Bil 212
Fall 2022
Assignment2
December 4, 2022 23:59 max 110points
December 15, 2022 23:59 max 80points

Task-Resource Scheduling Algorithm (Job-Resource Scheduler)

In this assignment you are expected to create a different version of the previous scheduler class, JobScheduler. The purpose of this class is to store the input processes in a min-heap data structure according to their arrival time (arrivalTime), direct them to an appropriate resource (processor core or thread) and schedule their execution time.

Inputs: Jobs.txt and Dependencies.txt files with columns separated by spaces

Jobs.txt // space separated txt file
JobID duration

Each line of this file either contains a job information or a "no job" string. The ids do not have to be sequential.

Example:
1 3
2 2
no job
3 4
4 1
no job
5 2
6 3
7 2

Dependencies.txt // space separated txt file
JobID1 JobID2 //JobID1 cannot start until JobID2 is finished.

This file contains the dependencies between Jobs. Each Job can have 0,1 or more than one dependency. That Job cannot start until all its dependencies are finished.

Example:
4 3
5 3
7 4
7 5

Operation

In the first assignment, the time simulation and schedule filling phases did not intersect with each other. In other words, first all the jobs that will come to the schedule were taken and then the time was started with the run() method and no jobs could be added to the schedule after that. In this assignment, the execution of the schedule and the execution of your assignment code will be synchronized. A new job may come during runtime, that job should be placed in the min-heap appropriately.

In this system, there are two different blocking reasons for a job. The first one is due to the dependency of the job (*dependencyBlocked)*, the job it depends on cannot run because it is not finished yet. The second one is due to resources being busy *(resourceBlocked*), there is nothing binding it but it is waiting because all resources are busy.

The first thing to do in the driver code you are given is to create a hashmap for the dependencies using the Dependency.txt file.

There will then be a loop. The condition checked in this loop checks if there is a new line in the file containing the jobs. If the loop is entered, the system proceeds as follows:

1. When a new line is found, the *insertJob( )* method increments the **timer** variable by one. This new value is the arrival time of the incoming row. If the incoming row is a new job, this job is stored in the min-heap according to the arrival time. If the string "no job" is received, no action is taken. Thus the min-heap is reorganized for the next time slot
2. When the *run()* method runs, it saves the jobs that have run out of time in a suitable data structure. If no resources are available, it doesn't need to do anything (if you want to do something extra for resourceblocked jobs, you can). But if there is a resource to run a job, it takes a job from min-heap and checks the dependencies of this job. If the processes it depends on are not finished yet, this job is added to a temporary data structure. This process continues until a job with no dependency problems arrives. When such a job arrives, the necessary resource assignment is made and the jobs in the temporary data structure are sent back to the min-heap.

Failure to enter the loop means that no new job will come. In this case, the *runAllRemaining()* method runs to finish the jobs that are running in the system or waiting for their turn (Note: setting the timer variable and assigning jobs are the two most important components of this method). The method exits when there are no jobs left in the system.

Data structures to be used:

min heap: Incoming jobs are kept here according to their arrival time.
timer: the time flow will be realized through this variable. The timer variable is set with insertJob() when in the loop and runAllRemaining() when out of the loop.
filePath: IO operations will be performed inside the class. When creating an object of the class, this value is taken as a parameter and used in the required methods.
map: the data structure to hold the dependencies (use the

nomenclature from JobScheduler.java)

This one data structures use **mandatory**. These are except for necessity you can use the data structures from the sections seen in the tutorial.

Scheduling interface:

public boolean stillContinues( ) Checks if the Jobs.txt file has a continuation.

public void insertDependencies( ) reads the dependency file and saves the dependencies.

public void setResourcesCount(Integer) Classroom to use required sets the total number of resources (cores).

public void insertJob( ) Reads a new line from the Job file. This line may contain a job and its duration, or it may contain "no job" information that no job will be added to the schedule. It will insert or not according to this information.
It also increments the timer variable by one.

public void run( ) Performs resource assignment for jobs that have runtime and have no dependencies. (See the second item in the Functionality section)

public void completedJobs( ) Prints the operations that have been executed and finished so far.

public void dependencyBlockedJobs( ) Prints the pending jobs on the screen because the dependent job/jobs are not finished yet.

public void resourceBlockedJobs( ) Prints the waiting jobs on the screen because there is no free resource.

public void workingJobs( ) Prints the running job/jobs on the screen. For example, if it is called when the timer variable is 5, it prints only the jobs that actively use resources at 5.

public void allTimeLine( ) Prints on the screen which job was executed in which time unit in which resource until the time unit it is called.

public void runAllRemaining( ) If there is no new job, the loop cannot be entered. But there may be jobs that still need to run in the system. It finishes all of these operations (Here you can set the timer variable and call the run() method).

