Collectors.joining("|"));
```

Concatenates the key and value with -

Collector creates a string , concatenating the rows with delimiter |.

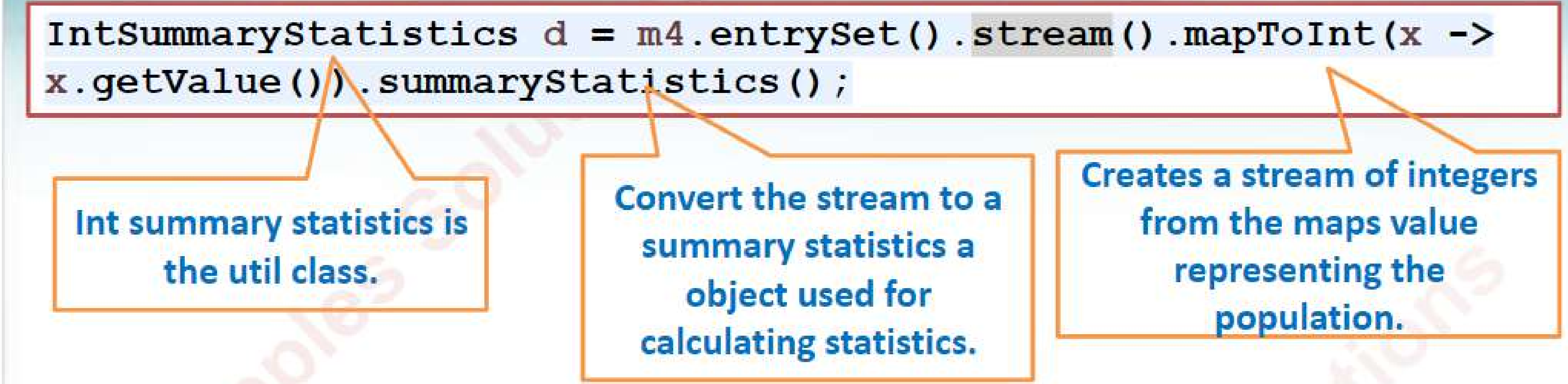
```
OUTPUT: Srilanka-Colombo|US-Washington|India-New Delhi|Russia-Moscow
```

Statistics Functions: Let us now see some statistics function. Assume the following map with country population,

```
Map<String, Integer> m4 = new
HashMap<>();
m4.put("India", 10000);
m4.put("US", 20000);
m4.put("Russia", 30000);
m4.put("Srilanka", 40000);
```

Try it out : Stream Demo

Click to Continue



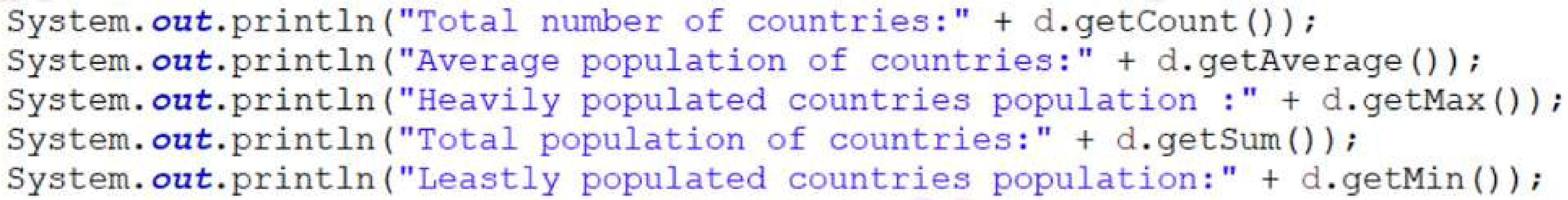
```
IntSummaryStatistics d = m4.entrySet().stream().mapToInt(x ->
x.getValue()).summaryStatistics();
```

Int summary statistics is the util class.

Convert the stream to a summary statistics a object used for calculating statistics.

Creates a stream of integers from the maps value representing the population.

The below code snippet prints the statistics using the API's in IntSummaryStatistics.
Example: max, media, average.



```
System.out.println("Total number of countries:" + d.getCount());
System.out.println("Average population of countries:" + d.getAverage());
System.out.println("Heavily populated countries population :" + d.getMax());
System.out.println("Total population of countries:" + d.getSum());
System.out.println("Leastly populated countries population:" + d.getMin());
```

Try it out : Stream Demo

Click to Continue

Assume we have a list of strings and we need to remove the duplicates

```
List<String> l5 = new
ArrayList<>();
l5.add("Apple");
l5.add("Coconut");
l5.add("Grapes");
l5.add("Grapes");
l5.add("Cherry");
l5.add("Coconut");
l5.add("Coconut");
l5.add("Cherry");
```

Distinct method used to remove duplicate items from stream.

```
List responseList =
l5.stream().distinct().collect(Collectors.toList());
System.out.println("Duplicate removed from list and converted
to string:{" + responseList);
```

```
Output: [Apple, Coconut, Grapes, Cherry]
```

Try it out : Stream Demo

Click to Continue

Assume we have a list of integers and we need to reduce it to a single value, say sum of all even numbers

```
List<Integer> l6 = new
ArrayList<>();
l6.add(2);
l6.add(3);
l6.add(5);
l6.add(8);
l6.add(10);
```

Reduce function, used to reduce the data elements into a single value.

0 is the initial value or default value the reduce function should return if the stream is empty.

```
int result = l6.stream().reduce(0, (intermediateValue,
elementValue) -> {
intermediateValue = (elementValue % 2 == 0) ?
intermediateValue + elementValue : intermediateValue;
return intermediateValue;
});
```

The second argument of reduce function is the lambda expression which is the accumulator, again accepts two arguments. This checks if the given value is even number , if yes adds the value to the intermediate value. Which finally totals to the sum of all even numbers

```
Output: 20
```