public class GlobalVariables

{

//These Variables Are All Globally Used By Both Threads

//Class Based Variables

static boolean ClassListOccupied = false;

static boolean ClassInSession[] = {false, false, false, false, false};

static String ClassStatus[] ={"Didn't Start","Didn't Start","Didn't Start","Didn't Start","Didn't Start"};

//Join Based Variables

static int amountOfPeopleFinished = 0;

//System Variables

public static long time = System.currentTimeMillis();

//Wait & Notify functions used for when the Teacher Arrives

public static GlobalVariables TeacherArrives = new GlobalVariables();

public synchronized void waitForTeacher()

{

try

{

wait();

}

catch (InterruptedException e)

{

}

}

public synchronized void teacherArrivesAtClass()

{

notifyAll();

}

}

import javax.swing.JOptionPane;

public class Main

{

public static void main(String[] args)

{

/\*\*

\* The number of students is initialized at 13, 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 13 default will kick in.

\*/

int numberOfStudents = 13;

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 = 13;

}

}

/\*\*

\* 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;

public class Students extends Thread {

    //Variables Pertaining to the Student

    static int id = 1;

    int StudentID;

    String StudentStatus = "Running";

    boolean wokenUp = false;

    boolean numberadded = false;

    int TotalClassesTaken = 0;

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

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

    //Variables Pertaining to the Join

    boolean sortedForJoin = false;

    static boolean everyOneReady = false;

    //Variables Pertaining to the Classes

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

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

    boolean wentToClass[] = {false, false, false, false, false};

    boolean ClassMsg[] = {false, false, false, false, false};

    //Variables Pertaining to the Bathroom

    public static int[] BathroomTurn = new int[1];

    static int nextInLine = 0;

    static int herewego = 0;

    boolean wentToBathroom = false;

    /\*\*

     \* Constructor for EACH Student

     \*/

    public Students() {

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

        setName("Student-" + id++);

/\*

\* Creates an array for each student but keeps expanding it as more students are

\* created. The last student created makes the final size of the this array.

\* For now this is all to declare the appropriate size of the array to be used

\* later on.

\*/

        int[] BathroomTemp = BathroomTurn;

        BathroomTurn = new int[BathroomTemp.length + 1];

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

            BathroomTurn[i] = BathroomTemp[i];

        }

        BathroomTurn[id - 1] = 0;

        StudentID = id - 1;

        //Finally after the student is established the Thread is started.

        start();

    }

    /\*\*

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

     \* The time is grabbed from the class "GlobalVariables" so that every

     \* student is on the same time schedule and not have a time of its own.

     \*

     \* @param m

     \*/

    public void msg(String m) {

        System.out.println("[" + (System.currentTimeMillis() - GlobalVariables.time) + "] " + 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

     \* getting wasted.  After it wakes up it sets the priority of the Thread

     \* back to the default.

     \*/

    public void HaveFun() {

        setPriority(10);

        try {

            sleep(RandomNumber());

        } catch (InterruptedException e) {

            e.printStackTrace();

        }

        setPriority(5);

        return;

    }

    /\*\*

     \* Runs a Thread that has been started

     \*/

    public void run() {

        while (finishedClassList != true) {

/\*

\* Wakes each Thread up at a different random time

\* and makes sure since the thread is awake from sleeping

\* it will not "wake up" again.

\*/

            if (wokenUp == false) {

                try {

                    sleep(RandomNumber());

                } catch (InterruptedException e) {

                    e.printStackTrace();

                }

                wokenUp = true;

            }

/\*

\* After waking up from a long night's rest the StudentID

\* is put into an array to use the bathroom and then

\* makes sure that it doesn't get added to the array

\* again.

\*/

            if (numberadded == false) {

                BathroomTurn[nextInLine] = StudentID;

                nextInLine++;

                numberadded = true;

            }

/\*

\* This is where the students wait their turn in line for the bathroom.

\* When it is their turn, they break out of the while loop and

\* head into the bathroom if they didn't already go. (Busy Wait)

\*/

            while (BathroomTurn[herewego] != StudentID && wentToBathroom == false) {

                yield();

            }

/\*

\* If it is the student's turn to use the bathroom this is where they

\* use it. To simulate the use of the bathroom they are put to sleep

\* for a random time. After their turn, they signal the next person

\* whose in line to use the bathroom and that the bathroom is available.