You should also print the additional information requested in the comments in the driver script. You can also see it in the output.

Acceptances

- The dependent job does not have to arrive before the job itself.
- *setResorcesCount() and insertDependencies()* driver in the code main will be executed before the loop starts.
- *runAllRemaining( )* will be run after the main loop in the driver code.
- Resource assignment priority is set according to the resource with the lower number. For example, when both resources 1 and 2 are idle and a job comes in, resource 1 is assigned.
- Except min-heap, you can use other classes from libraries (such as Hashmap, Set)
- The driver program will not be given a dependency that will lead to an infinite loop.
- One job both resourceblocked both dependencyblocked if, o job you should include dependencyblocked (like the example program and job 7 in the output).

Sample Program and Output:

Below is the sample driver code of the designed class. **The console output should be as given below. Do not use any other words.**

Driver code:

```
public static void main(String[] args) {
   String path1 = "pathOfJobFile";
  String path2 = "pathOfDependencyFile";
  JobScheduler cizelge = new JobScheduler(path1);
  cizelge. setResourcesCount(2); //same as one in hw1
  cizelge. insertDependencies(path2);
 while(line.stillContinues()){ // The stillContinues method checks if there is
anything in the job file


  cizelge. insertJob(); //insertJob reads a new line from the inputfile, adds a job if
necessary
  System.out.println("min-heap\n" +schedule); // a proper toString method for JobSchedular
class to print content of heap as tree
// printing as a tree
 cizelge.run(); //different from one in hw1


 cizelge. completedJobs(); // prints completed jobs
 cizelge. dependencyBlockedJobs(); // prints jobs whose time is up but waits due to its
dependency, also prints its dependency
```

```
   cizelge. resourceBlockedJobs(); // prints jobs whose time is up but waits due to busy
resources
   cizelge. workingJobs(); // prints jobs working on this cycle and its
   resource System.out.println("------------- "+cizelge.timer+"       ");
   }

   cizelge.runAllRemaining();
  cizelge.allTimeLine();
  }
```

Sample output: (output when run with the initial sample file)

```
min-heap
1
completed jobs
dependency blocked jobs
resource blocked jobs
working jobs (1,1)
------------- 1 -------------
min-heap
2
completed jobs
dependency blocked jobs
resource blocked jobs
working jobs (1,1) (2,2)
------------- 2 -------------
min-heap
completed jobs
dependency blocked jobs
resource blocked jobs
working jobs (1,1) (2,2)
------------- 3 -------------
min-heap
3
completed jobs 1, 2
dependency blocked jobs
resource blocked jobs
working jobs (3,1)
------------- 4 -------------
min-heap
4
completed jobs 1, 2
dependency blocked jobs (4,3)
resource blocked jobs
working jobs (3,1)
------------- 5 -------------
min-heap
```

4
completed jobs
dependency blocked jobs (4,3)
resource blocked jobs
working jobs (3,1)
---------- *6* ----------
min-heap
4
5
completed jobs 1,2
dependency blocked jobs (4,3) (5,3)
resource blocked jobs
working jobs (3,1)
---------- *7* ----------
min-heap
6
completed jobs 1, 2, 3
dependency blocked jobs
resource blocked jobs 6
working jobs (4,1) (5,2)
---------- *8* ----------
min-heap
7
completed jobs 1, 2, 3, 4
dependency blocked jobs (7,5)
resource blocked jobs
working jobs (6,1) (5,2)
---------- *9* ----------
alltimeline

	R1	R2
1	1	
2	1	2
3	1	2
4	3	
5	3	
6	3	
7	3	
8	4	5
9	6	5
10	6	7
11		7

My submission:

- Add all the classes you wrote as inner classes in *JobScheduler.java.* Only *JobScheduler.java* will be loaded on the remote system.
- Add your name, surname and number as a comment at the top of your code.

- Compile with Java 11.
- Make it easier to understand by adding comments inside your code.
- Uncompiled or above driver code erroneous employee your codes will not be evaluated.
- Points will be deducted from codes that do not comply with the restrictions specified here.
- Properly written codes will receive bonus points.