```
package Algorithm;

import Items.Job;
import Items.Queue;

/**
 *
 *
 * "preemptive priority" works like "shortest remaining time" except
 * that the jobs in the ready queue are ordered by priority
 */
public class Priority2 extends MyAlgorithm{

   /**
    * pass the work queue to super class to initialize lists
    * @param workQueue queue of lists to be worked on
    */
   public Priority2(Queue workQueue)
   {
      super(workQueue);
   }

   /**
    * shows what happen in a single step when using this algorithm
    * @param simulationTime current time of this simulation
    * @return job the CPU was working on
    */
   @Override
   public Job nextStep (int simulationTime)
   {
      updateReadyQueue(simulationTime); // add newly arrived jobs to the ready queue
      /* return the current job to the ready queue to make sure that the job with
         higher priority will be worked first*/
      if(simulationTime!=0 && currentJob.getRemainTime() !=0) {readyQueue.addJob(currentJob); }
      if(readyQueue.size() > 1){ readyQueue.OrderedByPriority();}
      if(readyQueue.isEmpty()) { return null;}
      setCurrentJob();  // move the first job in the ready queue to be the current working job
      return workInCPU(simulationTime);
   }

}
```