Lec 32 Multi-Threading Part 3 :

Launch.java

```java
package com.mainapp;
public class Launch {

    public static void main(String[] args) throws Inte

        Database1 database1 = new Database1();
        Database2 database2 = new Database2();

        database1.start();  //finish
        database2.start();  //finish

//      database1.join();
//      database2.join();

        //main pause

        process(database1.collect,database2.collect);

    }

    private static void process(String[] collect1, Str
        for( String d: collect1)
            System.out.println(d);

        for( String d: collect2)
            System.out.println(d);
    }
}
```

Database1.java / Database2.java

```java
package com.mainapp;
public class Database2 extends Thread {

    public String collect[]=new String[4];

    @Override
    public void run() {

        String data[]= {"xd1","xd2","xd3","xd4"}
        int i=0;
        for( String d: data) {
            collect[i]=d;
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch blo
                e.printStackTrace();
            }
            System.out.println("DATA"+i+" COLLEC
            i++;
        }
    }
}
```

Console

```
<terminated> Launch (13) [Java Application] C:\P
null
null
null
null
null
null
null
null
DATA0 COLLECTED FROM DB1
DATA0 COLLECTED FROM DB2
DATA1 COLLECTED FROM DB1
DATA1 COLLECTED FROM DB2
DATA2 COLLECTED FROM DB2
DATA2 COLLECTED FROM DB1
DATA3 COLLECTED FROM DB2
DATA3 COLLECTED FROM DB1
```

Launch.java

```java
package com.mainapp;
public class Launch {

    public static void main(String[] args) throws Inte

        Database1 database1 = new Database1();
        Database2 database2 = new Database2();

        database1.start();  //finish
        database2.start();  //finish

        database1.join();
        database2.join();

        //main pause

        process(database1.collect,database2.collect);

    }

    private static void process(String[] collect1, Str
        for( String d: collect1)
            System.out.println(d);

        for( String d: collect2)
            System.out.println(d);
    }
}
```

Database1.java / Database2.java

```java
package com.mainapp;
public class Database1 extends Thread {

    public String collect[]=new String[4];

    @Override
    public void run() {

        String data[]= {"d1","d2","d3","d4"};
        int i=0;
        for( String d: data) {
            collect[i]=d;
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch blo
                e.printStackTrace();
            }
            System.out.println("DATA"+i+" COLLEC
            i++;
        }
    }
}
```

Console

```
<terminated> Launch (13) [Java Application] C:\P
DATA0 COLLECTED FROM DB1
DATA0 COLLECTED FROM DB2
DATA1 COLLECTED FROM DB2
DATA1 COLLECTED FROM DB1
DATA2 COLLECTED FROM DB2
DATA2 COLLECTED FROM DB1
DATA3 COLLECTED FROM DB2
DATA3 COLLECTED FROM DB1
d1
d2
d3
d4
xd1
xd2
xd3
xd4
```

- Above diagram explains the working of Threads
- Main has process method, first JVM will make main's thread and without join() method it will try to complete before even completing data collection, that's why it is printing null because it is running before collecting any data

- But using the join() method will ensure that first the DB1 and DB2 class's thread execution finishes and then only main thread runs. That's all → it's easy!!!

This lesson explains how Java handles multithreading using both the Thread class and Runnable interface, and explores important concepts like thread scheduling, context switching, thread sequencing, the join method, and thread priorities. Understanding these concepts helps programmers write efficient and reliable concurrent programs in Java.

Introduction to Multithreading in Java

Multithreading allows a program to execute multiple tasks simultaneously by creating separate threads. In Java, threads can be created by extending the Thread class or implementing the Runnable interface.

Limitation of Extending Thread Class

If a class extends the Thread class, it cannot extend any other class due to Java's single inheritance rule. This limits flexibility when designing classes.

Using Runnable Interface

Implementing the Runnable interface allows a class to be used as a thread without extending Thread, which means the class can still extend another class and implement multiple interfaces. Threads are created by passing Runnable objects to Thread constructors.

Context Switching and Lightweight Threads

Context switching in multithreading refers to switching between threads within the same application, which is lightweight because memory, resources, and ports remain the same. Switching between applications (multiprocessing) is heavier because it involves different memory and resources.

CPU Time Allocation and Thread Priorities

The operating system decides how much CPU time each thread gets, not the programmer. However, thread priority can influence scheduling, with higher-priority threads potentially receiving more CPU time depending on OS design.

Creating Threads with Runnable

To create threads with Runnable, objects of the class implementing Runnable are passed to Thread

constructors. The run() method must be overridden to define the thread's task.

Multiple Threads on One Class

Multiple threads can be created on the same class by making several Runnable objects and passing them to different Thread instances. Each thread can perform its own task independently.

Thread Sequencing and the Need for Order

Sometimes, threads need to run in a specific sequence, especially when one thread depends on the results of others (e.g., processing data only after it's collected from databases).

Example: Collecting and Processing Data with Threads

Two threads are used to collect data from two databases, and a third thread processes the data. If the processing thread runs before data collection is complete, it may process incomplete (null) data.

The Join Method for Thread Synchronization

The join() method pauses the current thread until the specified thread finishes its execution. This ensures that dependent tasks (like processing data) only happen after prerequisite threads complete.

Thread Priority and Scheduling

Threads have priorities (1 to 10, default is 5). Higher priority threads are more likely to be scheduled first, but this is not guaranteed and depends on the OS. Thread priorities can be set using constants like Thread.MAX_PRIORITY.

Practice and Understanding Multithreading

Practicing multithreading concepts—such as creating threads, using Runnable, applying join, and managing priorities—is crucial to mastering safe and efficient concurrent programming in Java.