```
package Algorithm;

import Items.Job;
import Items.Queue;

/**
 *
 * "Round Robin" algorithm let every job to be processed by the CPU for a
 * certain time "quantum time" then replace it the next job in the ready queue.
 * when a job finishes its quantum time it get to the end of the ready queue.
 *
 * the program here works step by step so we have to make a variable (process time)
 * to track what's the time left form the quantum time for a specific job.
 *
 */
public class RR extends MyAlgorithm{

   private int Quantum; // quantum time of the algorithm
   private int processTime; // the remaining of the quantum time for a specific job

   /**
    * pass the work queue to super class to initialize lists
    * @param workQueue queue of lists to be worked on
    */
   public RR(Queue workQueue , int quantum)
   {
      super(workQueue);
      this.Quantum = quantum;  // set the wanted quantum time
   }

   /**
    * 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
      if(!busy) // if CPU is not processing a job ( RR is non-preemptive algorithm)
      {
         if(simulationTime!=0 && currentJob.getRemainTime() !=0)
         {readyQueue.addJob(currentJob); } // if job is not finished add it to the ready queue again
         if(readyQueue.isEmpty()) {return null;}
         processTime = Quantum;  // restart quantum time for the new job
         busy = true;
         setCurrentJob();  // move the first job in the ready queue to be the current working job
      }
```

```
    return workInCPU(simulationTime);
  }

  /**
   * work the current job in the CPU for one simulation time step
   * @param simulationTime current time of the simulation
   * @return the current job the CPU is working on
   */
  @Override
  protected Job workInCPU(int simulationTime)
  {
    currentJob.jobWorked(simulationTime);
    processTime--;  // the rest of quantum time for this time of working of the job
    if(processTime == 0 || currentJob.getRemainTime() ==0 )
      {busy = false;} // if job is finished or round time is finished, make CPU not busy
    return currentJob;
  }

}
```