Code looks something like this (still using Java Threads)

```
class SumThread extends java.lang.Thread {
  int lo; int hi; int[] arr; // fields to know what to do
  int ans = 0; // result
  SumThread(int[] a, int l, int h) { … }
  public void run(){ // override
    if(hi – lo < SEQUENTIAL_CUTOFF)
      for(int i=lo; i < hi; i++)
        ans += arr[i];
    else {
      SumThread left = new SumThread(arr,lo,(hi+lo)/2);
      SumThread right= new SumThread(arr,(hi+lo)/2,hi);
      left.start();
      right.start();
      left.join();
      right.join();
      ans = left.ans + right.ans;
    }
  }
}
```

```
int sum(int[] arr){ // just make one thread!
   SumThread t = new SumThread(arr,0,arr.length);
   t.run();
   return t.ans;
}
```


Fork Join Framework Version: (missing imports)

pollev.com/artliu

```
class SumTask extends RecursiveTask<Integer> {
  int lo; int hi; int[] arr; // fields to know what to do
  SumTask(int[] a, int l, int h) { … }
  protected Integer compute(){// return answer
    if(hi – lo < SEQUENTIAL_CUTOFF) {
      int ans = 0; // local var, not a field
      for(int i=lo; i < hi; i++)
        ans += arr[i];
      return ans;
    } else {
      SumTask left = new SumTask(arr,lo,(hi+lo)/2);
      SumTask right= new SumTask(arr,(hi+lo)/2,hi);
      left.fork(); // fork a thread and calls compute
      int rightAns = right.compute();//call compute directly
      int leftAns  = left.join(); // get result from left
      return leftAns + rightAns;
    }
  }
}
```

```
static final ForkJoinPool POOL = new ForkJoinPool();
int sum(int[] arr){
   SumTask task = new SumTask(arr,0,arr.length)
   return POOL.invoke(task);
   // invoke returns the value compute returns
}
```