**Article count**

Multithreading is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of CPU. Each part of such a program is called a thread. The threads are light-weight processes within a process.  
           
Let's have a quick look at the way threads work in Java. For multi-threading to work, the class that will be invoked as a thread should extend the Thread class. You may wonder, what is the use of multi-threading. Let's understand it by the following exercise. Given 'n' number of lines of text, you have to find the total number of articles present in the given lines. while obtaining inputs from the user, the Main method has the full control of the execution.  
  
The time is wasted in input gathering, which can be invaluable for large computing applications, has to be utilized properly. Hence a thread is invoked when a line is obtained and the articles are counted while the input for the subsequent lines is obtained from the user. Thus threading can increase efficiency and time constraints.  
  
**Strictly adhere to the Object-Oriented specifications given in the problem statement. All class names, attribute names and method names should be the same as specified in the problem statement.**  
   
Create a class called **Article**which extends the **Thread**class with the following private attributes.

|  |  |
| --- | --- |
| **Attributes** | **Datatype** |
| line | String |
| count | Integer |

Include appropriate getters and setters.  
Generate default and parameterized constructors. The format for the parameterized constructor is **Article(String line)**  
  
The Article class includes the following methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| void run() | This method counts the number of articles in a given line and stores the value in the count variable. |

Create a driver class called **Main**. In the Main method, invoke 'n' threads for 'n' lines of input and compute the total count of the articles in the given lines.  
  
**Input and Output format:**  
Refer to sample Input and Output for formatting specifications.

**[All text in bold corresponds to the input and rest corresponds to the output]**

**Sample Input and Output:**

Enter the number of lines

**3**

Enter line 1

**An article is a word used to modify a noun, which is a person, place, object, or idea.**

Enter line 2

**Technically, an article is an adjective, which is any word that modifies a noun.**

Enter line 3

**There are two different types of articles.**

There are 7 articles in the given input

import java.util.regex.Matcher;

import java.util.regex.Pattern;

public class Article extends Thread {

String line;

int count;

public Article() {

super();

}

public Article(String line) {

super();

this.line = line;

}

public String getLine() {

return line;

}

public void setLine(String line) {

this.line = line;

}

public int getCount() {

return count;

}

public void setCount(int count) {

this.count = count;

}

@Override

public void run() {

Matcher m = Pattern.compile("(?i)\\b((a)|(an)|(the))\\b").matcher(this.getLine());

while (m.find()) {

this.count++;

}

}

}

import java.util.Scanner;

public class Main {

public static void main(String[] args) throws InterruptedException{

Scanner sc = new Scanner(System.in);

System.out.println("Enter the number of lines");

int n = sc.nextInt();

int count = 0;

sc.nextLine();

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

Article a = new Article();

System.out.println("Enter line "+(i+1));

a.setLine(sc.nextLine());

a.start();

a.join();

count +=a.getCount();

}

System.out.println("There are "+count+" articles in the given input");

}

}

**Profit or Loss**

We are going to create a console application that can estimate whether the booking is a profit or loss, thereby enabling hall owners to reduce or increase expenses depending on the status. Hence if several Booking details are given, compute whether the bookings are profitable or not. Use Threads to compute for each booking, Finally display the details along with the profit/loss status.  
  
**Strictly adhere to the Object-Oriented specifications given in the problem statement. All class names, attribute names and method names should be the same as specified in the problem statement.**

Create a class **Event**with the following private attributes

|  |  |
| --- | --- |
| **Attributes** | **Datatype** |
| name | String |
| hallbooking | HallBooking |

Include appropriate getters and setters.  
Create default constructor and a parameterized constructor with arguments in order**Event(String name,HallBooking hallbooking).**

Create a class **HallBooking**with following private attributes

|  |  |
| --- | --- |
| **Attributes** | **Datatype** |
| hallName | String |
| cost | Double |
| hallCapacity | Integer |
| seatsBooked | Integer |

Include appropriate getters and setters.  
Create default constructor and a parameterized constructor with arguments in order  
**HallBooking(String hallName, Double cost, Integer hallCapacity,Integer seatsBooked)**.