\*/

            if (wentToBathroom == false) {

                msg(" is using the bathroom.");

                try {

                    sleep(RandomNumber());

                } catch (InterruptedException e) {

                    e.printStackTrace();

                }

                wentToBathroom = true;

                msg(" finished using the bathroom.");

                herewego++;

            }

/\*

\* Each student will go through the day of school here.

\* As long as their status is running they go through the school day.

\* Once they finish their last class they get out of of school and set

\* their status to "Done".

\*/

            if (StudentStatus == "Running") {

                for (int i = 0; i < Classes.length; i++)//Runs through each class

                {

                    while (GlobalVariables.ClassStatus[i] == "Didn't Start" && ClassesAttended[i] == "Pending" && wentToClass[i] == false) {

/\*

\* As long as the class hasn't started, the student's

\* report is still pending for the class and the student

\* didn't go inside the classroom yet they busy wait here.

\* They set that they are on time for class by setting

\* wentToClass = true and then wait for the teacher's signal

\* to begin class.

\*/

                        wentToClass[i] = true;

                        GlobalVariables.TeacherArrives.waitForTeacher();

                    }//ends while

                    while ((GlobalVariables.ClassStatus[i] == "On" && wentToClass[i] == true) && ClassesAttended[i] == "Pending") {

/\*

\* As long as the class is on, the students were waiting for

\* class to start, and they still have a pending status for

\* this class they wait in here until the teacher sets the

\* class status to over.

\*/

                        if (ClassMsg[i] == false) {

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

                        }//ends if

                        if (StartClassTime[i] == 0) {

                            StartClassTime[i] = (System.currentTimeMillis() - GlobalVariables.time);

                        }

                        ClassMsg[i] = true;

                        ClassesAttended[i] = "In Class";

                    }//ends while

                    if (GlobalVariables.ClassStatus[i] == "Over" && ClassesAttended[i] == "In Class" && wentToClass[i] == true) {

/\*

\* If the student was in the class when the teacher signaled

\* for class to be over the student gets their report

\* status as attended and then has fun before the next class.

\* If that class was the last class of the day then the

\* status of the student is set to done. The classes taken also

\* increases by 1.

\*/

                        msg(" is out of " + Classes[i] + " and went to get some drinks.");

                        if (EndClassTime[i] == 0) {

                            EndClassTime[i] = (System.currentTimeMillis() - GlobalVariables.time);

                        }

                        ClassesAttended[i] = " Attended";

                        HaveFun();

                        TotalClassesTaken++;

                        if (Classes[i] == Classes[Classes.length - 1]) {

                            StudentStatus = "Done";

                        }//ends if

                    }//ends if

                    if (wentToClass[i] == false) {

/\*

\* If the student didn't go to the class they set their

\* class report to did not attend. They then go do

\* some errands for a random time and then attempt to go

\* to the next class if they succeed they will busy wait

\* for that class to start else they will end up back here

\* and run more errands till the next class. If they miss

\* the last class they run errands and then end their day.

\*/

                        wentToClass[i] = true;

                        ClassesAttended[i] = " Did Not Attend";

                        if (GlobalVariables.ClassStatus[i] != "Over") {

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

                        }

                        if (Classes[i] == Classes[Classes.length - 1]) {

                            try {

                                sleep(Teacher.classlength);

                            } catch (InterruptedException e) {

                                e.printStackTrace();

                            }

                        } else {

                            try {

                                sleep(RandomNumber());

                            } catch (InterruptedException e) {

                                e.printStackTrace();

                            }

                        }

                        if (Classes[i] == Classes[Classes.length - 1]) {

                            StudentStatus = "Done";

                        }

                    }//ends if

                }//ends for

            }//ends if

/\*

\* This will only print out after the students are all done with their

\* school day. When they are finished their school day they will print

\* a report of which classes they attended or didn't attend based on

\* the Student's individual ClassesAttended array and the overall

\* Classes array. When matched together it will show if the student

\* attended or didn't attend the class.

