Batch: A3 Roll No.: 1911034

Experiment / assignment / tutorial No.08

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

TITLE : Multithreading Programming

AIM:
Develop a java program using multithreading in which each thread reads the data from a text file and display the data of each file on Console alternatively such that one line from first input file is printed and then one line from another input file is printed and so on.

Detailed Description:
You are required to develop a java program containing two threads that reads two files simultaneously. In a program, first thread reads data from file "Personal_Record.txt" and second thread reads the data from file "Academic_Record.txt". When one thread reads a line from one file then it should allow another thread to read a line from another file. After reading data from each file, program must write the output on console (Output screen) such that one line from first input file is printed and then one line from another input file is printed and so on.

Expected OUTCOME of Experiment:

CO4: Demonstrate programs on interface, exceptions, multithreading and applets.

Books/ Journals/ Websites referred:

1.Ralph Bravaco , Shai Simoson , "Java Programing From the Group Up" Tata McGraw-Hill.

2.Grady Booch, Object Oriented Analysis and Design .

Pre Lab/ Prior Concepts:
Java provides built-in support for *multithreaded programming*. A multithreaded program contains two or more parts that can run concurrently. Each part of such a program is called a thread, and each thread defines a

separate path of execution. A multithreading is a specialized form of multitasking. Multithreading requires less overhead than multitasking processing.
Multithreading enables you to write very efficient programs that make maximum use of the CPU, because idle time can be kept to a minimum.

Creating a Thread:
Java defines two ways in which this can be accomplished:

1. You can implement the Runnable interface.
2. You can extend the Thread class itself.

Create Thread by Implementing Runnable:

The easiest way to create a thread is to create a class that implements the Runnable interface.
To implement Runnable, a class needs to only implement a single method called run( ), which is declared like this:
public void run( )
You will define the code that constitutes the new thread inside run() method. It is important to understand that run() can call other methods, use other classes, and declare variables, just like the main thread can.

After you create a class that implements Runnable, you will instantiate an object of type Thread from within that class. Thread defines several constructors. The one that we will use is shown here:

```
Thread(Runnable threadOb, String threadName);
```

Here, threadOb is an instance of a class that implements the Runnable interface and the name of the new thread is specified by threadName.
After the new thread is created, it will not start running until you call its start( ) method, which is declared within Thread. The start( ) method is shown here:

```
void start( );
```

Here is an example that creates a new thread and starts it running:

```
class NewThread implements Runnable {
   Thread t;
   NewThread() {
      t = new Thread(this, "Demo Thread");
      System.out.println("Child thread: " + t);
      t.start(); // Start the thread
   }
      public void run() {
      try {
```

```
         for(int i = 5; i > 0; i--) {
            System.out.println("Child Thread: " + i);
            // Let the thread sleep for a while.
            Thread.sleep(50);
         }
      } catch (InterruptedException e) {
         System.out.println("Child interrupted.");
      }
      System.out.println("Exiting child thread.");
   }
}

public class ThreadDemo {
   public static void main(String args[]) {
      new NewThread();
      try {
         for(int i = 5; i > 0; i--) {
           System.out.println("Main Thread: " + i);
           Thread.sleep(100);
         }
      } catch (InterruptedException e) {
         System.out.println("Main thread interrupted.");
      }
      System.out.println("Main thread exiting.");
   }
}
```

The second way to create a thread is to create a new class that extends **Thread**, and then to create an instance of that class.
The extending class must override the **`run( )`** method, which is the entry point for the new thread. It must also call **`start( )`** to begin execution of the new thread.

```
class NewThread extends Thread {
   NewThread() {
      super("Demo Thread");
      System.out.println("Child thread: " + this);
      start(); // Start the thread
   }
     public void run() {
      try {
         for(int i = 5; i > 0; i--) {
            System.out.println("Child Thread: " + i);
                        //  Let  the  thread  sleep  for  a
while.
```

