**LAB 10.1**

*Write a program where, class cls2 will have a method display() – which prints table of 2 using FOR loop; now from another class cl1 (having main method) create two object s of cls2 andboth obejcts will call display method.*

*Now convert cls2 as a runnable one, using runnable implementation. Override run method, and call display from run method.*

*Now the constructor of cls2 should create two thread objects. And from cls1 now create only one object of cls2 and don’t call display method (as the display method will now be called from run method)*

*See the different in output in both the cases – you will find that in first scenario table of 2 was printing 2 times, one by one… using thread the table is printing simultaneously - Aim of this assignment is to see behave of thread, benefit of thread (as now the same code can be execute simultaneously by various thread) , how to use runnable interface to create thread object*

**Steps:**

* Write a class (say ThreadPractice having one method tableOfTwo() which display table of 2.
* Write another class ThreadMain which have p.s.v.main method
* Create two objects of ThreadPractice in main method and call tableOfTwo for both theobjects
* Execute ThreadMain class, note the output
* Now, Change the ThreadPractice and implement Runnable, override run() method (now run method will call tableOfTwo method
* In constructor of ThreadPractice , instantiate thread object and call its start method
* Modify ThreadMain class, now just create two objects of ThreadPractice class and do not call any method.
* Add the sleep method of thread class in tableOfTwo method, to provide a lag (just to observe the thread effect)
* Execute ThreadMain and compare the output with previous output

**package** cg.javaflp.practice;

**public** **class** ThreadPractice{

**public** **void** tableOfTwo(){

**for** (**int** i = 1; i <11; i++){

System.*out*.println(i\*2);

}

}

}

**public** **class** ThreadMain {

**public** **static** **void** main(String[] args) {

ThreadPractice practice1 = **new** ThreadPractice();

ThreadPractice practice2 = **new** ThreadPractice();

practice1.display();

practice2.display();

}

}

**Above code is normal programming (non thread), now let’s convert it into thread programming**

**package** cg.javaflp.practice;

**public** **class** ThreadPractice **implements** Runnable{

Thread thread;

**public** ThreadPractice() {

thread = **new** Thread(**this**);

thread.start();

}

@Override

**public** **void** run() {

display();

}

**public** **void** display(){

**for** (**int** i = 1; i <11; i++){

**try** {

thread.*sleep*(100);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

System.*out*.println(i\*2);

}

}

}

**THREAD MAIN CLASS**

**package** cg.javaflp.practice;

**public** **class** ThreadMain {

**public** **static** **void** main(String[] args) {

ThreadPractice practice1 = **new** ThreadPractice();

ThreadPractice practice2 = **new** ThreadPractice();

}

}

**Learning:**

* How to create thread using Runnable interface
* Difference between thread and normal programming

**LAB 10.1 A**

*Modify 10.1 now, also give names to the thread, and display them in result so that, we can have idea of which thread is executing when.*

**Steps:**

* Modify the constructor of ThreadPractice (as code given below in code snippet)
* While instantiating ThreadPractice, pass a name as argument

IN THREAD PRACTICE CLASS

/\* modified the constructor now it is taking name(String)

\* and this name will be set for thread

\*/

**public** ThreadPractice(String name) {

thread = **new** Thread(**this**, name);

thread.start();

}

MAKE THIS CHANGE IN TABLE OF TWO METHOD

Previous: System.*out*.println(i\*2);

Now: System.out.println(thread.getName() + i\*2);

IN THREAD MAIN CLASS

**public** **class** ThreadMain {

**public** **static** **void** main(String[] args) {

ThreadPractice practice1 = **new** ThreadPractice("First");

ThreadPractice practice2 = **new** ThreadPractice("Second");

}

}

**Learning:**

* Understanding of “new Thread (RunnableObject obj, String name) “ constructor

**LAB 10.2**

*Modify Lab 10.1 A; this time use Thread class (in placeof Runnable interface) to make objetc of Cls2 runnable.*

**Learning:**

* Write a new class (say ThreadExtendPractice) which extend thread class
* In thread Main class, instead of ThreadPractice now instantiate object of ThreadExtendClass

**package** cg.javaflp.practice;

**public** **class** ThreadExtendPractice **extends** Thread {

**public** ThreadExtendPractice(String name) {

**this**.setName(name);

**this**.start();

}

@Override

**public** **void** run() {

display();

}

**public** **void** display(){

**for** (**int** i = 1; i <11; i++){

**try** {

*sleep*(100);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

System.*out*.println(getName() + " " + i\*2);

}

}

}

**Learning:**

* How to create a runnable class by extending Thread class, How to use it

**LAB 10.3**

*Create a runnable class, and run multiple threads (We already did so in Lab 10.1 and 10.2) now use various methods of thread*

