Shaikh Salif

S22-94

EXPERIMENT-14

Aim: Implement a Program on Multithreading

Theory:

Multithreading in Java

Multithreading in Java

is a process of executing multiple threads simultaneously.

A thread is a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

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

1) It **doesn't block the user** because threads are independent and you can perform multiple operations at the same time.

2) You **can perform many operations together, so it saves time**.

3) 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.

What is Thread in java

A thread is a lightweight subprocess, 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.

Code:

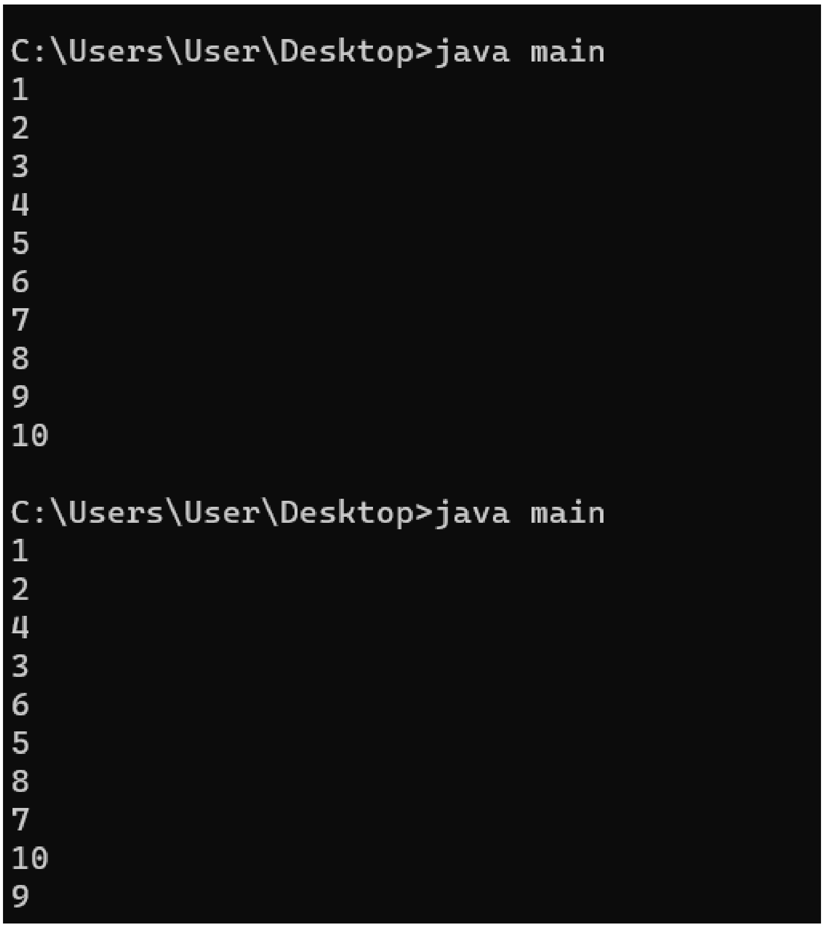
```
class Odd extends Thread

{

public void run()

{

int i;

for(i=1;i<=10;

i+=2)


{System.out.print

ln(i);try
```

```
{

Thread.sleep(100);
}
catch(Exception e)
{
}
}
}
}
class Even extends Thread
{
public void run()
{
int j;
for(j=2;j<=10;
j+=2)
{
System.out.print
ln(j); try

{
Thread.sleep(100);
}
```

```
catch(Exception e)
{
}
}
}
}
class main{
public static void main(String args[])
{
Odd n = new
Odd();n.start();
// Thread t1 = new
Thread(n);Even n1 =
new Even(); n1.start();
//Thread t2 = new Thread(n1);
//t1.start();
//t2.start(
);try
{
Thread.sleep(100);
}
catch(Exception e)
{
```

```
System.out.println("Hello");

}
}
}
```

Output:



```
C:\Users\User\Desktop>java main
1
2
3
4
5
6
7
8
9
10

C:\Users\User\Desktop>java main
1
2
4
3
6
5
8
7
10
9
```

Conclusion: In conclusion, a program that incorporates multithreading is a valuable and powerful approach to optimizing and enhancing the performance and responsiveness of software applications.