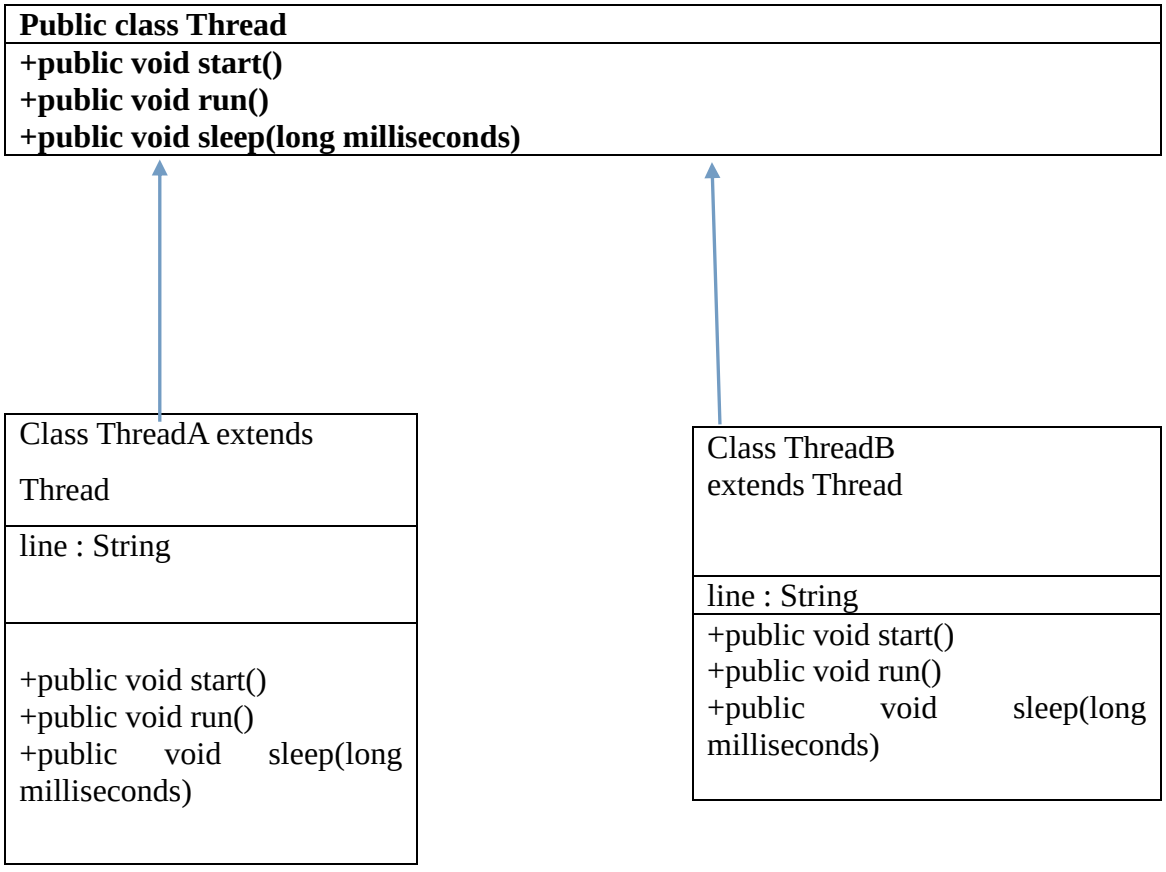
```
            Thread.sleep(50);
         }
      } catch (InterruptedException e) {
         System.out.println("Child interrupted.");
      }
      System.out.println("Exiting child thread.");
   }
}

public class ExtendThread {
   public static void main(String args[]) {
      new NewThread(); // create a new thread
      try {
         for(int i = 5; i > 0; i--) {
            System.out.println("Main Thread: " + i);
            Thread.sleep(100);
         }
      } catch (InterruptedException e) {
         System.out.println("Main thread interrupted.");
      }
      System.out.println("Main thread exiting.");
   }
}
```

Some of the Thread methods

methods	description
`void setName(String name)`	Changes the name of the Thread object. There is also a getName() method for retrieving the name
`Void setPriority(int priority)`	Sets the priority of this Thread object. The possible values are between 1 and 10. 5
`boolean isAlive()`	Returns true if the thread is alive, which is any time after the thread has been started but before it runs to completion.
`void yield()`	Causes the currently running thread to yield to any other threads of the same priority that are waiting to be scheduled.

`void sleep(long millisec)`	Causes the currently running thread to block for at least the specified number of milliseconds.
`Thread currentThread()`	Returns a reference to the currently running thread, which is the thread that invokes this method.

Class Diagram:



Algorithm:

Step1: Create two classes ThreadA , and ThreadB that extends the Thread superclass.

Step2: In these two classes , we override the run() method to read the data from the text files .

