INTRODUCING “JAVA 8”

Collections Enhancements

Collections Enhancement

- bulk data operations
- serial and parallel implementation
- parallel array sorting

Collection API Improvements

- Iterator default method forEachRemaining(Consumer action) to perform the given action for each remaining element until all elements have been processed or the action throws an exception.
- Collection default method removeIf(Predicate filter) to remove all of the elements of this collection that satisfy the given predicate.
- Collection spliterator() method returning Spliterator instance that can be used to traverse elements sequentially or parallel.
- Map replaceAll(), compute(), merge() methods.

Example :

```
//Sample list
List<Integer> data = new ArrayList<Integer>();
for(int i=1;i<=10;i++) data.add(i);

//Remove all EVEN values
data.removeIf((n)-> n%2==0);

//Print all elements
Iterator<Integer> it = data.iterator();
it.forEachRemaining(System.out::println);
```

Example:

```
//Sample list
List<Integer> data = new ArrayList<Integer>();
for(int i=1;i<=100;i++) data.add(i);

Spliterator it=data.spliterator();
Spliterator it2 = it.trySplit();
//Print 51 to 100
it.forEachRemaining(System.out::println);
```

Example:

```
Map<Integer,String> data = new
  HashMap<Integer,String>();
data.put(1,"Abc");
data.put(2,"Xyz");
data.put(3,"Pqr");
data.put(4,"KBC");

data.replaceAll((k,v)->(k%2==0)?"PPP":"ZZZ");
```

Bulk Data operations

- A new java.util.stream has been added in Java 8 to perform filter/map/reduce like operations with the collection.
- Stream API will allow sequential as well as parallel execution.
- Collection interface has been extended with *stream()* and *parallelStream()* default methods to get the Stream for sequential and parallel execution.
- We have a complete chapter dedicated on Stream API

Parallel Array Sort

```
int []data = {2,6,21,8,1,15};
Arrays.parallelSort(data);

for(int n: data)
      System.out.println(n);
```