ENTERPRISE ARCHITECTURE

Lab Sheet 1 - Thread

GAM/IT/2022/F/0096

Part 1: Introduction to Threads in Java

Create a Simple Thread Class

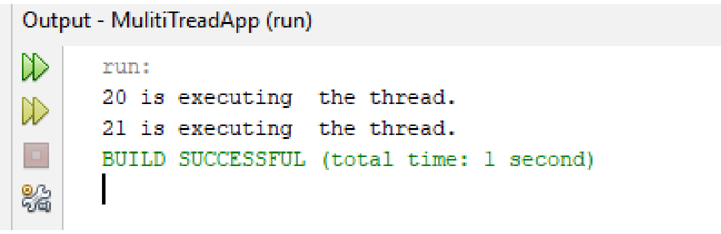
1.Open your Java IDE and create a new project named MultiThreadApp.

```
package mulititreadapp;
public class MulitiTreadApp {
    public static void main(String[] args) {
        // TODO code application logic here
    }
}
```

2. Inside the project, create a new class called SimpleThread.java.

```
package SimpleThread.java;
public class SimpleThread extends Thread{
     @Override
     public void run() {
 System.out.println(Thread.currentThread().getId() + " is executing  the thread.");
 }
 public static void main(String[] args) {
 SimpleThread thread1 = new SimpleThread();
 SimpleThread thread2 = new SimpleThread();
 thread1.start(); // Starts thread1
 thread2.start(); // Starts thread2
 }
}
```

Output:-



```
run:
20 is executing  the thread.
21 is executing  the thread.
BUILD SUCCESSFUL (total time: 1 second)
```

Part 2: Using Runnable Interface

Create a Runnable Class

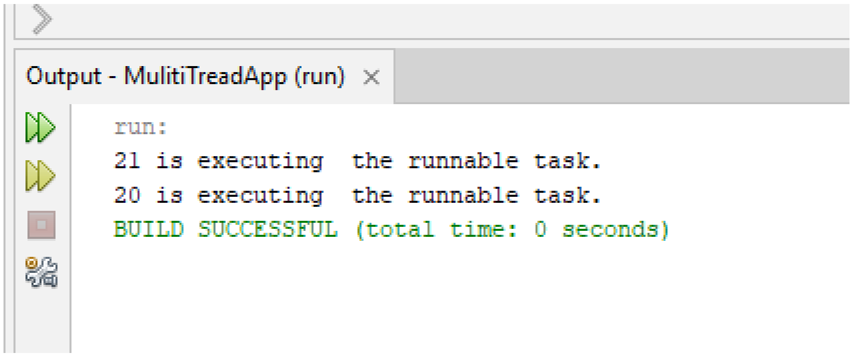
1. Create a new class called RunnableTask.java.

```
public class RunnableTask implements Runnable {
    @Override
 public void run() {
 System.out.println(Thread.currentThread().getId() + " is executing  the runnable task.");
 }
 public static void main(String[] args) {
 RunnableTask task1 = new RunnableTask();
 RunnableTask task2 = new RunnableTask();

 Thread thread1 = new Thread(task1);
 Thread thread2 = new Thread(task2);

 thread1.start(); // Starts thread1
 thread2.start(); // Starts thread2
 }
}
```

Output :-



```
run:
21 is executing  the runnable task.
20 is executing  the runnable task.
BUILD SUCCESSFUL (total time: 0 seconds)
```

Part 3: Synchronizing Threads

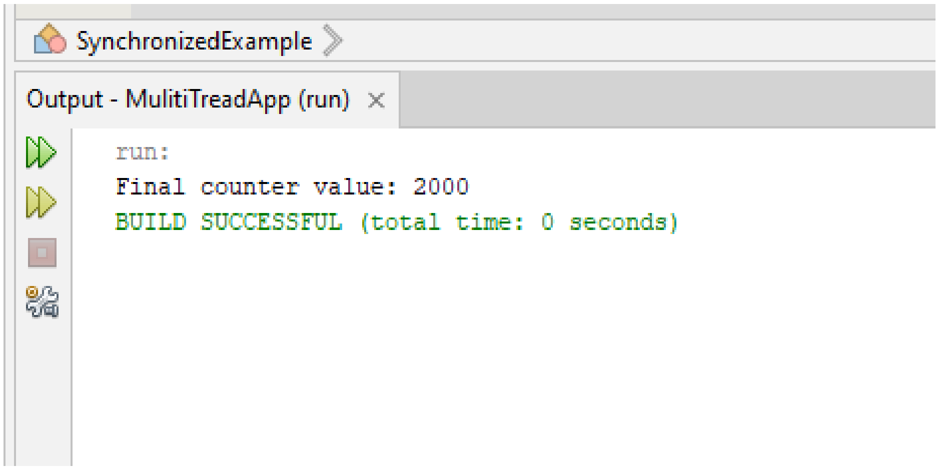
Synchronizing Shared Resources

1.Create a new class called Counter.java to demonstrate synchronization with shared resources.

```
    public class Counter {
     private int count = 0;
    // Synchronized method to ensure thread-safe access to the counter
    public synchronized void increment() {
    count++;
    }
    public int getCount() {
    return count;
    }
    }
```

```
public class SynchronizedExample extends Thread {
    private Counter counter;
    public SynchronizedExample(Counter counter) {
    this.counter = counter;
    }
    @Override
    public void run() {
    for (int i = 0; i < 1000; i++) {
    counter.increment();
    }
    }
    public static void main(String[] args) throws InterruptedException {
    Counter counter = new Counter();
    // Create and start multiple threads
    Thread thread1 = new SynchronizedExample(counter);
    Thread thread2 = new SynchronizedExample(counter);
    thread1.start();
    thread2.start();
    // Wait for threads to finish
    thread1.join();
    thread2.join();
    System.out.println("Final counter value: " + counter.getCount());
    }
```

