SESSION - 10
BASICS OF MULTITHREADING

1. Name of method in Thread class to pause execution?

- The sleep() method of Thread class is used to sleep a thread for the specified amount of time.
- Thread.sleep() method can be used to pause the execution of current thread for specified time in milliseconds.
- Thread.sleep() interacts with the thread scheduler to put the current thread in wait state for specified period of time. Once the wait time is over, thread state is changed to runnable state and wait for the CPU for further execution. So the actual time that current thread sleep depends on the thread scheduler that is part of operating system.

EXAMPLE:

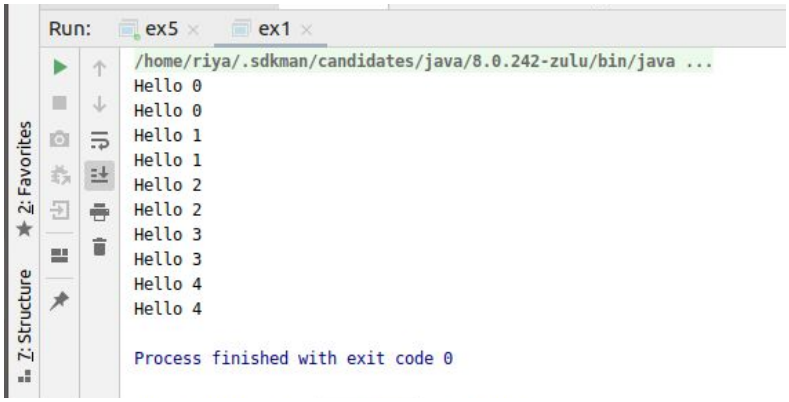
2. Role of "volatile" keyword.

- The contents of the particular device register could change at any time, so we need the volatile keyword to ensure that such accesses are not optimized away by the compiler.
- Volatile keywords are used to modify the value of a variable by different threads. It is also used to make classes thread safe. It means that multiple threads can use a method and instance of the classes at the same time without any problem.
- It guarantees that the value of the volatile variable will always be read from the main memory, not from the local thread cache.

3. Write a program to create a thread using Thread class and Runnable interface each.

>>>>>>>>>>>>>>>>>>>>>>>>>>>>Thread class>>>>>>>>>>>>>>>>>>>>>>>>>>>>

```
class Runners extends Thread
{
  public void run()
  {
     for(int i=0; i<5; i++)
     {
        System.out.println("Hello "+i);

        try
        {
           Thread.sleep(100);
        }
        catch (InterruptedException e)
        {
           e.printStackTrace();
        }
     }
  }
}
public class ex1
{
  public static void main(String[] args)
  {
     Runners runner1 = new Runners();
     runner1.start();

     Runners runner2 = new Runners();
     runner2.start();
  }
}
```



>>>>>>>>>>>>>>>>>>>>>>>>>>>>Runnable Interface>>>>>>>>>>>>>>>>>>>>>>>>>>>>

```
class Runner implements Runnable
{
  public void run()
  {
     for(int i=0; i<5; i++)
     {
        System.out.println("Hello "+i);

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

```
        }
        catch (InterruptedException e)
        {
            e.printStackTrace();
        }
    }
  }
}


public class ex2
{
  public static void main(String[] a)
  {
  Thread t1 = new Thread(new Runners());
  Thread t2 = new Thread(new Runners());