Create a class **ComputeStatus** that implements **Runnable interface** with **List<Event> eventList** attribute.  
  
Include following methods.  
**Override run() method** which displays event name along with their status (i.e) Profit or Loss.   
**If  (seats booked / hall capacity) \* 100 >= 60 then it is a profit else loss**.  
  
Create a driver class **Main** which creates **ThreadGroup** with two threads. Each thread will have half of the event details. After the first half of event details are obtained, invoke the first thread and after the other half is obtained invoke the second thread. The Threads print the status of the events. Use the join method appropriately to ensure printing the status in the correct order.

**Input Format:**  
  
The first line of input corresponds to the number of events 'n'.  
The next 'n' line of input corresponds to the event details in CSV format of (Event Name,Hall Name,Cost,Hall Capacity,Seats Booked).  
Refer to sample input for formatting specifications.  
  
**Output Format:**  
  
The output consists of event names with their status (Profit or Loss).  
Refer to sample output for formatting specifications.  
  
**Problem Constraints:**  
If n>0 and n then even. Otherwise, display as "**Invalid Input**".  
  
**Sample Input and Output 1:**  
**[All text in bold corresponds to input and rest corresponds to output]**  
  
Enter the number of events  
**4**  
Enter event details in CSV  
**Party,Le Meridian,12000,400,250  
Wedding,MS mahal,500000,1000,400  
Alumni meet,Ramans,10000,600,300  
Plaza party,Rizzodous,30000,1200,1000**  
Party yields profit  
Wedding yields loss  
Alumni meet yields loss  
Plaza party yields profit

import java.util.ArrayList;

import java.util.Iterator;

import java.util.List;

import java.util.Scanner;

public class Event {

private String name;

private HallBooking hallBooking;

public Event() {

}

public Event(String name, HallBooking hallBooking) {

this.name = name;

this.hallBooking = hallBooking;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public HallBooking getHallBooking() {

return hallBooking;

}

public void setHallBooking(HallBooking hallBooking) {

this.hallBooking = hallBooking;

}

}

import java.util.ArrayList;

import java.util.Iterator;

import java.util.List;

import java.util.Scanner;

public class ComputeStatus implements Runnable {

List<Event> eventList;

public List<Event> getEventList() {

return eventList;

}

public void setEventList(List<Event> eventList) {

this.eventList = eventList;

}

@Override

public void run() {

// System.out.println(Thread.currentThread().getName());

Iterator iterator = eventList.iterator();

try {

Thread.sleep(1000);

} catch (InterruptedException e) {

e.printStackTrace();

}

while (iterator.hasNext()) {

Event e = (Event) iterator.next();

if (e.getHallBooking().getSeatsBooked() \* 100 / e.getHallBooking().getHallCapacity() >= 60) {

System.out.println(e.getName() + " yields profit");

} else {

System.out.println(e.getName() + " yields loss");

}

}

}

}

public class HallBooking {

private String hallName;

private Double cost;

private Integer hallCapacity;

private Integer seatsBooked;

public HallBooking(String hallName, Double cost, Integer hallCapacity, Integer seatsBooked) {

this.hallName = hallName;

this.cost = cost;

this.hallCapacity = hallCapacity;

this.seatsBooked = seatsBooked;

}

public HallBooking() {

}

public String getHallName() {

return hallName;

}

public void setHallName(String hallName) {

this.hallName = hallName;

}

public Double getCose() {

return cost;

}

public void setCose(Double cose) {

this.cost = cose;

}

public Integer getHallCapacity() {

return hallCapacity;

}

public void setHallCapacity(Integer hallCapacity) {

this.hallCapacity = hallCapacity;

}

public Integer getSeatsBooked() {

return seatsBooked;

}

public void setSeatsBooked(Integer seatsBooked) {

this.seatsBooked = seatsBooked;

}

}

import java.util.ArrayList;

import java.util.Iterator;

import java.util.List;

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter the number of events");

int noOfEvents = Integer.parseInt(scanner.nextLine());

if (noOfEvents > 0 && noOfEvents % 2 == 0) {

System.out.println("Enter event details in CSV");

String eventDetail = "";

List<Event> eventList = new ArrayList<Event>();

