```java
package Items;

import java.util.Random;

/**
 * " Job " is a class to represent CPU job and its variables
 * and some useful methods for making, comparing and coping CPU jobs.
 * and some setters and getters.
 */
public class Job {

   public int jobNumber;    // job ID
   public int arrivalTime;
   public int burst;
   private int start;
   public int priority;
   public boolean finished;  // show if job is finished or not
   private int finish;       // finish time
   private int remaining;

   // <editor-fold defaultstate="collapsed" desc="constructors" >

   /**
    * create a job with random data
    * @param jobNumber job number
    */
   public Job(int jobNumber)
   {
      this.jobNumber = jobNumber;
      finished = false;
      Random rand = new Random();
      if(jobNumber == 1) {arrivalTime =0;}    // first job always arrive at time = 0
      else {arrivalTime = rand.nextInt(30)+1;}
      /* random numbers limits are selected by try and error to make sure job GUI representation
       won't exceed the program screen limits*/
      burst = rand.nextInt(12)+1;
      priority = rand.nextInt(125)+1;
      finish = 0 ;
      remaining = burst;
   }

   /**
    * create a job with assigned data
    * @param jobNumber number of the job
    * @param arrive arrive time of the job
    * @param burst burst time of the job
    * @param prior priority of the job
    */
```

```
public Job(int jobNumber , int arrive , int burst , int prior)
{
   this.jobNumber = jobNumber;
   finished = false;
   arrivalTime = arrive;
   this.burst = burst;
   priority = prior;
   finish = 0 ;       // finish time takes the value of zero till the job is really finished
   remaining = burst;
}

// </editor-fold>

/**
 * calculate what happens to the job when it gets worked by the CPU
 * @param simulationTime simulation time since the whole simulation has started
 */
public void jobWorked(int simulationTime){
   if( burst == remaining)  // if this is the first time for the job to be worked
   { start = simulationTime;}
   remaining--;
   if(remaining == 0)  // if job is finished
   {
      finish = simulationTime + 1;
      finished = true;
   }
}

/**
 * create a copy of job with all of its data
 * @return different job but with same data
 */
public Job copyJob()
{
   Job temp = new Job(this.jobNumber);
   temp.arrivalTime = this.arrivalTime;
   temp.burst = this.burst;
   temp.finished = this.finished;
   temp.jobNumber = this.jobNumber;
   temp.priority = this.priority;
   temp.setStart(this.start);
   temp.setFinish(this.finish);
   return temp;
}

/**
 * create a copy with only the start data of the job
 * note: this is used for restarting the simulation
```

```
 * @return a copy of start state of the job
 */
public Job getClearCopyJob()
{
   Job temp = new Job(this.jobNumber);
   temp.arrivalTime = this.arrivalTime;
   temp.burst = this.burst;
   temp.priority = this.priority;
   temp.remaining = this.remaining;
   return temp;
}

// <editor-fold defaultstate="collapsed" desc="getters" >

/**
 * @return percent of the done part of the job
 */
public int getPercent() {
   return (int)((burst - getRemainTime())*100) / burst;
}

/**
 * calculate the wait time of the job
 * wait = turnaround - elapsed time.
 * @param SimulationTime simulation time since the whole simulation has started
 * @return waiting time of the job
 */
public int getWaitTime(int SimulationTime) {
   return (getTurnaround(SimulationTime) - (burst - getRemainTime()));
}

/**
 * @return the remaining time of the job
 */
public int getRemainTime(){
   return this.remaining;
}

/**
 * calculate the turnaround time of the job
 * requires the simulation time if the job hasn't finished yet
 * @param SimulationTime simulation time since the whole simulation has started
 * @return turnaround time of the job
 */
public int getTurnaround(int SimulationTime){
   if(finished){  // if job is finished
      return (finish - arrivalTime );
   }
```

```
    if(SimulationTime > arrivalTime){ // if job arrived but hasn't finished yet
      return (SimulationTime - arrivalTime);
    }
    return 0; // if job hasn't arrived yet
}

/**
 * @return finish time if the job is finished
 * if not, it will return zero
 */
public int getFinish(){
    if(finished)
    {
       return finish;
    }
    return 0;
}

/**
 * @return start time of the job
 */
public int getStart(){
    return start;
}

// </editor-fold>

// <editor-fold defaultstate="collapsed" desc="setters" >

/**
 * set remaining time of the job
 * @param remaining remaining time of the job
 */
public void setRemainTime(int remaining){
    this.remaining = remaining;
}

/**
 * set finish time of the job
 * @param finish finish time of the job
 */
public void setFinish(int finish){
    this.finish = finish;
}

/**
 * set start time of the job
 * @param start start time of the job
```

```
 */
public void setStart(int start){
   this.start = start;
}

// </editor-fold>

// <editor-fold defaultstate="collapsed" desc="compare jobs" >

/**
 * compare the arrive time
 * @param other other job
 * @return true if this job is the first
 */
public boolean isFirst(Job other) {
   if(this.arrivalTime == other.arrivalTime) // if both have the same arrive time
   {
      return (this.jobNumber < other.jobNumber);
   }
   return (this.arrivalTime < other.arrivalTime);
}

/**
 * compare the shortest burst time
 * @param other other job
 * @return true if this job has shorter burst
 */
public boolean isShort(Job other){
   if(this.burst == other.burst) // if both have the same burst time
   {
      return isFirst(other);
   }
   return (this.burst < other.burst);
}

/**
 * compare the priority
 * @param other other job
 * @return true if this job has higher priority (smaller prior number)
 */
public boolean isPrior(Job other){
   if(this.priority == other.priority) //  if both have the same priority
   {
      return isFirst(other);
   }
   return (this.priority < other.priority);
}
```

```
    /**
     * compare the remaining time
     * @param other other job
     * @return true if this job has shorter remaining time
     */
    public boolean isShortRemain(Job other){
        if(this.remaining == other.remaining) // if both have the same remaining time
        {
            return isFirst(other);
        }
        return (this.remaining < other.remaining);
    }

    // </editor-fold>



    /**
     * show this job data.
     * improved version of toString method but used for testing.
     */
    public void ShowData()
    {
        System.out.println("Showing job data");
        if(this == null) {System.out.println("Empty job"); return;}
        System.out.println("# = " + this.jobNumber + " , arrive = " + this.arrivalTime + " , burst = " +
this.burst);
    }
}
```