  t1.start();
  t2.start();
  }
}
```

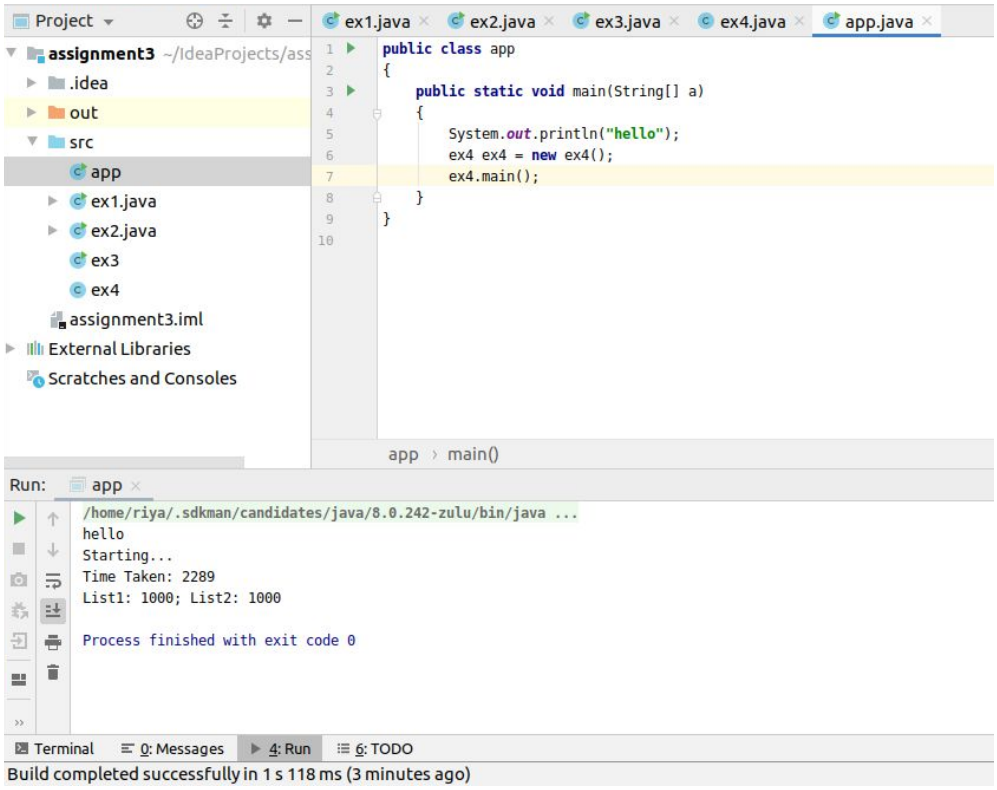
4. Write a program using synchronization block.

>>>>>>>>>>>>>>>>>ex4.java>>>>>>>>>>>>>>>>>>>>>>

```
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import java.util.ArrayList;

public class ex4
{
  private Random random = new Random();

  private Object lock1 = new Object();
  private Object lock2 = new Object();

  private List<Integer> list1 = new ArrayList<Integer>();
  private List<Integer> list2 = new ArrayList<Integer>();
```

```
  public void stageOne()
  {
     synchronized (lock1)
     {
     try
     {
        Thread.sleep(1);
     }
     catch (InterruptedException e)
     {
        e.printStackTrace();
     }
     list1.add(random.nextInt(100));
  }}
  public void stageTwo()
  {   synchronized (lock2)
  {
     try
     {
        Thread.sleep(1);
     }
     catch (InterruptedException e)
     {
        e.printStackTrace();
     }
     list2.add(random.nextInt(100));
  }}
  public void process()
  {
     for(int i=0;i<1000;i++)
     {
        stageOne();
        stageTwo();
     }
  }
  public void main()
  {
     System.out.println("Starting...");
     long start = System.currentTimeMillis();
     process();
     long end = System.currentTimeMillis();
     System.out.println("Time Taken: "+(end - start));
     System.out.println("List1: "+ list1.size()+ "; List2: "+list2.size());
  }
}
```

>>>>>>>>>>>>>>>>>app.java>>>>>>>>>>>>>>>>>>



5. Write a program using synchronization method

```
public class Table
{
  synchronized void printTable(int n)
{
    for(int i=1;i<=5;i++)
{
      System.out.println(n*i);
      Try
{
        Thread.sleep(400);
      }
catch(Exception e){System.out.println(e);}
    }

  }


static class MyThread1 extends Thread
{
  Table t;
  MyThread1(Table t){
    this.t=t;
  }
```

```
  public void run(){
    t.printTable(5);
  }

}
static class MyThread2 extends Thread{
  Table t;
  MyThread2(Table t){
    this.t=t;
  }
  public void run(){
    t.printTable(100);
  }
}

public static class TestSynchronization2{
  public static void main(String args[]){
    Table obj = new Table();
    MyThread1 t1=new MyThread1(obj);
    MyThread2 t2=new MyThread2(obj);
    t1.start();
    t2.start();
  }
}}
```

6. Write a program to create a Thread pool of 2 threads where one Thread will print even numbers and other will print odd numbers.

>>>>>>>>>>>>>>>>>>WorkerThread.java>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

