**LAB # 3**

**Introduction to Concurrency**

**OBJECTIVE**

Understanding and implementing the concept of concurrency through different mechanisms of multithreading.

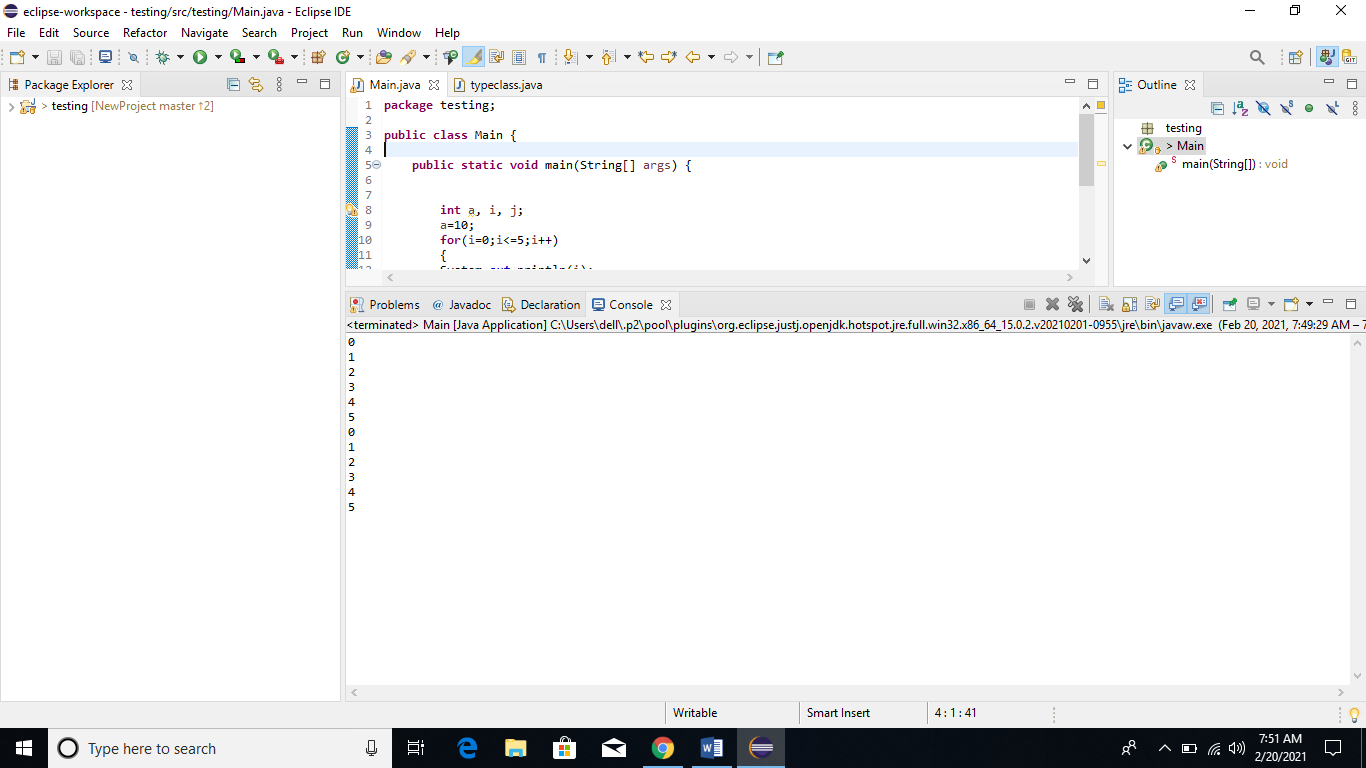
**THEORY:**

Threading:

A thread is a lightweight sub-process, the smallest unit of processing. It is a separate path of execution. Threads are independent. If there occurs exception in one thread, it doesn't affect other threads. It uses a shared memory area. Consider the following example:

|  |
| --- |
| **public** **class** Main {  **public** **static** **void** main(String[] args) {    **int** i, j;    **for**(i=0;i<=5;i++)  {  System.***out***.println(i);  }  **for**(j=0;j<=5;j++)  {  System.***out***.println(j);  }  }  } |

**Output:**



Multithreading:

Multithreading in Java is a process of executing multiple threads simultaneously. However, we use multithreading than multiprocessing because threads use a shared memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process. Java Multithreading is mostly used in games, animation, etc.