Step3 : In both the Files , create an object , obj , of the File class, using a try catch block, the catch block will catch the FileNotFoundException object e and print it.

Step4: In ThreadA , obj is made to read from Personal_Record.txt and in ThreadB obj is reading from Academic_Record.txt.

Step5: In both classes create a Scanner object , and pass the File as an argument to the scanner class , so that the scanner object can read from the respective files of the Threads.

Step6: Using hasNextLine() method , we read and print every line that we get in the file and print it on the screen. (Reading is done using object of scanner class and printing is done using object of String class which holds the line currently being read).

Step7: After reading and printing a line from a file , we make use of the sleep() method , which causes the currently running thread to sleep, for the specified period of time and transfers the control to the next thread

Step8: We enclose the sleep() method in a try-catch block.

Step9: In the main() method , we create an object each of ThreadA and ThreadB and we call the start method which calls the run method and executes both the threads to read and print lines alternatively from each file.

Implementation details:(printout of code)

```
import java.util.*;
import java.io.*;

public class Exp08
{

public static void main(String args[])
{
ThreadA t1 = new ThreadA();
ThreadB t2 = new ThreadB();
t1.start();
t2.start();
}

}



class ThreadA extends Thread
{

public void run()

{
try{
File obj1 = new File("Personal_Record.txt");
if(!obj1.exists()){
      throw new FileNotFoundException ("File Not found");
    }

else
{
Scanner sc = new Scanner(obj1);
```

```
while(sc.hasNextLine())
{
String line = sc.nextLine();
System.out.println(" "+line);
try
{
sleep(200); /*after printing this line, the thread sleeps so that the
other thread can start printing*/
}

catch(InterruptedException e)
{
  System.out.println(e);
}

}
sc.close();

}
}

catch(FileNotFoundException e)
{
System.out.println(e);
}
}

}
```

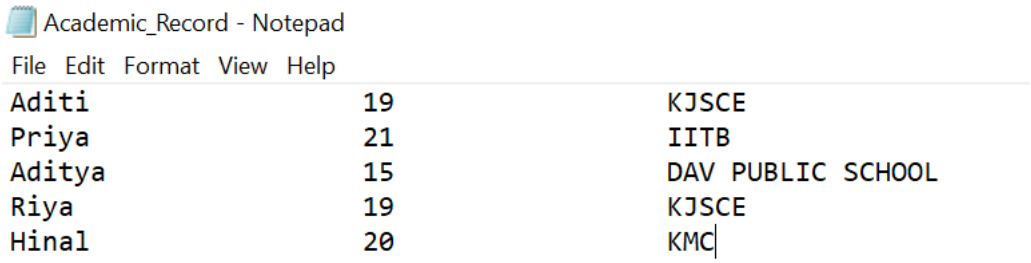
```
class ThreadB extends Thread
{

public void run()

{
```

```
try{
File obj1 = new File("Academic_Record.txt");
if(!obj1.exists()){
      throw new FileNotFoundException ("File Not found");
    }

else
{
Scanner sc = new Scanner(obj1);
while(sc.hasNextLine())
{
String line = sc.nextLine();
System.out.println(" "+line);
try
{
sleep(200); /*after printing this line, the thread sleeps so that the
other thread can start printing*/
}

catch(InterruptedException e)
{
  System.out.println(e);
}

}
sc.close();

}

}catch(FileNotFoundException e)
{
System.out.println(e);
}
}

}
```