```
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
class WorkerThread implements Runnable
{
  private String message;
  public WorkerThread(String s){
```

```
        this.message=s;
    }
    public void run() {

        System.out.println(Thread.currentThread().getName()+" (Start) message = "+ message);
        processmessage();//call processmessage method that sleeps the thread for 2 seconds
        System.out.println(Thread.currentThread().getName()+" (End)");//prints thread name
    }
    private void processmessage() {
        for(int i=1; i<11; i++) {
            if (i % 2 == 0) {
                System.out.println("Even number: " + i);
            } else {
                System.out.println("Odd Number: " + i);
            }

            try {
                Thread.sleep(2000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }}}
```

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>TestThreadPool.java>>

```
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
public class TestThreadPool
{

    public static void main(String[] args) {
        ExecutorService executor = Executors.newFixedThreadPool(5);//creating a pool of 5 threads
        for (int i = 0; i < 2; i++) {
            Runnable worker = new WorkerThread("" + i);
            executor.execute(worker);//calling execute method of ExecutorService
        }
        executor.shutdown();
        while (!executor.isTerminated()) {   }

        System.out.println("Finished all threads");
    }

}
```

7. Write a program to demonstrate wait and notify methods.

>>>>>>>>>>>>>>>>>>app.java>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

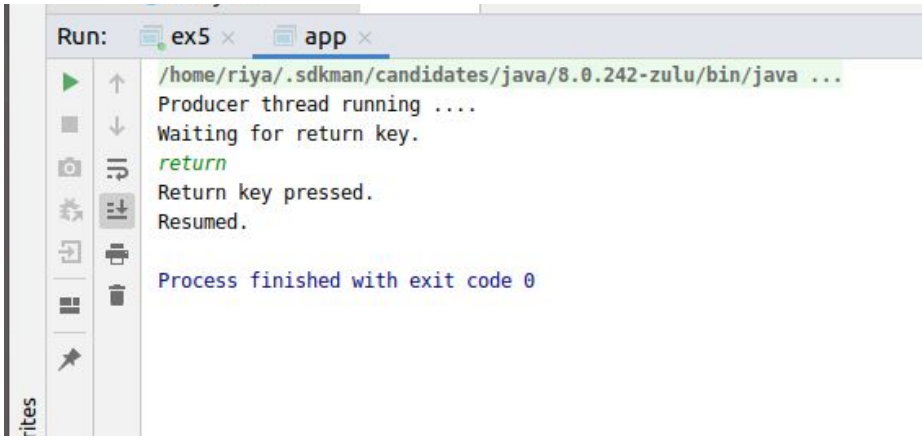
```
public class app
{
  public static void main(String[] args) throws InterruptedException {

    final Processor processor = new Processor();

    Thread t1 = new Thread(new Runnable() {

      @Override
      public void run() {
        try {
          processor.produce();
        } catch (InterruptedException e) {
          e.printStackTrace();
        }
      }
    });

    Thread t2 = new Thread(new Runnable() {
```

```
        @Override
        public void run() {
            try {
                processor.consume();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    });

    t1.start();
    t2.start();

    t1.join();
    t2.join();
  }
}
```

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>Processor.java>>>>>>>>>>>>>>>>>>>