Output:-



```
run:
Final counter value: 2000
BUILD SUCCESSFUL (total time: 0 seconds)
```

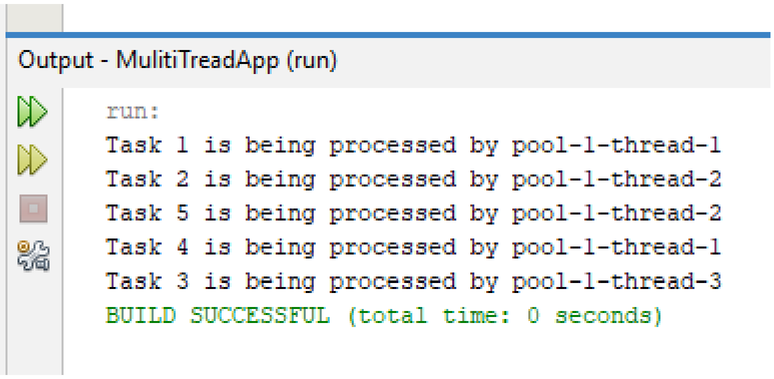
Part 4: Thread Pooling

Using ExecutorService for Thread Pooling

1.Create a new class called ThreadPoolExample.java.

```
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
    class Task implements Runnable {
private int taskId;
public Task(int taskId) {
this.taskId = taskId;
}
@Override
public void run () {
 System.out.println("Task " + taskId + " is being processed by " +
Thread.currentThread().getName());
 }
}
public class ThreadPoolExample {
    public static void main(String[] args) {
    // Create a thread pool with 3 threads
   ExecutorService executorService = Executors.newFixedThreadPool(3);
   // Submit tasks to the pool
  for (int i = 1; i <= 5; i++) {
  executorService.submit(new Task(i));
  }
  // Shutdown the thread pool
 executorService.shutdown();
 }
 }
```

Output:-



```
run:
Task 1 is being processed by pool-1-thread-1
Task 2 is being processed by pool-1-thread-2
Task 5 is being processed by pool-1-thread-2
Task 4 is being processed by pool-1-thread-1
Task 3 is being processed by pool-1-thread-3
BUILD SUCCESSFUL (total time: 0 seconds)
```

Part 5: Thread Lifecycle and States

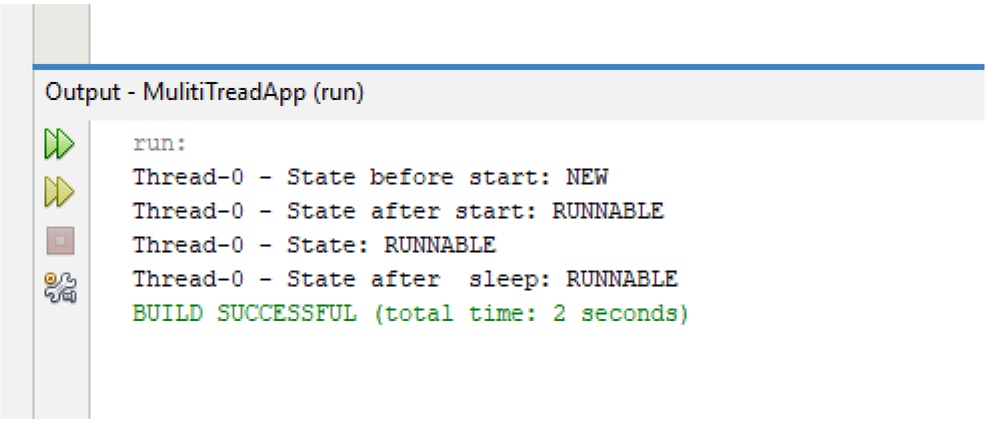
Thread Lifecycle Example

1.Create a new class called ThreadLifecycleExample.java.

```
public class ThreadLifecycleExample extends Thread {
    @Override
 public void run() {
 System.out.println(Thread.currentThread().getName() + " - State: " +
Thread.currentThread().getState());
 try {
 Thread.sleep(2000); // Simulate waiting state
 } catch (InterruptedException e) {
 e.printStackTrace();
 }
 System.out.println(Thread.currentThread().getName() + " - State after  sleep: " +
Thread.currentThread().getState());
 }
 public static void main(String[] args) {
 ThreadLifecycleExample thread = new ThreadLifecycleExample();
System.out.println(thread.getName() + " - State before start: " +  thread.getState());
 thread.start(); // Start the thread
 System.out.println(thread.getName() + " - State after start: " +  thread.getState());
 }


}
```

Output:-



Output - MulitiTreadApp (run)

```
run:
Thread-0 - State before start: NEW
Thread-0 - State after start: RUNNABLE
Thread-0 - State: RUNNABLE
Thread-0 - State after  sleep: RUNNABLE
BUILD SUCCESSFUL (total time: 2 seconds)
```