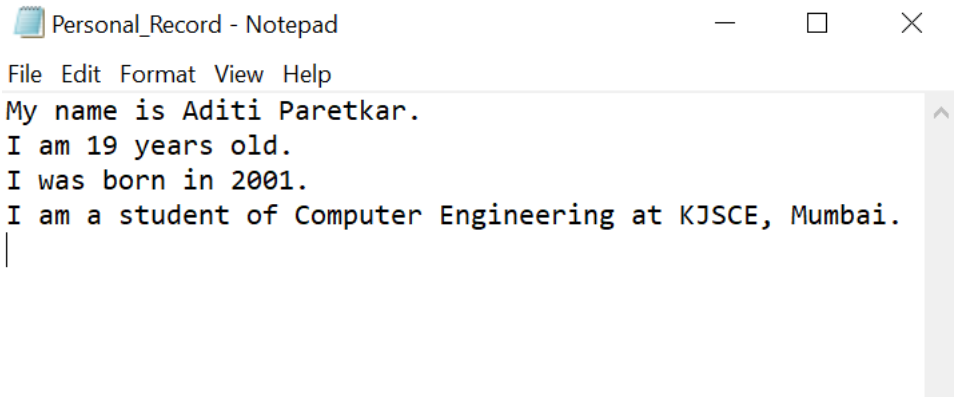
Output Screenshots :

```
C:\Users\arvin\Desktop>java Exp08
 Aditi                  19                  KJSCE
 My name is Aditi Paretkar.
 Priya                  21                  IITB
 I am 19 years old.
 Aditya                 15                  DAV PUBLIC SCHOOL
 I was born in 2001.
 Riya                   19                  KJSCE
 I am a student of Computer Engineering at KJSCE, Mumbai.
 Hinal                  20                  KMC
```

Academic_Record.txt



Personal_Record.txt



Conclusion:

In this experiment , we have learnt how to use threads , in order to execute 2 parts of a process simultaneously , and we have also created 2 separate threads to read and display the data in the form of lines from two text files onto the screen , alternatively.

We have used the sleep() method , which , after executing the process for a currently running thread causes the thread to sleep for a given amount of time as specified in the argument.

In this case , the process to be executed is reading and displaying the lines onto the screen.

Date:_____ **Signature of faculty in-charge**

Post Lab Descriptive Questions (Add questions from examination point view)

1.What do you mean by multithreading?

Multithreading is a feature of Java which allows us to execute concurrent parts of a program for maximum utilization of the CPU. Each part of a program is called as a thread.
A thread has a life cycle as follows :

- **New** − A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread. It is also referred to as a **born thread**.
- **Runnable** − After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be executing its task.
- **Waiting** − Sometimes, a thread transitions to the waiting state while the thread waits for another thread to perform a task. A thread transitions back to the runnable state only when another thread signals the waiting thread to continue executing.
- **Terminated (Dead)** − A runnable thread enters the terminated state when it completes its task or otherwise terminates.

.2 Explain the use of sleep and run function with an example?

Sleep() :

The sleep function is used to put a currently running thread into 'sleep' mode for a given amount of time as specified in the parameter , in milliseconds.

Example :

This program uses two threads , one to print $ symbol and the other to print numbers from 1 to 9 in the following fashion :

$1$2$3$4$5$6$7$8$9

```
class Main {
  public static void main(String[] args) {
    System.out.println("Hello world!");
    ThreadA t1 = new ThreadA();
    ThreadB t2 = new ThreadB();
    t1.start();
    t2.start();
  }
}

class ThreadA extends Thread
{
  public void run()
  {
    int i;
    for(i=0;i<9;i++)
    {
      System.out.print("$");
      try{
        sleep(200);
      }
   catch(InterruptedException e)
   {
     System.out.print(e);
   }
    }
  }
}

class ThreadB extends Thread

{
```

```
  public void run()
  {
    int i;
    for(i=1;i<10;i++)
    {
      System.out.print(i);
      try{
        sleep(200);
      }
   catch(InterruptedException e)
   {
     System.out.print(e);
   }
    }
  }
}
```

```
> javac -classpath .:/run_dir/junit-4.12.jar:target/dependency/* -d . Main.java
> java -classpath .:/run_dir/junit-4.12.jar:target/dependency/* Main
Hello world!
$1$2$3$4$5$6$7$8$9>
```

Run ()

The run method is defined in the Runnable interface or in the class Thread. We have to override this method whenever we have to use the a user defined thread class in order for the thread to run or execute

Example :

```
class Main {
  public static void main(String[] args) {
    System.out.println("Hello world!");
    ThreadA t1 = new ThreadA();

    t1.start();

  }
}
```

```
class ThreadA extends Thread

{
  public void run()
  {
    char i;
    for(i='a';i<='z';i++)
    {
      System.out.print(i);
      try{
        sleep(200);
      }
   catch(InterruptedException e)
   {
     System.out.print(e);
   }
    }
  }
}
```

```
> javac -classpath .:/run_dir/junit-4.12.jar:target/dependency/* -d . Main.java
> java -classpath .:/run_dir/junit-4.12.jar:target/dependency/* Main
Hello world!
abcdefghijklmnopqrstuvwxyz>
```