```
import java.util.Scanner;
public class Processor
{
  public void produce() throws InterruptedException {
    synchronized (this) {
      System.out.println("Producer thread running ....");
      wait();
      System.out.println("Resumed.");
    }
  }

  public void consume() throws InterruptedException {

    Scanner scanner = new Scanner(System.in);
    Thread.sleep(2000);

    synchronized (this) {
      System.out.println("Waiting for return key.");
      scanner.nextLine();
      System.out.println("Return key pressed.");
      notify();
      Thread.sleep(5000);
    }
  }
}
```

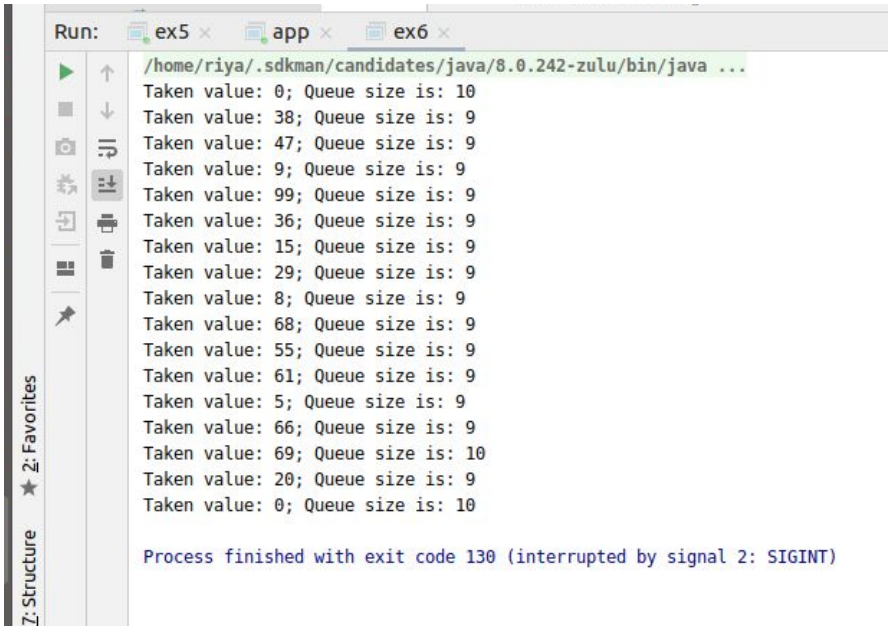
After any key pressed:



8. Write a program to demonstrate sleep and join methods.

```
import java.util.Random;
import java.util.concurrent.ArrayBlockingQueue;
import java.util.concurrent.BlockingQueue;
public class ex6
{
  private static BlockingQueue<Integer> queue = new ArrayBlockingQueue<Integer>(10);

      public static void main(String[] args) throws InterruptedException
      {
         Thread t1 = new Thread(new Runnable()
         {
            public void run()
            {
               try {
                  producer();
               } catch (InterruptedException e)
               {
                  e.printStackTrace();
               }
            }
         });

         Thread t2 = new Thread(new Runnable()
         {
            public void run() {
               try {
                  consumer();
               } catch (InterruptedException e)
               {

                  e.printStackTrace();
```

```
            }
        }
    });

    t1.start();
    t2.start();

    t1.join();
    t2.join();
}

private static void producer() throws InterruptedException
{
    Random random = new Random();

    while(true)
    {
        queue.put(random.nextInt(100));
    }
}

private static void consumer() throws InterruptedException
{
    Random random = new Random();

    while(true)
    {
        Thread.sleep(100);

        if(random.nextInt(10) == 0)
        {
            Integer value = queue.take();

            System.out.println("Taken value: " + value + "; Queue size is: " + queue.size());
        }
    }
}
}
```


Run:
ex5
app
ex6
/home/riya/.sdkman/candidates/java/8.0.242-zulu/bin/java ...
Taken value: 0; Queue size is: 10
Taken value: 38; Queue size is: 9
Taken value: 47; Queue size is: 9
Taken value: 9; Queue size is: 9
Taken value: 99; Queue size is: 9
Taken value: 36; Queue size is: 9
Taken value: 15; Queue size is: 9
Taken value: 29; Queue size is: 9
Taken value: 8; Queue size is: 9
Taken value: 68; Queue size is: 9
Taken value: 55; Queue size is: 9
Taken value: 61; Queue size is: 9
Taken value: 5; Queue size is: 9
Taken value: 66; Queue size is: 9
Taken value: 69; Queue size is: 10
Taken value: 20; Queue size is: 9
Taken value: 0; Queue size is: 10
Process finished with exit code 130 (interrupted by signal 2: SIGINT)
2: Favorites
7: Structure