\*/

            while (GlobalVariables.ClassListOccupied == false && finishedClassList == false && StudentStatus == "Done") {

                GlobalVariables.ClassListOccupied = true;

                System.out.println("\n" + currentThread().getName() + "'s Daily Report" + "\n");

                for (int i = 0; i < 5; i++) {

                    for (int j = 0; j < 5; j++) {

                        if (i == j) {

                            if (ClassesAttended[j] == " Did Not Attend") {

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

                            } else {

                                System.out.println(Classes[i] + ":" + ClassesAttended[j] + " From: " + StartClassTime[i] + " To: " + EndClassTime[i]);

                            }

                        }

                    }

                }

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

                GlobalVariables.amountOfPeopleFinished++;

                finishedClassList = true;

                GlobalVariables.ClassListOccupied = false;

            }//ends while

/\*

\* This will just busy wait until everyone is ready AKA is in the dorm

\* and ready for the join function.

\*/

            while (finishedClassList == true && everyOneReady == false) {

                if (GlobalVariables.amountOfPeopleFinished == BathroomTurn.length - 1) {

                    everyOneReady = true;

                }

            }

/\*

\* This function joins the thread with the previous one and kills it.

\* The students join in descending order and die.

\*/

            while (finishedClassList == true && sortedForJoin == false && everyOneReady == true) {

                if (StudentID == id - 1) {

                    try {

                        interrupt();

                        join();

                    } catch (InterruptedException e) {

                        System.out.println(getName() + " is dead");

                        sortedForJoin = true;

                        id--;

                    }

                }//ends if

            }//ends while

        }//ends while

    }// ends run

}//ends Students class

public class Teacher extends Thread

{

//Teacher's Class Schedule Variables

public static int classlength = 10000;

static int ClassSched [] = {15000, 30000, 50000, 65000, 80000};

String OfficeHoursStarted = "No";

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

/\*\*

\* Gives the Teacher Threads the name Teacher when created and then starts them

\*/

Teacher()

{

setName("Teacher");

start();

}

/\*\*

\* This is an easier function to be used that just grabs the current time

\* of the system.

\* @return

\*/

public long currentTime()

{

return System.currentTimeMillis()-GlobalVariables.time;

}

/\*\*

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

\* The time is grabbed from the currentTime function so that teacher

\* teacher is on the same time schedule as the students.

\* @param m

\*/

public void msg(String m)

{

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

}

/\*\*

\* Runs the Thread that has been started.

\*/

public void run()

{

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

{

for (int i = 0; i < Students.Classes.length; i++)//Runs through each class

{

if (currentTime() >= ClassSched[i] && currentTime() <= ClassSched[i] + classlength)

{

/\*\*

\* If the time right now is in-between the time the class is

\* started and the time the class's length is over then the teacher

\* sends a signal to the student class that the class has started,

\* this function is in the teacherArrivesAtClass function.

\*/

if (ClassStarted[i] == "No")

{

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

}

ClassStarted[i] = "Yes";

GlobalVariables.ClassStatus[i] = "On";

GlobalVariables.TeacherArrives.teacherArrivesAtClass();

}

if (currentTime() > ClassSched[i] + classlength)

{

/\*\*

\* If this class is over the teacher sends another signal back to

\* the student this time by using the ClassStatus = "Over". The

\* student will exit its while loop. After the signal is sent the

\* for loop begins again and the next class beings.

\*/

if (ClassStarted[i] == "Yes")

{

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

}

ClassStarted[i] = "Done";

GlobalVariables.ClassStatus[i] = "Over";

}

}//ends for

//OFFICE HOURS

if (currentTime() >= ClassSched[1] + classlength && currentTime() <= ClassSched[2])

{

/\*\*

\* If the time is between the end of the second class and the beginning

\* of the third class then Office Hours begin for the Teacher.

\*/

if (OfficeHoursStarted == "No")

{

msg(" started Office Hours.");

}

OfficeHoursStarted = "Yes";

}

if (currentTime() > ClassSched[2] - 1)

{

/\*\*

\* Right before the third class begins office hours end and classes

\* resume.

\*/

if (OfficeHoursStarted == "Yes")

{

msg(" ended Office Hours.");

}

OfficeHoursStarted = "Done";

}

//END OF OFFICE HOURS

/\*\*

\* The join function kills off each thread individually

\*/

if (GlobalVariables.ClassStatus[Students.Classes.length-1] == "School's Over")

{

try

{

currentThread().join();

}

catch (InterruptedException e)

{

e.printStackTrace();

}

}//ends join

}//ends while

}//ends run

}//ends Teacher class