public class GlobalVariables

{

//System Variable

public static long time = System.currentTimeMillis();

}

import javax.swing.JOptionPane;

public class Main

{

public static void main(String[] args)

{

/\*\*

\* The number of students is initialized at 20, if you want to have a different

\* number of students then in the command line argument you must type "-s"

\* followed by the number of students desired.  The minimum is 1 student.

\* If anything besides this command line argument is entered after 2 failed

\* attempts (the second being a JOptionPane) the 20 default will kick in.

\*/

int numberOfStudents = 20;

if (args.length > 0 && args[0].equals("-s"))

{

try

{

numberOfStudents = Integer.parseInt(args[1]);

while (numberOfStudents<=0 && numberOfStudents > 50)

{

numberOfStudents = Integer.parseInt(JOptionPane.showInputDialog(null, "There must be at least 1 Student and no more than 50."));

}

}

catch(NumberFormatException e)

{

numberOfStudents = 20;

}

}

/\*\*

\* The Teacher thread is created and so are the student threads based

\* on the number entered in the command line argument. The threads are started

\* automatically in the constructor of each class.

\*/

new Teacher();

Students[] TotalStudents  = new Students[numberOfStudents];

for(int i = 0; i < numberOfStudents; i++)

{

TotalStudents[i] = new Students();

}

}

}

import java.util.Random;

import java.util.concurrent.Semaphore;

public class Students extends Thread {

    //Variables Pertaining to Each Student

    static int id = 1;

    int MyGroup = 0;

    int TotalClassesTaken = 0;

    long StartClassTime[] = {0, 0, 0, 0};

    long EndClassTime[] = {0, 0, 0, 0};

    String ClassesAttended[] = {"Pending", "Pending", "Pending", "Pending"};

    //Variables Pertaining to the Bathroom

    public static Semaphore BathroomMutex = new Semaphore(1, true);

    //Variables Pertaining to the Classes

    static String Classes[] = {"Bioinformatics", "Math 241", "Quantum Computing", "Operating Systems"};

    public static Semaphore ClassMutex = new Semaphore(1, true),

            StartClass = new Semaphore(0, true),

            InClass = new Semaphore(0, true);

    public static int waitingForClass[] = {0, 0, 0, 0};

    //Variables Pertaining to the Group

    public static Semaphore GroupFormed = new Semaphore(0, true);

    static int numStudents = 0;

    static int groupsize = 0;

    static int groupNum = 0;

    //Variables Pertaining to the Tables

    public static Semaphore TableMutex = new Semaphore(1, true);

    static int numTables = 5;

    public static Semaphore TablesAvailable = new Semaphore(numTables, true);

    static int TableCapacity = 3;

    public static Semaphore TableGroup = new Semaphore(0, true);

    //Variables Pertaining to the Eating

    static int peopleeating = 0;

    public static Semaphore ReadyToEatMutex = new Semaphore(1, true);

    public static Semaphore WaitingToEat = new Semaphore(0, true);

    //Variables Pertaining to the Dorms

    public static Semaphore DormsMutex = new Semaphore(1, true);

    static int studentsInDorms = 0;

    //Variables Pertaining to the Student Schedule

    public static Semaphore StudentSchMutex = new Semaphore(0, true);

    /\*\*

     \* Constructor for EACH Student

     \*/

    public Students() {

        //Sets the name of EACH student and assigns them a unique id (never duplicated).

        //Starts the thread.

        numStudents = id;

        setName("Student-" + id++);

        start();

    }

    /\*\*

     \* A simple function that displays the current time in milliseconds

     \* @return

     \*/

    public long currentTime() {

        return System.currentTimeMillis() - GlobalVariables.time;

    }

    /\*\*

     \* Messages that occur throughout the day come from this function.

     \* @param m

     \*/

    public void msg(String m) {

        System.out.println("[" + (currentTime()) + "] " + getName() + ":" + m);

    }

    /\*\*

     \* This function is used to generate a random number that has a max

     \* no greater than 5000 milliseconds.

     \* @return

     \*/

    public static int RandomNumber() {

        Random rand = new Random();

        int timeToSleep = rand.nextInt(5000);

        return timeToSleep;

    }

    /\*\*

     \* The HaveFun function takes the Thread that calls on it and sets the Priority

     \* higher than the default (5).  Then sleeps for random time to simulate

     \* having fun.  After it wakes up it sets the priority of the Thread

     \* back to the default.

     \*/

    public void HaveFun() {

        try {

            setPriority(10);

            sleep(RandomNumber());

        } catch (InterruptedException e) {

            setPriority(5);

        }

        return;

    }

    /\*\*

     \* The Errands function takes the Thread that calls on it and puts it to

     \* sleeps for random time to simulate running errands.

