Shortest Remaining Time :-

Process
wait_queue
completed.

(1) main.

{
create 30 process and add it into processes arraylist.
}

sort the processes list by arrival time.

(2) create new obj. of class :

srt.

Printing the output :

for (P_H process : srt. getProcess())
↳ gives the completed process.

table += .... getting all the values which needs to be printed in output file.

1. getName() : as process is a object of class process_Handler we can access all of its methods
   - this method will return the name of the process.

2. getArrivalTime : we are using setter and getter methods to set & retrieve the values of variable These methods are declared in Process_Handler class. arrival time ranges from 0 to 99.

3. getGivenExecutionTime : this will return total execution time req. for any particular process to complete the execution. execution time is beth 0 to 10.

4. calculateTurnaroundTime : it gives the total time from process arrival to completion. that is (completion_time) - (arrival_time)

5. calculateWaitTime : wait time is diff beth turnaround time and completion time. (or end)

6. CalculateResponseTime : it is diff beth (execution_start_time) - (arrival_time)

7. arrangeListBy_ArrivalTime (processes) :

It uses comparator interface. so it is mandatory to implement that interface method. so that's why we are using

Collections.sort (list, comparator)

- this with "sort" method will internally call the "compare" method of the class it is sorting.
- compare method returns -1, 0 or 1 to say if it is less than, equal or greater.
- this results used by the sort method to determine they should be swaped or not. (processes)

method.

⇒ so steps of arrange_ListBy_ArrivalTime :

1- create new object of class comparator

2- implement the compare method.

↳ it takes two processes to compare

↳ if process 1 arrives first
then return -1

↳ if process 2 arrives first
then return 1

else
0.

3. After that call collections. sort (List, Comparator which will sort all the processes based on their arrival time.

8. ⇒ arrange_ListBy_ExecutionTime :

- This method is same as arrange_ListBy_A.T.
- the only diff is it compares processes based on their execution time.

9. getCompletionTime : gives the time when process is completed.

10. get ExecutionStartTime : returns the time when execution is started for any particular process.

(3) Public ~~static~~ SRT (A.L. processes) :

- assign process
- create new wait_queue A.L.
- for now shortest is null (initially)
- create new A.L. for completed process.
- call run method.

(4) Private void run()

- set all values initially 0.
  (quanta, T.T, wait time, R.T, process-finished)
- add the processes to the wait_queue according to current time quanta.
- if process arival time is < current quanta then it will be added to queue.
- if wait_queue is not empty then choose the process with shortest execution time from the process arraylist and assign it to "shortest" variable.
- as shortest will be executed next remove it from wait_queue (wait_queue.remove(shortest))
- now start the execution of current shortest time process.
- if process is already running then, change following:
  decrement its execution time and quanta remaining value. ++.

- if shortest process is completed then mark its completion time & current Quanta value. and add this process into completed process arraylist. and also increment the "processes_finished" counter.

also update values for total_turnaroundtime, total_waitime, total_responsetime

- if process is not completed then add it to wait_queue again.

(5) Public string getAverages ( ):

calculate Avg. T.T, Avg. wait time, Avg. response time, and throughput and return it in a form of string.

this string (output) will be used in "Tester.java" class while you call your class (e.g. "SRT" here) and that will be written into the output txt file.