Assignment- 07
22610011 - Anuja Suntnur (S-2)

1. Can we call the run() method instead of start()?
Ans: The `start()` method is used to begin the execution of a thread, while the `run()` method contains the code that constitutes the thread's processing task.

Typically, you don't directly call the `run()` method to start a new thread because calling `run()` directly will execute the code synchronously in the current thread, without creating a new thread.

When you call the `start()` method on a `Thread` object, it creates a new thread of execution and invokes the `run()` method on that thread. This separation allows for concurrent execution, with the `run()` method executing in its own thread.

So, while it's technically possible to call the `run()` method directly, doing so won't create a new thread; it will just execute the code synchronously in the current thread. If your intention is to create a new thread and execute code concurrently, you should use the `start()` method.

2. Explain the use of word Synchronized.

Ans: The `synchronized` keyword is used to control access to critical sections of code, ensuring that only one thread can execute the synchronized block at a time. This is important in concurrent programming to prevent race conditions and ensure data consistency when multiple threads are accessing shared resources.

Here's a breakdown of how `synchronized` is used:

1. Synchronized Methods: You can declare entire methods as synchronized by placing the `synchronized` keyword before the method signature. When a thread invokes a synchronized method, it acquires the intrinsic lock (also known as monitor lock) associated with the object the method belongs to. Other threads attempting to invoke synchronized methods on the same object will be blocked until the lock is released.

```
public synchronized void synchronizedMethod() {

        // Synchronized method code

}
```

2.Synchronized Blocks: Alternatively, you can use synchronized blocks to synchronize specific sections of code. This allows for finer-grained control over synchronization. To create a synchronized block, you specify an object reference within parentheses after the `synchronized` keyword. The synchronized block is then associated with the intrinsic lock of that object.

```
synchronized (object) {

        // Synchronized block code

}
```

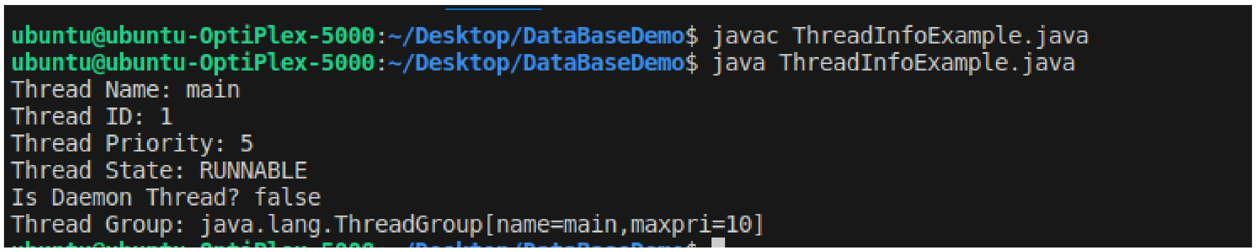
The use of `synchronized` ensures that only one thread can execute the synchronized code at a time, preventing concurrent access to shared resources and helping to maintain data integrity. However, excessive use of synchronization can lead to performance issues such as contention and deadlock, so it's important to use it judiciously and only where necessary.

3. Write a program to display thread information.

Ans:

```
public class ThreadInfoExample {
   public static void main(String[] args) {

       // Get a reference to the current thread

       Thread currentThread = Thread.currentThread();


       // Display information about the current thread

       System.out.println("Thread Name: " + currentThread.getName());

       System.out.println("Thread ID: " + currentThread.getId());

       System.out.println("Thread Priority: " +
currentThread.getPriority());

       System.out.println("Thread State: " + currentThread.getState());

       System.out.println("Is Daemon Thread? " +
currentThread.isDaemon());

       System.out.println("Thread Group: " +
currentThread.getThreadGroup());

   }

}
```

Output:



```
ubuntu@ubuntu-OptiPlex-5000:~/Desktop/DataBaseDemo$ javac ThreadInfoExample.java
ubuntu@ubuntu-OptiPlex-5000:~/Desktop/DataBaseDemo$ java ThreadInfoExample.java
Thread Name: main
Thread ID: 1
Thread Priority: 5
Thread State: RUNNABLE
Is Daemon Thread? false
Thread Group: java.lang.ThreadGroup[name=main,maxpri=10]
```

