```
package Algorithm;

import Items.Job;
import Items.Queue;

/**
 *
 * "Shortest job first" algorithm is the same as "First come first served"
 * except that the ready queue is reordered by shortest burst time job
 * every time a new job arrives.
 *
 * shortest job first is non-preemptive algorithm.
 */
public class SJF extends MyAlgorithm{

   /**
    * pass the work queue to super class to initialize lists
    * @param workQueue queue of lists to be worked on
    */
   public SJF(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
      if(readyQueue.size() > 1) { readyQueue.OrderedByShortest();}  // order ready queue by shortest
time burst
      if(!busy)   // if CPU is not processing a job ( SJF is non-preemptive algorithm)
      {
         if(readyQueue.isEmpty()) {return null;}
         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);
        if(currentJob.getRemainTime() == 0) {busy = false;} // if job is finished make CPU not busy
        return currentJob;
    }

}
```