     \*/

    public void Errands() {

        try {

            sleep(RandomNumber());

        } catch (InterruptedException e) {

        }

        return;

    }

    /\*\*

     \* Runs a Thread that has been started

     \*/

    public void run() {

        //Rise and Shine

        //Sleeps for random time and wakes up this simulates

        //waking up in the morning.

        try {

            sleep(RandomNumber());

        } catch (InterruptedException e) {

        }

        //Thread is awake

        //Each thread uses the bathroom in the order in which they woke up

        //Sleep is used to simulate the time inside the bathroom

        try {

            BathroomMutex.acquire();

            msg(" is in the bathroom");

            sleep(RandomNumber());

        } catch (InterruptedException e1) {

        }

        msg(" is out of the bathroom.");

        BathroomMutex.release();

        //Finished using the Bathroom

        /\*\*

         \* This is a for loop that runs through every class for each student

         \*/

        for (int i = 0; i < Classes.length; i++) {

            /\*\*

             \* Simple enough if the Teacher has not signaled that class has

             \* started then they go into the auditorium and wait for the teacher

             \* to arrive. Once the Teacher arrives they stay in class till class

             \* is over and then go have fun before the next class begins.

             \*/

            if (Teacher.ClassStarted[i] == "No") {

                try {

                    ClassMutex.acquire();

                    ClassesAttended[i] = "Attended";

                    StartClassTime[i] = currentTime();

                    waitingForClass[i]++;

                    TotalClassesTaken++;

                    ClassMutex.release();

                    //msg(" is waiting for " + Classes[i]);

                    StartClass.acquire();

                    msg(" is in " + Classes[i]);

                    InClass.acquire();

                    msg(" is out of " + Classes[i] + " and having fun.");

                    EndClassTime[i] = currentTime();

                    HaveFun();

                } catch (InterruptedException e) {

                }

            } else {

                /\*\*

                 \* If the class is over they might as well wait for the next

                 \* class to begin

                 \*/

                if (Teacher.ClassStarted[i] == "Over") {

                    ClassesAttended[i] = "Did Not Attend";

                }

                /\*\*

                 \* If the class is in session and they missed class they

                 \* try to run errands before the next class starts.

                 \*/

                else {

                    ClassesAttended[i] = "Did Not Attend";

                    msg(" missed " + Classes[i] + " and doing errands");

                    Errands();

                }

            }

        }

        /\*\*

         \* Cafeteria Starts Here! After all classes are over

         \* each student steps into the cafeteria waiting for two other people

         \* to pair up with. Once in a group of three they let the teacher know

         \* and the teacher places them into a table.  If no table is available

         \* they will have to wait until a whole group leaves 1 table.  When the

         \* group leaves the table the Teacher is notified that there is a table

         \* available.  The teacher then releases the next group onto that table.

         \*/

        try {

            TableMutex.acquire();

            groupsize++;

            if (groupsize % TableCapacity != 0 && groupsize != numStudents) {

                msg(" is waiting for fellow students");

                TableMutex.release();

                TableGroup.acquire();

                MyGroup = groupNum;

                WaitingToEat.acquire();

            } else {

                groupNum++;

                msg(" formed Group " + groupNum);

                MyGroup = groupNum;

                TableMutex.release();

                for (int i = 1; i < 3; i++) {

                    TableGroup.release();

                }

                GroupFormed.release();

                WaitingToEat.acquire();

            }

            ReadyToEatMutex.acquire();

            peopleeating++;

            sleep(RandomNumber());

            if (peopleeating % TableCapacity == 0 || peopleeating == numStudents) {

                TablesAvailable.release();

            }

            msg(" finished eating and heads to the dorms.");

            ReadyToEatMutex.release();

            /\*\*

             \* Once a student finishes eating he/she heads back to the dorms.

             \* The last person that heads to the dorms lets the teacher know that

             \* everyone has finished eating and the Teacher goes home.

             \*/

            DormsMutex.acquire();

            studentsInDorms++;

            msg(" is in the dorms");

            if (studentsInDorms == numStudents) {

                GroupFormed.release();

                Teacher.TeacherWaitsForStudent.release();

                StudentSchMutex.release();

            }

            DormsMutex.release();

            /\*\*

             \* Once everyone has arrived at the dorms the Daily Report Of Classes

             \* for each student is displayed.

