**Topics**

Multithreading

Synchronization

Competitive Synchronization

Cooperative Synchronization

Synchronized block

Wait and Notify

**Description**

Create a GUI program that provides for writing to a file using two threads. Each thread will write a specified message, a specified number of times to the specified file. For example, thread 1 may write �Hi� 10 times to abc.txt and thread 2 may write �Bye� 10 times to same file. The program will run in the following modes:

No synchronization. (NoSync Write)

With competitive synchronization. (CompSync Write)

With cooperative synchronization. (CoopSync Write)

**GUI**

The GUI will contain three panels:

The North panel will contain the following:

A labeled text field for entering message 1 to be written by thread 1.

A labeled text field for entering message 2 to be written by thread 2.

A labeled text field count for entering the number of times the message1 is to be written by thread 1 and message 2 is to be written by thread 2.

A labeled text field for entering the name of the file where the messages are to be written.

The Center panel will contain scrollable text area for displaying the content of the file or other messages.

The South panel will contain the following:

A button labeled �Write� for running the two threads writing to the same file without synchronization.

A button labeled �Synch Write� for running the two threads writing to the same file with synchronization (i.e. synchronized block).

A button labeled �Coop Write� for running the two threads writing to the same file using cooperative synchronization (i.e. wait/notify).

A button for displaying the content of the file in the text area.

A button for clearing the contents of the text area and all the text fields.

**Implementation**

**Class NoSyncRunnable**

Create a class NoSyncRunnable that will implement Runnable.

It will provide a String variable message, an int variable count, and a String variable fileName.

It will provide a constructor with three parameters for initializing message, count, and fileName variables.

It will over-ride the run method.

In the run method, in a loop, it will write the contents of the �message� variable on a line by itself, a �count� number of times, to the file with the name in variable �fileName�.

After creating the object PrintWriter for writing to the file in append mode with autoflush on, it will sleep [Thread.sleep(100)] for 100 ms to allow other threads to run. This will be done to artificially create an end of the time slice and allow other threads to run.

After writing each line, it will sleep [Thread.sleep(100)] for 100 ms to allow other threads to run. This will be done to artificially create an end of the time slice and allow other threads to run.

**Class CompSyncRunnable**

Create a class CompSyncRunnable that will implement Runnable.

�        It will provide a String variable message, an int variable count, a String variable fileName, and an Object variable obj.

�        It will provide a constructor with four parameters for initializing message, count, fileName, and object variable obj.

�        It will over-ride the run method.

�        In the run method, in a loop, it will write the contents of the �message� variable on a line by itself, a �count� number of times, to the file with the name in variable �fileName�.

�        After creating the object PrintWriter for writing to the file in append mode with autoflush on, it will sleep [Thread.sleep(100)] for 100 ms to allow other threads to run. This will be done to artificially create an end of the time slice and allow other threads to run.

�        After writing each line, it will sleep [Thread.sleep(100)] for 100 ms to allow other threads to run. This will be done to artificially create an end of the time slice and allow other threads to run.

�        Above the whole writing loop will be within a synchronized block which will synchronize on Object obj as below. (Code shown without try/catch).

//Create a PrintWriter object to write in append mode with autoflush on.

PrintWriter pw = new PrintWriter (new FileWriter (fileName), true), true);

Thread.sleep (100);

synchronized (obj) {

//The whole write loop is here

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

pw.println (message);

Thread.sleep (100);

}

}

**Class CoopSyncRunnable**

Create a class CoopSyncRunnable that will implement Runnable.

�        It will provide a String variable message, an int variable count, a String variable fileName, and an Object variable obj.

�        It will provide a constructor with four parameters for initializing message, count, fileName, and object variables.

�        It will over-ride the run method.

�        In the run method, in a loop, it will write the contents of the �message� variable on a line by itself, a �count� number of times, to the file with the name in variable �fileName�.

�        After creating the object PrintWriter for writing to the file in append mode with autoflush on, it will sleep [Thread.sleep(100)] for 100 ms to allow other threads to run. This will be done to artificially create an end of the time slice and allow other threads to run.

�        Then it will write lines using a loop.

�        After writing each line, it will issue a notify on Object obj to wake up the other thread waiting on Object obj and then issue a wait on Object obj to put itself in wait state.

