```java
package Items;

import java.util.ArrayList;

/**
 * Queue is an array list of jobs, used to provide some helpful methods
 * to make using list of jobs much easier.
 */
public class Queue {

   private ArrayList<Job> mainList;  // Queue is a list of jobs
   private int number; // max number of jobs in the queue

   // <editor-fold defaultstate="collapsed" desc="constructors" >
   /**
    * create an empty queue with size for a specific number of jobs
    * @param number number of jobs
    */
   public Queue(int number)
   {
      mainList = new ArrayList<Job>(number);
      this.number = number;
   }

   /**
    * create a queue and fill it with a given list of jobs
    * @param list list of jobs for the queue
    */
   public Queue(ArrayList<Job> list)
   {
      mainList = new ArrayList<Job>(list.size());
      mainList.addAll(list);
   }
   // </editor-fold>

   /**
    * fill the queue with jobs with random data
    * @param number number of jobs in the queue
    */
   public void fill()
   {
      for(int i=0 ; i<number ; i++)
      {
         Job temp = new Job(i+1); // job number starts from 1 not 0
         mainList.add(i, temp);
      }
   }
```

```java
/**
 * @param num number of the job in the queue
 * @return selected job
 */
public Job getJob(int num)
{
   return mainList.get(num);
}

/**
 * remove selected job from the queue
 * @param num number of the selected job to be removed
 */
public void removeJob(int num)
{
   mainList.remove(num);
}

/**
 * add job to the end of the queue
 * @param job job to be added
 */
public void addJob(Job job)
{
   mainList.add(job);
}

/**
 * replace the job at a specific place of the queue
 * @param i place of job to be replaced
 * @param job new job to replace with
 */
public void set(int i , Job job)
{
   mainList.set(i, job);
}

/**
 * check if the queue is empty
 * @return true if empty
 */
public boolean isEmpty()
{
   return (mainList.isEmpty());
}

/**
 * @return the size of the queue
```

```
 */
public int size()
{
   return mainList.size();
}

/**
 * remove all the elements of the queue
 */
public void clearQueue()
{
   for(int i =0 ; i< mainList.size() ; i++)
   {
     mainList.remove(i);
   }
}

// <editor-fold defaultstate="collapsed" desc="order queue" >

/**
 * order the jobs inside the queue by arrive time
 */
public void OrderedByArrive()
{
   for(int i=0 ; i<mainList.size()-1 ; i++)
   {
      for(int j=i+1 ; j<mainList.size() ;j++)
      {
         Job j1 = mainList.get(i);
         Job j2 = mainList.get(j);
         if(j2.isFirst(j1))
         {
            mainList.set(i, j2);
            mainList.set(j, j1);
         }
      }
   }
}

/**
 *  order the jobs inside the queue by shortest burst
 */
public void OrderedByShortest()
{
   for(int i=0 ; i<mainList.size()-1 ; i++)
   {
      for(int j=i+1 ; j<mainList.size() ;j++)
      {
```

```
            Job j1 = mainList.get(i);
            Job j2 = mainList.get(j);
            if(j2.isShort(j1))
            {
               mainList.set(i, j2);
               mainList.set(j, j1);
            }
         }
      }
   }

   /**
    * order the jobs inside the queue by priority
    */
   public void OrderedByPriority()
   {
      for(int i=0 ; i<mainList.size()-1 ; i++)
      {
         for(int j=i+1 ; j<mainList.size() ;j++)
         {
            Job j1 = mainList.get(i);
            Job j2 = mainList.get(j);
            if(j2.isPrior(j1))
            {
               mainList.set(i, j2);
               mainList.set(j, j1);
            }
         }
      }
   }

   /**
    * order the jobs inside the queue by the shortest remaining time
    */
   public void OrderedByShortRemain()
   {
      for(int i=0 ; i<mainList.size()-1 ; i++)
      {
         for(int j=i+1 ; j<mainList.size() ;j++)
         {
            Job j1 = mainList.get(i);
            Job j2 = mainList.get(j);
            if(j2.isShortRemain(j1))
            {
               mainList.set(i, j2);
               mainList.set(j, j1);
            }
         }
```

```
    }
  }

  // </editor-fold>

  /**
   * create a copy of a queue with all jobs data
   * @param list queue to be copied
   * @return complete separated copy of the queue
   */
  public Queue getCopy ()
  {
    return new Queue(mainList);
  }

  /**
   * create a copy of a queue with only the start data of every job.
   * note: this is used for restarting the same simulation by taking
   * a copy of the start data of the queue to use it in another simulation.
   * @param list queue to be copied
   * @return clear copy of the queue
   */
  public Queue getClearCopy()
  {
    ArrayList<Job> list = new ArrayList<Job>(mainList.size());
    for(int i=0 ; i<mainList.size() ; i++)
    {
      Job temp = mainList.get(i).getClearCopyJob();  // copy only start data of the job
      list.add(temp);
    }
    return new Queue(list);
  }

  /**
   * Show queue content (every job details)
   * improved version of toString method but used for testing.
   * @param list Queue to show it's content
   */
  public void showQueue(int simulationTime)
  {
    if(mainList.isEmpty()){System.out.println("Empty Queue");  return;} // if queue is empty
    System.out.println("number of jobs " + mainList.size() );
    System.out.println("# "+" Arrive "+" Burst "+" Priority "+" Start "+" Wait "+" Remain "+" Finish "+"
Turn "+" % ");
    for(int i=0 ; i<mainList.size() ; i++) // show every job data
    {
      Job temp = mainList.get(i);
      System.out.print(temp.jobNumber + "    " + temp.arrivalTime + "      " + temp.burst +"      ");
```

```
            System.out.print(temp.priority + "      " + temp.getStart() + "      " +
temp.getWaitTime(simulationTime)+ "      ");
            System.out.print(temp.getRemainTime() + "       " + temp.getFinish() + "      " +
temp.getTurnaround(simulationTime)+ "    ");
            System.out.println(temp.getPercent());
        }
    }
}
```