**Steps:**

* Add a method (threadMethodPracice) in ThreadPractice class and use various method of thread

**public** **void** run() {

threadMethodPracice();

}

**private** **void** threadMethodPracice() {

System.*out*.println(thread.getName());

System.*out*.println(thread.getPriority());

System.*out*.println(thread.isAlive());

System.*out*.println(thread.*MAX\_PRIORITY*);

System.*out*.println(thread.*MIN\_PRIORITY*);

System.*out*.println(thread.*NORM\_PRIORITY*);

System.*out*.println(thread.isDaemon());

thread.setPriority(8);

System.*out*.println(thread.getPriority());

System.*out*.println(thread);

}

**LAB 10.4**

*Write a stack-class, which should have Pop and push method and check if stack is full or empty prior to pushing or popping, so that out of array exception can never occur using this stack.*

*Now write a program, where one thread is pushing the objects in stack while other is popping it out, run the two threads simultaneously, so that array-out-of-bound exception occur; which means common resource (stack) is being used by two thread and chances of resource manipulation has occurred.*

*Aim of this assignment is to create thread safety issue.*

**Steps:**

* Create a stack (Say StacPrac) having Push and Pop method to push and pop object respectively; the size of stack will be determine at the time of creating stack object, and there must be methods like isEmpty and isFull to ensure that there should not be any ArrayOutOfBound Exception ever.
* Now write a call (say ThreadCall), having p.s.v.main method, this class will have a global final variable for stackPrac object of size 50
* In one method of this call, will have two inner thread class having run method; one thread will try to Push 50 objects and other thread will try to pop them out
* In normal scentrio it will run fine, now apply sleep method in both inner thread class of ThreadCall class and push and pop method of StackPrac class, and run it
* There will be ArrayOutOfBound exception, which is clear case of manipulation of common resource

**package** cg.javaflp.practice;

/\*\*

\* This class is basically creating a stack object

\* having push and pop behaviour and works on pricipal of LIFO

\* Instantiated class of this object will define the size of stack

\*/

**public** **class** StackPrac {

**int** top;

Object [] objArr;

/\*\*

\* Constructor of stack; it creates an empty stack and set it size

\* **@param** size - to define size of stack

\*/

**public** StackPrac(**int** size) {

objArr = **new** Object[size];

top = -1;

}

// to check if stack is empty

**private** **boolean** isEmpty(){

**return** top < 0;

}

// to check if stack is full

**private** **boolean** isFull(){

**return** top >= objArr.length;

}

/\*\*

\* method to Push objects in stack, not allow push, if stack is full

\* **@return** boolean value for confirmation

\*/

**public** **boolean** push(Object obj){

**if** (isFull()){

**return** **false**;

}

top ++;

**try** {

Thread.*sleep*(500);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

objArr[top] = obj;

**return** **true**;

}

/\*\*

\* Method to Pop objects from stack, not allow pop, if stack is empty

\* **@return** popped out Object,in case of empty stack it will return null

\*/

**public** Object pop(){

**if** (isEmpty()){

**return** **null**;

}

Object poppedObj = objArr[top];

top -- ;

**try** {

Thread.*sleep*(1000);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

**return** poppedObj;

}

}

**package** cg.javaflp.practice;

**public** **class** ThreadCall {

**final** StackPrac stackPrac = **new** StackPrac(50);

**public** **static** **void** main(String[] args) {

ThreadCall callObj = **new** ThreadCall();

callObj.call();

}

**public** **void** call(){

**new** Thread(){

**public** **void** run(){

**for** (**int** i = 0; i< 50; i ++){

**try** {

*sleep*(1000);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

**boolean** isPushed = stackPrac.push((Object)**new** Integer(i));

System.*out*.println(i + " is pushed " + isPushed);

}

}

}.start();

**new** Thread(){

**public** **void** run(){

**for** (**int** i = 0; i< 50; i ++){

**try** {

*sleep*(500);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

Object obj = stackPrac.pop();

System.*out*.println("Object " + obj + " is pooped");

}

}

}.start();

}

}

**OUTPUT**

Object null is pooped

Object null is pooped

For i = 0 is pushed true

Object 0 is pooped

For i = 1 is pushed true

Object 0 is pooped

For i = 2 is pushed true

Object 2 is pooped

Exception in thread "Thread-0" java.lang.ArrayIndexOutOfBoundsException: -1

at cg.javaflp.practice.StackPrac.push(StackPrac.java:55)

at cg.javaflp.practice.ThreadCall$1.run(ThreadCall.java:23)

Object 3 is pooped

**LAB 10.5**

*Make the program of LAB 10.4 as thread safe with the help of synchronization.*

**LAB 10.6**

*Write a program, and use various methods like wait(), notifyAll(), yeild(), etc.*

**LAB 10.7**

*Write a program, which creates a dead lock situation.*