```java
import java.util.LinkedList;

public class IOScheduler
{
    /**
     * VARIABLES***************************************************************
     */
    LinkedList<Integer> defaultQueue;    // I/O for non-blocked jobs
    LinkedList<Integer> blockedInQueue; // I/O for blocked, in-memory jobs
    LinkedList<Integer> blockedOutQueue;// I/O for blocked, not in-memory jobs
    LinkedList<Integer> terminatedQueue;// I/O for terminated jobs
    LinkedList<Integer> reportedToSwap;
    int inIO;                          // current process in I/O

    /**
     * CONSTRUCTOR*************************************************************
     */
    IOScheduler ()
    {
        defaultQueue = new LinkedList<Integer>();
        blockedInQueue = new LinkedList<Integer>();
        blockedOutQueue = new LinkedList<Integer>();
        terminatedQueue = new LinkedList<Integer>();
        reportedToSwap = new LinkedList<Integer>();
        inIO = -1;
    }

    /**
     * PRIVATE METHODS*********************************************************
     */
    /**
     * Gets next available I/O task
     */
    void ioCheck ()
    {
        // System.out.println("-I/O Queues:");
        // System.out.println("--BlockedIn Queue has " + blockedInQueue.size());
        // System.out.println("--Terminated Queue has " + terminatedQueue.size());
        // System.out.println("--Default Queue has " + defaultQueue.size());
        if (inIO == -1)
        {
            // Checks terminated first
            if (!terminatedQueue.isEmpty()) {
                inIO = terminatedQueue.remove();
            }
            // Then in-memory-blocked
            else if (!blockedInQueue.isEmpty()) {
                inIO = blockedInQueue.remove();
            }
            // Then regular jobs
            else if (!defaultQueue.isEmpty()) {
                inIO = defaultQueue.remove();
            }
            // If an I/O task was found, run it
            if (inIO != -1) {
                // System.out.println("--Job is sent to do I/O");
                JobTable.setDoingIO(inIO);
            sos.siodisk(inIO);
        }
    }
}

/**
 * PUBLIC METHODS**********************************************************
 */

/**
 * Accepts new job for I/O
 * Adds new job to default I/O queue
 * @param jobID of the job to add
 */
public void add (int jobID)
{
    // System.out.println("-IOScheduler accepts new job");
    JobTable.incrementIO(jobID);
    defaultQueue.add(jobID);
    ioCheck();
}

/**
 * Checks to see if given job is doing I/O
 * @param  jobID job to query
 * @return       boolean true if job in I/O, false otherwise
 */
public boolean doingIO (int jobID)
{
    if (jobID == inIO) {
        // System.out.println("--Job " + jobID + " is doing I/O");
        return true;
    }
    else {
        // System.out.println("--Job " + jobID + " is not doing I/O");
        return false;
    }
}

/**
 * When I/O has finished, removes current job from inIO
 * Tries to start next job if available
 * Updates inIO
 * @return jobID of process that just finished I/O and jobID of
 *               process that needs to be brought into memory
 */
public int ioDone ()
{
    // If job which finished I/O is valid
    int jobID = inIO;
    inIO = -1;
    if (jobID != -1) {
        // Decrement & get it's I/O pending
        JobTable.decrementIO(jobID);
        JobTable.unsetDoingIO(jobID);
        // If the job is ready to be unblocked
        if (JobTable.getIO(jobID) == 0 && JobTable.isBlocked(jobID)) {
                JobTable.unsetBlocked(jobID);
            }
        }

        ioCheck();
        return jobID;
    }

    /**
     * Prints status of I/O
     */
    public void print () {
        // System.out.println("-I/O Report:");
        // System.out.println("--In I/O: " + inIO);
        // System.out.println("--Next In Queue: ");
        // System.out.print("---Terminated: ");
        // for (int i = 0; i < terminatedQueue.size(); i++)
        // {
        //  System.out.print(terminatedQueue.get(i) + ", ");
        // }
        // System.out.print("---blockedInQueue: ");
        // for (int i = 0; i < blockedInQueue.size(); i++)
        // {
        //  System.out.print(blockedInQueue.get(i) + ", ");
        // }
        // System.out.print("---blockedOutQueue: ");
        // for (int i = 0; i < blockedOutQueue.size(); i++)
        // {
        //  System.out.print(blockedOutQueue.get(i) + ", ");
        // }
        // System.out.print("---DefaultQueue: ");
        // for (int i = 0; i < defaultQueue.size(); i++)
        // {
        //  System.out.print(defaultQueue.get(i) + ", ");
        // }
    }

    /**
     * Checks blocked in-memory for jobs to swap out
     * @return jobID of job to swap out
     */
    public int readyToLeave ()
    {
        // Job that is ready to leave, initialized to invalid
        int swapOutBlockedJob = -1;

        // If blocked jobs are in memory
        if (!blockedInQueue.isEmpty()) {
            // Checks if last element in queue is already swapping,
            // if not, then set to swap in
            if (blockedInQueue.getLast() != inIO) {
                swapOutBlockedJob = blockedInQueue.getLast();
            }
            // Checks if a blocked job has exceeded its in memory time,
            // if it has, then set to swap out
            else if (JobTable.getPriorityTime(blockedInQueue.getFirst())
                > 1000) {
```

```
                swapOutBlockedJob = blockedInQueue.getFirst();
            }
            // Checks to if job selected hasn't already requested to
            // be swapped. If it has, then invalidate job to swap out.
