**1- What is Idempotency??**

Idempotency is an important thing while building a fault-tolerant RESTful API.

No matter, how many times you call, the result is the same. This is the meaning of Idempotency.

For example, you had hit post request and it fails somehow. We need to figure where exactly is the failure.

Is it at sending request or at updating db.?

But, for put request, you can hit multiple times without worrying about , if it actually failed or not.

**Options, get, head, delete and put are Idempotent**

Post and patch are not Idempotent.

**2- Count of days remaining for the next day with higher temperature??**

Input: arr[] = {73, 74, 75, 71, 69, 72, 76, 73}

Output: {1, 1, 4, 2, 1, 1, -1, -1}

* Iterate over the everyday temperature of the given array arr[] using the current index.
* If the stack is empty, push the current index to the stack.
* If the stack is not empty then do the following:

1. If the temperature at the current index is lesser than the temperature of the index at top of the stack, push the current index.
2. If the temperature at the current index is greater than the temperature of the index at top of the stack, then update the no of days to wait for warmer temperature as:

**current index – index at top of the stack**

* Pop the stack once the number of days has been updated in the output list.

Repeat the above steps for all the indices in the stack that are lesser than the temperature at the current index.

* **peek() method** is used to return the top element from this Stack and it retrieves elements without removing.

class DailyTemperatureProblem {  
 static void dailyTemperatures(int[] temperatureArray) {  
 int n = temperatureArray.length;  
  
 // To store the answer  
 int[] daysOfWait = new int[n];  
 Arrays.*fill*(daysOfWait, -1);  
  
 Stack<Integer> s = new Stack<>();  
  
 // Traverse all the temperatures  
 for (int i = 0; i < n; i++) {  
 // Check if current index is the next warmer temperature of any previous indexes  
 while (!s.isEmpty() && temperatureArray[s.peek()] < temperatureArray[i]) {  
 daysOfWait[s.peek()] = i - s.peek();  
 s.pop();  
 }  
 s.push(i);  
 }  
  
 // Print waiting days  
 for (int i = 0; i < n; i++) {  
 System.*out*.print(daysOfWait[i] + " ");  
 }  
 }  
  
 // Driver Code  
 public static void main(String[] args) {  
 // Given temperatures  
 int[] arr = {73, 74, 75, 71, 69, 72, 76, 73};  
 // Function call  
 *dailyTemperatures*(arr);  
 }  
}

**3- Print numbers using 3 Threads**

public class PrintNumbersUsingThreeThreads {  
 final static int *MAX\_NUMBERS* = 10;  
  
 public static void main(String[] args) {  
 // shared object  
 PrintNumbersUsingThreeThreads obj = new PrintNumbersUsingThreeThreads();  
 // Creating 3 threads  
 Thread t1 = new Thread(new NumberRunnable(obj, 0), "T1");  
 Thread t2 = new Thread(new NumberRunnable(obj, 1), "T2");  
 Thread t3 = new Thread(new NumberRunnable(obj, 2), "T3");  
 t1.start();  
 t2.start();  
 t3.start();  
 }  
}  
  
class NumberRunnable implements Runnable {  
 PrintNumbersUsingThreeThreads obj;  
 int threadNumber;  
 static volatile int *number* = 0;  
  
 NumberRunnable(PrintNumbersUsingThreeThreads obj, int result) {  
 this.obj = obj;  
 this.threadNumber = result;  
 }  
  
 @Override  
 public void run() {  
 synchronized (obj) {  
 while (*number* < PrintNumbersUsingThreeThreads.*MAX\_NUMBERS* - 2) {  
 while (*number* % 3 != threadNumber) {  
 try {  
 obj.wait();  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 System.*out*.println(Thread.*currentThread*().getName() + " - " + ++*number*);  
 obj.notifyAll();  
 }  
 }  
 }  
}