4. Create a thread using Thread class and Runnable class.

Ans:

1. **Using the `Thread` class:**

```
public class MyThread extends Thread {

   public void run() {

       System.out.println("Thread using Thread class is running");

   }


   public static void main(String[] args) {

       MyThread thread = new MyThread();

       thread.start(); // Start the thread

   }

}
```

2. **Using the `Runnable` interface:**

Ans:

```
public class MyRunnable implements Runnable {

   public void run() {

       System.out.println("Thread using Runnable interface is
running");

   }
```

```
    public static void main(String[] args) {

        MyRunnable myRunnable = new MyRunnable();

        Thread thread = new Thread(myRunnable); // Create a thread
passing the runnable object

        thread.start(); // Start the thread

    }

}
```

5. Write a program for thread communication and synchronization.

Ans:

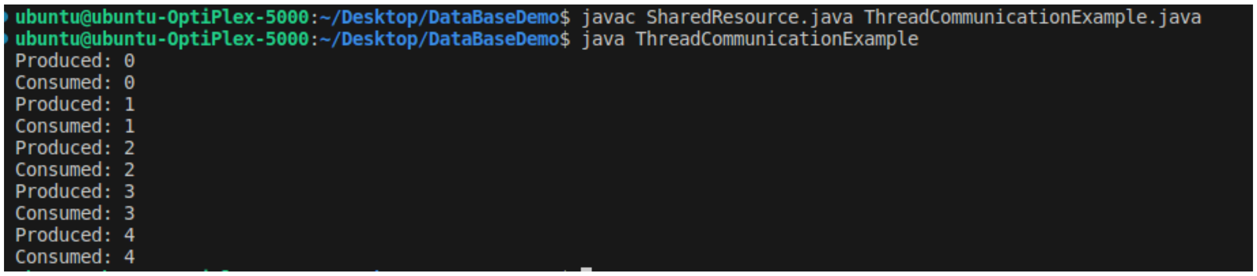
```
class SharedResource {

   private int data;

   private boolean available = false;


   public synchronized void produce(int newData) {

       while (available) {

           try {

               wait(); // Wait for consumer to consume

           } catch (InterruptedException e) {

               Thread.currentThread().interrupt();
```

```java
            }
        }
        data = newData;
        available = true;
        System.out.println("Produced: " + data);
        notify(); // Notify consumer that new data is available
    }

    public synchronized int consume() {
        while (!available) {
            try {
                wait(); // Wait for producer to produce
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
        available = false;
        System.out.println("Consumed: " + data);
        notify(); // Notify producer that data has been consumed
        return data;
    }
}
```

```java
class Producer extends Thread {
   private SharedResource sharedResource;


   public Producer(SharedResource sharedResource) {
       this.sharedResource = sharedResource;
   }


   public void run() {
       for (int i = 0; i < 5; i++) {
           sharedResource.produce(i);
           try {
               Thread.sleep((int) (Math.random() * 100));
           } catch (InterruptedException e) {
               Thread.currentThread().interrupt();
           }
       }
   }
}


class Consumer extends Thread {
   private SharedResource sharedResource;


   public Consumer(SharedResource sharedResource) {
       this.sharedResource = sharedResource;

   }



   public void run() {

       for (int i = 0; i < 5; i++) {

           int consumedData = sharedResource.consume();

           try {

               Thread.sleep((int) (Math.random() * 100));

           } catch (InterruptedException e) {

               Thread.currentThread().interrupt();

           }

       }

   }

}



public class ThreadCommunicationExample {

   public static void main(String[] args) {

       SharedResource sharedResource = new SharedResource();

       Producer producer = new Producer(sharedResource);

       Consumer consumer = new Consumer(sharedResource);

       producer.start();

       consumer.start();

   }
```

```
}
```

Output:



```
ubuntu@ubuntu-OptiPlex-5000:~/Desktop/DataBaseDemo$ javac SharedResource.java ThreadCommunicationExample.java
ubuntu@ubuntu-OptiPlex-5000:~/Desktop/DataBaseDemo$ java ThreadCommunicationExample
Produced: 0
Consumed: 0
Produced: 1
Consumed: 1
Produced: 2
Consumed: 2
Produced: 3
Consumed: 3
Produced: 4
Consumed: 4
```