             \*/

            StudentSchMutex.acquire();

            System.out.println();

            System.out.println("Daily Report Of Classes");

            System.out.println("Name: " + getName());

            if (ClassesAttended[0] == "Attended") {

                System.out.println(Classes[0] + ": " + ClassesAttended[0] + " from " + StartClassTime[0] + " to " + EndClassTime[0]);

            } else {

                System.out.println(Classes[0] + ": " + ClassesAttended[0]);

            }

            if (ClassesAttended[1] == "Attended") {

                System.out.println(Classes[1] + ": " + ClassesAttended[1] + " from " + StartClassTime[1] + " to " + EndClassTime[1]);

            } else {

                System.out.println(Classes[1] + ": " + ClassesAttended[1]);

            }

            if (ClassesAttended[2] == "Attended") {

                System.out.println(Classes[2] + ": " + ClassesAttended[2] + " from " + StartClassTime[2] + " to " + EndClassTime[2]);

            } else {

                System.out.println(Classes[2] + ": " + ClassesAttended[2]);

            }

            if (ClassesAttended[3] == "Attended") {

                System.out.println(Classes[3] + ": " + ClassesAttended[3] + " from " + StartClassTime[3] + " to " + EndClassTime[3]);

            } else {

                System.out.println(Classes[3] + ": " + ClassesAttended[3]);

            }

            System.out.println("Total Classes Attended: " + TotalClassesTaken);

            System.out.println(getName() + " was a member of Group " + MyGroup);

            StudentSchMutex.release();

        } catch (InterruptedException e) {

        }

    }// ends run

}//ends Students class

import java.util.concurrent.Semaphore;

public class Teacher extends Thread

{

//Teacher's Class Schedule Variables

static int ClassSched [] = {15000, 30000, 50000, 65000};

static String ClassStarted[] = {"No", "No", "No", "No"};

public static int classlength = 10000;

int ClassTurn = 0;

//Teacher Has To Wait For The Last Student to Go To The Dorms

public static Semaphore TeacherWaitsForStudent = new Semaphore(0, true);

/\*\*

\* Constructor for EACH Student

\*/

Teacher()

{

setName("Teacher");

start();

}

/\*\*

\* A simple function that displays the current time in milli seconds

\* @return

\*/

public long currentTime()

{

return System.currentTimeMillis()-GlobalVariables.time;

}

/\*\*

\* Messages that occur throughout the day come from this function.

\* @param m

\*/

public void msg(String m)

{

System.out.println("["+(currentTime())+"] "+getName()+":"+m);

}

/\*\*

\* Runs a Thread that has been started

\*/

public void run()

{

while(ClassStarted[Students.Classes.length - 1] != "Over")

{

//Office Hours kicks in after the second class

if(ClassTurn == 2)

{

msg(" started Office Hours.");

try

{

sleep(10000);

}

catch (InterruptedException e)

{}

msg(" ended Office Hours.");

}

/\*\*

\* Runs through each class by releasing all the students that

\* were in the auditorium waiting for the class to start.

\* Then moves to the next class till classes are all over.

\*/

if (currentTime() >= ClassSched[ClassTurn])

{

ClassStarted[ClassTurn] = "Yes";

msg(" started " + Students.Classes[ClassTurn] + ".");

for(int i = 0; i < Students.waitingForClass[ClassTurn]; i++)

{

Students.StartClass.release();

}

try

{

sleep(classlength);

ClassStarted[ClassTurn] = "Over";

msg(" ended " + Students.Classes[ClassTurn]+ ".");

for(int i = 0; i < Students.waitingForClass[ClassTurn]; i++)

{

Students.InClass.release();

}

}

catch (InterruptedException e)

{}

ClassTurn++;

}

}//ends for

/\*\*

\* The first thing the teacher does is wait for a group of students

\* to be formed.  Once the group is formed it checks for an available

\* table.  If no table is available it will wait until a table is free

\* aka after 1 group finishes eating at 1 table. When there is no one

\* left waiting to be seated to eat the teacher waits till the last

\* student finishes eating and leaves the cafeteria. This student

\* notifies the teacher that alll students are done and the teacher leaves.

\*/

try

{

Students.GroupFormed.acquire();

Students.TablesAvailable.acquire();

while(Students.WaitingToEat.hasQueuedThreads())

{

for (int i = 0; i < Students.TableCapacity; i++)

{

Students.WaitingToEat.release();

}

if(Students.WaitingToEat.hasQueuedThreads())

{

Students.GroupFormed.acquire();

Students.TablesAvailable.acquire();

}

else

{

TeacherWaitsForStudent.acquire();

msg(" Leaves The Cafeteria");

}

}

}

catch (InterruptedException e)

{}

}

}