**New features in Java 8**

Java 8 supports functional programming. The features that support that are:

1. Lambda Expressions
2. Streams API
3. Optional Object

Let’s examine the following code to understand the concepts:

Map <String, List<String>> phoneNumbers = new HashMap <String, List<String>>();

phoneNumbers.put("John Lawson", Arrays.asList("3232312323", "8933555472"));

phoneNumbers.put("Mary Jane", Arrays.asList("12323344", "492648333"));

phoneNumbers.put("Mary Lou", Arrays.asList("77323344", "938448333"));

Map <String, List<String>> filteredNumbers = phoneNumbers.entrySet().stream()

.filter(x - > x.getKey().contains("Mary"))

.collect(Collectors.toMap(p - > p.getKey(), p - > p.getValue()));

filteredNumbers.forEach((key, value) - > {

System.out.println("Name: " + key + ": ");

value.forEach(System.out::println);

});

* Imperative programming:

Much like imperative mood in the speech, it tells the program to perform some command. It also changes the state of the program (i.e. variable values are changed before and after the execution)

* Functional programming:

It is a subset of declarative programming paradigm. It also has commands but the commands are treated more like mathematical functions. Their task is to take some input, perform some action and return a result. The essential part here is that they *do not modify* the state of the program. The input variable(s) remain unchanged, and the returned result is always a new variable.

Methods as used in the middle section:

* entrySet: To get a set of entries each consisting of a key and a value
* stream: To get a sequence of elements supporting sequential and parallel aggregate operations
* filter: To filter the items in the stream with some criteria. The criteria are expressed as x - > x.getKey().contains(“Mary”)

This is called a lambda expression. This concept is also introduced in Java 8. Here it acts like a predicate, a Boolean function. Its goal is to evaluate the filtering criteria and tell whether each specific item in the collection (stream) should be kept or removed.

* collect: To take the elements of the stream (java.util.Map.Entry) and converts them back into a regular collection.

As you can see, the methods are chained, and they all belong to the Stream interface, so this suggests that each of them returns the stream to be accessible for subsequent methods in the invocation process.