**Print numbers sequentially by multiple threads**

**class** Worker **implements** Runnable {  
  
 BlockingQueue<Integer> **q** = **new** LinkedBlockingQueue<Integer>();  
 Worker **next** = **null**; *// next worker in the chain* **public void** setNext(Worker t) {  
 **this**.**next** = t;  
 }  
  
 **public void** accept(**int** i) {  
 **q**.add(i);  
 }  
  
 @Override  
 **public void** run() {  
 **while** (**true**) {  
 **try** {  
 **int** i = **q**.take(); *// blocks till it receives a number* System.***out***.println(Thread.*currentThread*().getName() +**"\t"**+ i);  
  
 Thread.*sleep*(1000); *// delay to slow the printing* **if** (**next** != **null**) {  
 **next**.accept(i + 1); *// pass the next number to the next worker* }  
  
 } **catch** (InterruptedException e) {  
 System.***err***.println(Thread.*currentThread*().getName() + **" interrrupted."**);  
 }  
 }  
 }  
}

**public void** printInSequence(**int** numOfThreads) {  
  
 ArrayList<Worker> threadList = **new** ArrayList<>();  
  
 Worker previous = **null**;  
 Worker current = **null**;  
 Worker first = **null**;  
*// Create the workers and chain them in a round robin fashion* **for** (**int** i = 0; i < numOfThreads; i++) {  
 current = **new** Worker();  
 **if** (previous != **null**) {  
 previous.setNext(current);  
 } **else** {  
 first = current;  
 }  
 previous = current;  
 threadList.add(current);  
 }  
 current.**next** = first;  
 **for** (**int** i = 1; i <= numOfThreads; i++) {  
 Thread t = **new** Thread(threadList.get(i-1),**"Thread -"**+i);  
 t.start();  
 }  
  
 *// Seed the first current* first.accept(1);  
 }  
}

**Sort Map by value**

*/\*\*  
 \* Steps to sort HashMap by values  
 One difference between sorting HashMap by keys and values is that it can contain duplicate values by not duplicate keys.  
 You cannot use TreeMap here because it only sort entries by keys. In this case you need to :  
  
 1. Get all entries by calling entrySet() method of Map  
 2. Create a custom Comparator to sort entries based upon values  
 3. Convert entry set to list  
 4. Sort entry list by using Collections.sort() method by passing your value comparator  
 5. Create a LinkedHashMap by adding entries in sorted order.  
  
  
 \*/***public class** SortMapByValue {  
  
 **public static void** main(String[] args) {  
  
 HashMap<String, Integer> map = **new** HashMap<>();  
 map.put(**"a"**,10);  
 map.put(**"f"**,5);  
 map.put(**"b"**,25);  
 map.put(**"i"**,3);  
  
 System.***out***.println(**new** SortMapByValue().sortByValue(map));  
 }  
  
 **public** HashMap<String, Integer> sortByValue(HashMap<String, Integer> ipMap) {  
  
 LinkedHashMap<String, Integer> opMap = **new** LinkedHashMap<>();  
  
 Set<Map.Entry<String, Integer>> entries = ipMap.entrySet();  
 ArrayList<Map.Entry<String, Integer>> entriesList = **new** ArrayList<>(entries);  
 Collections.*sort*(entriesList,**new** EntrySetComparator());  
  
 **for** (Map.Entry<String,Integer> entry:entriesList) {  
  
 opMap.put(entry.getKey(),entry.getValue());  
 }  
 **return** opMap;  
 }  
  
 **class** EntrySetComparator **implements** Comparator<Map.Entry<String, Integer>>  
 {  
  
 @Override  
 **public int** compare(Map.Entry<String, Integer> o1, Map.Entry<String, Integer> o2) {  
 **return** o1.getValue()-o2.getValue();  
 }  
 }  
}

/\*

In case of Comparable, method is compareTo(Object o) -> answer should be **this-o** for ascending

\*/