ComputeStatus computeStatus = new ComputeStatus();

// computeStatus.setEventList(eventList);

ThreadGroup threadGroup = new ThreadGroup("TG1");

Thread t1 = new Thread(threadGroup, computeStatus, "T1");

Thread t2 = new Thread(threadGroup, computeStatus, "T2");

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

eventDetail = scanner.nextLine();

String[] details = eventDetail.split(",");

HallBooking hallBooking = new HallBooking(details[1],

Double.parseDouble(details[2]),

Integer.parseInt(details[3]),

Integer.parseInt(details[4]));

Event event = new Event(details[0], hallBooking);

eventList.add(event);

computeStatus.setEventList(eventList);

if (i + 1 == noOfEvents / 2) {

t1.start();

try {

t1.join();

} catch (InterruptedException e) {

e.printStackTrace();

}

eventList.clear();

}

if (i + 1 == noOfEvents) {

t2.start();

}

}

} else {

System.out.println("Invalid Input");

}

}

}

**Multi-Threading**

To illustrate the creation of multiple threads in a program performing concurrent operations, let us consider the processing of the following mathematical equation:  
**p = sin (x) + cos (y) + tan (z)**  
As these trigonometric functions are independent operations without any dependencies between them, they can be executed concurrently. After that their results can be combined to produce the final result.  
  
All three worker threads are concurrently executed on shared or dedicated CPUs depending on the type of machine. Although the master thread can continue its execution, in this case, it needs to make sure that all operations are completed before combining individual results. This is accomplished by waiting for each thread to complete by invoking **join()** method associated with each worker thread.  
  
The main thread is called **Main**, which acts like a master thread. It creates three worker threads (**SineClass**, **CosClass**, and **TanClass**) and assigns them to compute values for different data inputs.  
  
**Input & Output Format:**  
Refer sample Input and Output for formatting specifications.  
  
**Hint:**  
Use the following code snippet to print to 2 decimal places.  
import java.text.DecimalFormat;  
DecimalFormat df = new DecimalFormat("#.##");  
System.out.println("Sum of sin, cos, tan = " + df.format(z));  
  
**Sample Input and Output :**  
Enter the Degree for Sin :  
**45**  
Enter the Degree for Cos :  
**30**  
Enter the Degree for Tan :  
**30**  
Sum of sin, cos, tan = 2.15

public class CosClass extends Thread {

double value;

double cos;

public CosClass(double cos) {

// TODO Auto-generated constructor stub

this.cos = cos;

}

@Override

public void run() {

double b = Math.toRadians(cos);

value = Math.cos(b);

}

public double getValue() {

return value;

}

}

public class SineClass extends Thread {

double value;

double sins;

public SineClass(double sins) {

this.sins = sins;

}

@Override

public void run() {

double b = Math.toRadians(sins);

value = Math.sin(b);

}

public double getValue() {

return value;

}

}

public class TanClass extends Thread {

double value;

double tan;

public TanClass(double tan) {

// TODO Auto-generated constructor stub

this.tan = tan;

}

@Override

public void run() {

double b = Math.toRadians(tan);

value = Math.tan(b);

}

public double getValue() {

return value;

}

}

import java.text.DecimalFormat;

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

// TODO Auto-generated method stub

DecimalFormat df = new DecimalFormat("0.00");

Scanner sc = new Scanner(System.in);

System.out.println("Enter the Degree for Sin : ");

double sins = sc.nextDouble();

System.out.println("Enter the Degree for Cos : ");

double coss = sc.nextDouble();

System.out.println("Enter the Degree for Tan : ");

double tans = sc.nextDouble();

SineClass s = new SineClass(sins);

CosClass c = new CosClass(coss);

TanClass t = new TanClass(tans);

s.start();

try {

s.join();

} catch (InterruptedException e) {

e.printStackTrace();

}

c.start();

try {

c.join();

} catch (InterruptedException e) {

e.printStackTrace();

}

t.start();

try {

t.join();

} catch (InterruptedException e) {

e.printStackTrace();

}

System.out.println("Sum of sin, cos, tan = " + df.format(s.getValue() + t.getValue() + c.getValue()));

}

}Top of Form

Bottom of Form