            // If it hasn't then add to reported to swap list
            if (reportedToSwap.contains(swapOutBlockedJob)) {
                swapOutBlockedJob = -1;
            }
            else {
                reportedToSwap.add(swapOutBlockedJob);
            }
        }
        return swapOutBlockedJob;
    }

    /**
     * Checks blocked not in-memory for jobs to swap in if there is no
     * blocked job in memory
     * @return jobID of job to swap in
     */
    public int readyToReturn ()
    {
        int swapInBlockedJob = -1;
        if (blockedInQueue.isEmpty() && !blockedOutQueue.isEmpty()) {
            swapInBlockedJob = blockedOutQueue.getFirst();
            if (reportedToSwap.contains(swapInBlockedJob)) {
                swapInBlockedJob = -1;
            }
            else {
                reportedToSwap.add(swapInBlockedJob);
            }
        }
        return swapInBlockedJob;
    }



    /**
     * Moves a jobs I/O to correct queue when status changes
     * @param jobID the job to move
     */
    public void moveIO (int jobID)
    {
        if (jobID != -1) {
            int ioCount = JobTable.getIO(jobID);
            if (ioCount > 0) {
                if (reportedToSwap.contains(jobID)) {
                    reportedToSwap.remove((Integer)jobID);
                }
                // System.out.println("-IO");
                // Query the jobTable to determine which list I/O should move to
                // Move to terminated
                if (JobTable.isTerminated(jobID)) {
                    // Remove from default, blockedIn
                    // System.out.println("--Moving to terminatedQueue");
                    while (defaultQueue.contains((Integer)jobID)) {
                        defaultQueue.remove((Integer)jobID);
```

```java
                        terminatedQueue.add(jobID);
                    }
                    while (blockedInQueue.contains((Integer)jobID)) {
                        blockedInQueue.remove((Integer)jobID);
                        terminatedQueue.add(jobID);
                    }
                }
                // Move to blockedOut
                else if (JobTable.isBlocked(jobID)) {
                    // Move to blockedOut
                    if (JobTable.getAddress(jobID) == -1 ||
                        (JobTable.getAddress(jobID) != -1 &&
                            JobTable.isSwapping(jobID))) {
                        // System.out.println("--Moving to blockedOutQueue");
                        // Remove from blockedIn
                        while (blockedInQueue.contains((Integer)jobID)) {
                            blockedInQueue.remove((Integer)jobID);
                            blockedOutQueue.add(jobID);
                        }
                        while (defaultQueue.contains((Integer)jobID)) {
                            defaultQueue.remove((Integer)jobID);
                            blockedOutQueue.add(jobID);
                        }
                    }
                    // Move to blockedIn
                    else if (JobTable.getAddress(jobID) != -1 ||
                        (JobTable.getAddress(jobID) == -1
                            && JobTable.isSwapping(jobID))) {
                        // System.out.println("--Moving to blockedInQueue");
                        // Remove from default
                        while (defaultQueue.contains((Integer)jobID)) {
                            defaultQueue.remove((Integer)jobID);
                            blockedInQueue.add(jobID);
                        }
                        while (blockedOutQueue.contains((Integer)jobID)) {
                            blockedOutQueue.remove((Integer)jobID);
                            blockedInQueue.add(jobID);
                        }
                    }
                }
            }
        }
    }
}
```