�        Above will be within a synchronized block which will synchronize on Object obj as shown in code below.

�        At the end of the write loop, one of the thread will be in wait state. After exiting the loop, each thread will issue a notify on Object obj in a synchronized block. The notify will wake up the thread in wait state as shown below. (Code shown without try/catch).

PrintWriter pw = new PrintWriter (new FileWriter (fileName), true), true);

Thread.sleep (100);

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

synchronized (obj) {

        //A single write within the loop will be here

        pw.println (message);

        Thread.sleep (100);

        obj.notify ( );

        obj.wait ( );

}

Thread.sleep (100);

}

synchronized (obj) {

        obj.notify ( );

}

**Class JFrameExt**

Create a class JFrameExt that will contain the following:

A constructor JFrameExt that will create the Graphical User Interface (GUI).

An actionPerformed event handler that will be invoked when the user clicks the �NoSync� button and will do the following:

        It will have local variables: String message1, String message2, int count, String fileName.

        It will obtain the values of these variables from appropriate text fields as entered by the user.

It will create an instance of class NoSyncRunnable and pass its constructor message1, count, and fileName.

It will create an instance of Threadand pass its constructor the instance of NoSynRunnable.

It will start t1, and t2.

An actionPerformed event handler that will be invoked when the user clicks the �CompSynch� button.

        It will create a new Object object as below:

                Object object = new Object ( );

        It will have local variables: String message1, String message2, int count, String fileName.

        It will obtain the values of these variables from appropriate text fields as entered by the user.

It will create an instance t1 of ThreadExtSynch and pass its constructor message1, count, fileName, and object.

It will create an instance t2 of ThreadExtSynch and pass its constructor message2, count, fileName, and object.

It will start t1, and t2.

An actionPerformed event that will be invoked when the user clicks the �CoopSync� button.

It will create a new Object object as below:

                Object object = new Object ( );

        It will have local variables: String message1, String message2, int count, String fileName.

        It will obtain the values of these variables from appropriate text fields asd entered by the user.

It will create an instance t1 of ThreadExtCoop and pass its constructor message1, count, fileName, and object.

It will create an instance t2 of ThreadExtCoop and pass its constructor message2, count, fileName, and object.

It will start t1, and t2.

**Class TestThreadExt**

Create a class TestJFrameExt containg the main method. Using the main method, it will create and instance of JFrameExt and display the frame.

**Testing**

**Test Run 1**

Enter �Hi� in text field message 1, �Bye� in text field message 2, 10 in text field count, �Write.txt� in text field File Name and press the �Write� button.

Then press the �Display� button to see the contents of the file.

The output from the two threads (lines containing �Hi� and �Bye�) will be intermixed in the file. A few lines may altogether be missing.

Since the writing to the file by the two threads is not synchronized. Each one writes when it�s running. Therefore the output is unpredictable.

**Test Run 2**

Enter �Hi� in text field message 1, �Bye� in text field message 2, 10 in text field count, �SynchWrite.txt� in text field File Name and press the �Synch Write� button.

Then press the �Display� button to see the contents of the file.

A complete output from one thread (e.g. 10 lines containing �Hi�) will be followed by a complete output from the other thread (e.g. 10 lines containing �Bye�).

This test shows that the outputs from the two threads were perfectly synchronized. One thread completed its output first. Then the other thread completed its output. This synchronization occurred in spite of the fact that each thread issued a sleep after writing each line to let the other thread run. The other thread did run but didn�t output a line because the other thread had the monitor. It waited till the monitor was released to start outputting.

**Test Run 3**

Enter �Hi� in text field message 1, �Bye� in text field in message 2, 10 in text field count, �CoopWrite.txt� in text field File Name and press the �Coop Write� button.

Then press the �Display� button to see the contents of the file.

The writes from the two threads will alternate perfectly (e.g. First Line: �Hi�, Second Line: �Bye�, Third Line: �Hi�, Fourth Line: �Bye� and so on).

This test shows that the writes in two threads cooperatively alternated using wait and notify. Also the call to wait and notify were themselves synchronized within a synchronize block so that they occur synchronously. (Java require that wait and notify be synchronized in a synchronize block with the same object upon which they are called).

**Turn In**

The contents of source files.

Three pictures of GUI, one for each run.

**GUI**

