```
package Algorithm;

import Items.Job;
import Items.Queue;

/**
 *
 * "MyAlgorithm" is class used to be inherited by all scheduling algorithms classes
 * for the sake of polymorphism. it also holds mutual variables and methods between
 * all algorithm classes.
 */
public abstract class MyAlgorithm {

   protected Queue list ;  // list of all jobs (havn't worked yet)
   protected Queue readyQueue;  // list of all jobs in the ready queue
   protected Job currentJob;  // current job that simulation is working on now
   protected boolean busy ;  // indicates whether cpu is busy or not (non-preemptive jobs)

   /**
    * initializes list and ready queues and take a copy of the simulation queue
    * to work on and arrange it by arrive time it becomes easy for the scheduling
    * algorithm to define which job came first.
    * @param workQueue queue of jobs from the simulation
    */
   public MyAlgorithm(Queue workQueue)
   {
      readyQueue = new Queue(workQueue.size()); // intialize ready queue size
      currentJob = new Job(9);  //initialize an empty job to avoid null pointer exception
      busy = false;  // set busy to default
      list = workQueue.getCopy(); // copy simulation queue to the algorithm main queue
      list.OrderedByArrive();  // order algorithm queue by arrive time
   }

   /**
    * abstract method needs to be override in inherited classes.
    * implements how the scheduling algorithm will behave on jobs in
    * one step.
    * @param simulationTime current time of the simulation
    * @return the job that the algorithm currently working on
    */
   public abstract Job nextStep (int simulationTime);

   /**
    * changes the data of the current job after being worked on
    * in the CPU in one simulation time step.
    * @param simulationTime current time of the simulation
    * @return current job the CPU is working on
    */
```

```
protected Job workInCPU(int simulationTime)
{
   currentJob.jobWorked(simulationTime);
   return currentJob;
}

/**
 * @return a separated copy of the ready queue
 */
public Queue getReadyQueue ()
{
   return readyQueue.getCopy();
}

/**
 * check whether the algorithm finished the simulation or not.
 * it check whether the main list and ready queue are empty and
 * the CPU is not working on any job
 * @return true if the simulation is finished
 */
public boolean isFinished()
{
   return (list.isEmpty() && readyQueue.isEmpty()  && !busy &&  currentJob.getRemainTime() == 0);
}

/**
 * add the newly arrived jobs to the ready queue
 * by comparing the arrive time with the simulation time.
 * @param simulationTime current time of the simulation
 */
protected void updateReadyQueue(int simulationTime)
{
   for (int i = 0 ; i<list.size() ; i++)
   {
      Job temp = list.getJob(i);
      if(temp.arrivalTime == simulationTime)  // if job arrived
      {
         readyQueue.addJob(temp);  // if job arrived then move it to the ready queue
         list.removeJob(i);   // remove the job from main job list
         i--; // removing reduces the size of the list by one
      }
   }
}

/**
 * move the first job at the ready queue
 * to be the current job for the CPU to work on.
 */
```

```
    protected void setCurrentJob()
    {
        currentJob = readyQueue.getJob(0);
        readyQueue.removeJob(0);
    }

}
```