Advantages of Java Multithreading

* It doesn't block the user because threads are independent and you can perform multiple operations at the same time.
* You can perform many operations together, so it saves time.
* Threads are independent, so it doesn't affect other threads if an exception occurs in a single thread.

Multitasking

Multitasking is a process of executing multiple tasks simultaneously. We use multitasking to utilize the CPU. Multitasking can be achieved in two ways:

* Process-based Multitasking (Multiprocessing)
* Thread-based Multitasking (Multithreading)

**1) Process-based Multitasking (Multiprocessing)**

Each process has an address in memory. In other words, each process allocates a separate memory area. A process is heavyweight. Cost of communication between the process is high. Switching from one process to another requires some time for saving and loading registers, memory maps, updating lists, etc.

**2) Thread-based Multitasking (Multithreading)**

Threads share the same address space. A thread is lightweight. Cost of communication between the thread is low.

Note: At least one process is required for each thread.

Methods of creating threads:

There are two ways to create a thread:

* By extending Thread class
* By implementing Runnable interface.

**Java Thread Example by extending Thread class**

|  |
| --- |
| **class** Main **extends** Thread{  **public** **void** run(){  System.***out***.println("thread is running...");  }  **public** **static** **void** main(String args[]){  Main t1=**new** Main();  t1.start();  }  } |

**Java Thread Example by implementing Runnable interface**

|  |
| --- |
| **class** Main **implements** Runnable{  **public** **void** run(){  System.***out***.println("thread is running...");  }    **public** **static** **void** main(String args[]){  Main m1=**new** Main();  Thread t1 =**new** Thread(m1);  t1.start();  }  } |

Some simple examples of threading with built-in methods:

**Naming Thread**

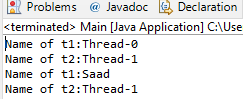
The Thread class provides methods to change and get the name of a thread. By default, each thread has a name i.e. thread-0, thread-1 and so on. By we can change the name of the thread by using setName() method. The syntax of setName() and getName() methods are given below:

public String getName(): is used to return the name of a thread.

public void setName(String name): is used to change the name of a thread.

|  |
| --- |
| **class** Main **extends** Thread{  **public** **void** run(){  System.***out***.println("running...");  }  **public** **static** **void** main(String args[]){  Main t1=**new** Main();  Main t2=**new** Main();  System.***out***.println("Name of t1:"+t1.getName());  System.***out***.println("Name of t2:"+t2.getName());    t1.start();  t2.start();    t1.setName("Saad");  System.***out***.println("After changing name of t1:"+t1.getName());  }  } |

**Output:**



**Priority of a Thread (Thread Priority):**

Each thread have a priority. Priorities are represented by a number between 1 and 10. In most cases, thread schedular schedules the threads according to their priority (known as preemptive scheduling). But it is not guaranteed because it depends on JVM specification that which scheduling it chooses.

3 constants defined in Thread class:

public static int MIN\_PRIORITY

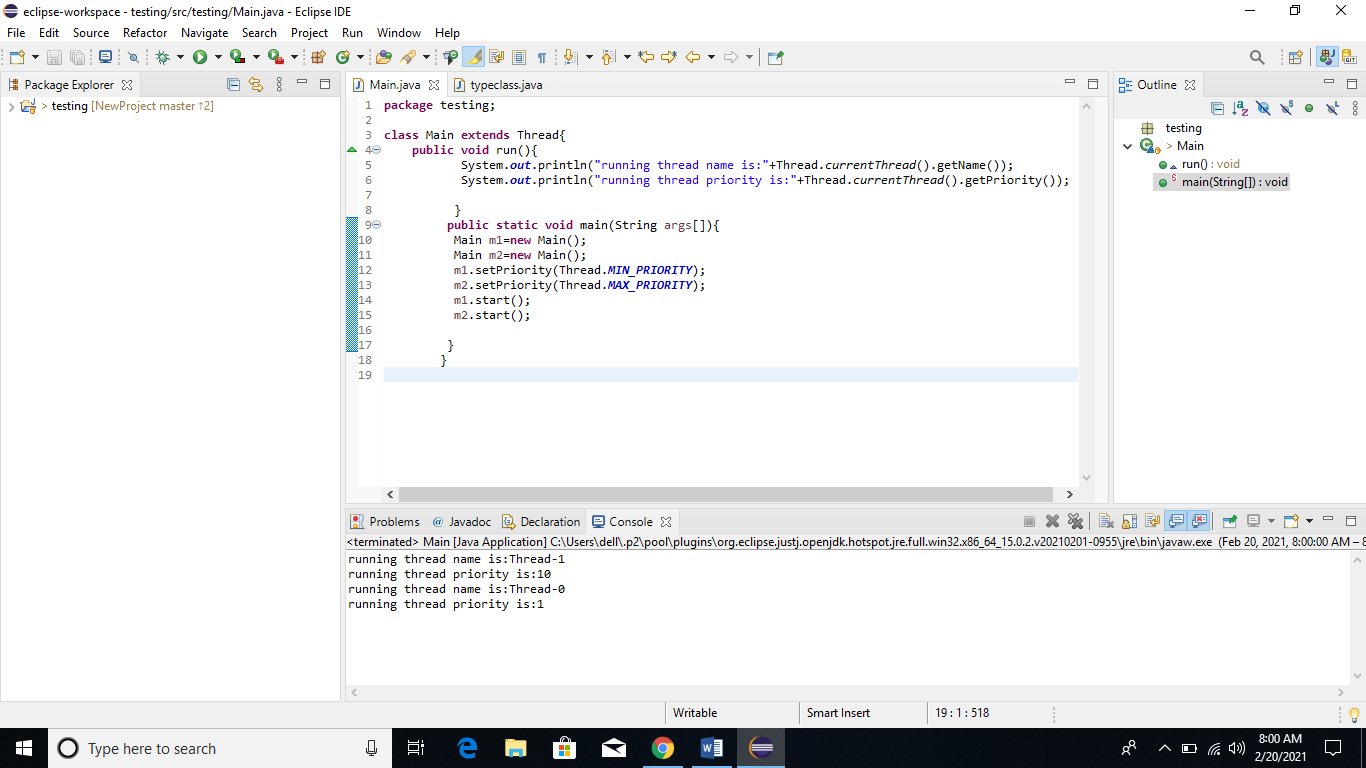
public static int NORM\_PRIORITY

public static int MAX\_PRIORITY

Default priority of a thread is 5 (NORM\_PRIORITY). The value of MIN\_PRIORITY is 1 and the value of MAX\_PRIORITY is 10.

|  |
| --- |
| **class** Main **extends** Thread{  **public** **void** run(){  System.***out***.println("running thread name is:"+Thread.*currentThread*().getName());  System.***out***.println("running thread priority is:"+Thread.*currentThread*().getPriority());    }  **public** **static** **void** main(String args[]){  Main m1=**new** Main();  Main m2=**new** Main();  m1.setPriority(Thread.***MIN\_PRIORITY***);  m2.setPriority(Thread.***MAX\_PRIORITY***);  m1.start();  m2.start();    }  } |

**Output:**



**Lab Task:**

1. Implement the following program on eclipse IDE and answer the following questions:

* How many threads are running?
* How many tasks are running?
* If more tasks are added than what will be the impact on number of threads?
* Explain the flow of program:

|  |
| --- |
| **class** Main **extends** Thread{  **public** **void** run(){  System.***out***.println("task one");  }  **public** **static** **void** main(String args[]){  Main t1=**new** Main();  Main t2=**new** Main();  Main t3=**new** Main();  t1.start();  t2.start();  t3.start();  }  } |

1. With the help of threading print two tables concurrently, print one table number of student roll number e.g. 2019-SE-092 and second number should be date of birth e.g. 05-April.