**Title:- Thread Detection**

1.Thread Occurrence.

**A Thread is what?**

A process's path of execution is called a thread. A process can have many threads.

**Why Multithreading?**

Another name for a thread is a lightweight process. By breaking a process up into several threads, parallelism is intended to be achieved. For instance, many tabs in a browser could represent various threads. Multiple threads are used by MS Word, including one for processing inputs and another for formatting the text. Below is a discussion of further advantages of multithreading.

Threat detection is the process of scrutinising a security ecosystem from top to bottom to find any malicious behaviour that could jeopardise the network. If a threat is identified, mitigating measures must be taken to effectively neutralise it before it can take advantage of any existing vulnerabilities.

Being penetrated is a nightmarish scenario, so most firms that value their information will employ educated people and cutting-edge technology to create a protective barrier against potential troublemakers. Security, however, is a process and not a given.

The term "threat detection" has several different meanings when used in the context of a security programme for an organisation. Even the most effective security measures need to prepare for worst-case scenarios, where someone or something manages to bypass its defence and prevention mechanisms.

**How to control Main thread**

When our software is launched, the main thread is automatically established. We need to get a reference to it in order to govern it. You can accomplish this by using the Thread class's currentThread() function. A reference to the thread on which it was called is returned by this method. The priority of the Main thread is set to 5 by default, and all other user threads will inherit their parent's priority.

**Describe the methodology used to investigate thread:**

A thread is a small process that may run numerous pieces of code concurrently. The thread, however, differs from a process. Each process will have its own memory space allocated in the OS. The same holds true for threads, as they have distinct memory. The same RAM that the process has been allotted will be used to run all of the threads.

The Thread class includes a number of constructors for generating task-specific threads as well as thread management methods. It is a predefined class that is listed in the default package for Java.lang.

A distinct object of the Thread class creates and manages every thread in Java. A JVM thread is managed by an object of thread.

The thread class in a programme contains a number of methods that can be used to initiate, manage, stop, and perform a variety of other thread-related operations.

**Java Thread Benefits**

1.Java Threads are more lightweight than processes; they can be created faster and with fewer resources.

2. Threads share the data and code of their parent processes.

3. Switching context between threads is typically less expensive than switching context between processes.

4. Process communication is more difficult than thread intercommunication.

It implements the Runnable interface and extends the Object class. The thread class is declared as follows:

**A Thread can be created in Java in the following ways:**

1. By Extending Thread class
2. Implementing Runnable interface

**Extending Thread class:**

public class PlaySong extends Thread {

public void run() {

for(int i=0;i<100;i++) {

System.out.println("Song Playing ...... ");

}

}

public static void main(String Args[])

{

PlaySong p=new PlaySong ();

p.start();

for(int i=0;i<1000;i++) {

System.out.println("coding");

}

}

}

**Implementing Runnable interface:**

public class DemoThread implements Runnable{

public void run() {

for(int i=0;i<100;i++) {

System.out.println("hii thread1 started");

}

}

public static void main(String[] args) {

DemoThread d=new DemoThread();

Thread t1=new Thread(d);

t1.start();

Thread t2=new Thread(d);

t2.start();

}

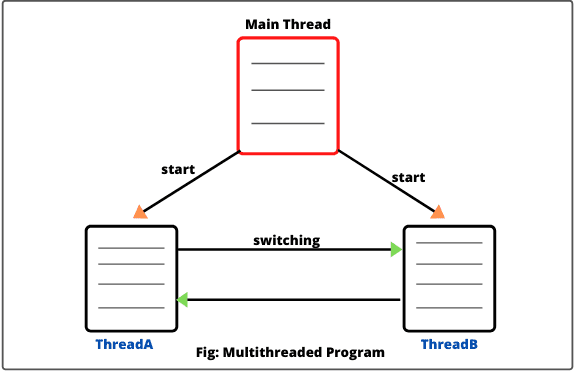
}

**Multithreading**

Multithreading is the simultaneous execution of several threads. Java refers to the process of running many threads concurrently as multithreading.

To put it another way, multithreading is a programming approach or idea in which a programme (process) is split up into two or more smaller programmes (subprocesses), each of which can carry out many tasks at once (at the same time and in parallel manner). Java refers to each program's subprogram as a thread.

Take a look at the figure below, which shows a Java application with three threads—one main and two others. The primary thread is used to create and launch the additional two threads, ThreadA and ThreadB. A programme is said to as multithreaded when it has numerous control flows.



Threads ThreadA and ThreadB run concurrently and share the resources after being started by the main thread. It is analogous to having a large family where everyone shares certain resources.

When a program has several threads, the CPU can switch between two threads so that they can run simultaneously. Context switching occurs when two threads are switched between.

The consumers perceive that all threads are running simultaneously because of how quickly one thread is switched to another. However, only one thread actually runs at a time.

Applications that need the simultaneous completion of numerous activities can benefit from this strategy. Time-sharing is the practise of several threads sharing CPU time on a single processor system.

Resource allocation and scheduling are handled by the operating system. Thus, multithreading enhances CPU efficiency by maximizing CPU utilization and minimizing CPU idle time.

Each thread in a multi-threaded software is given a single task to complete and runs independently. During execution, if an exception occurs in one thread, it has no impact on the other threads.

One thread might read data, another would process it, and a third might write it, improving an application's overall performance.

A multithreaded software can download a file from the network, play music, and display animations all at once.

**Advantage of Multithreading in Java**

The following are some benefits of employing the multithreading programming concept:

1. Distinct threads in a multithreaded application programme execute different portions of the application. Even if an exception arises in one of the threads, the entire application keeps running. During application execution, it has no impact on other threads.

2. Different threads are assigned to various processors, and each thread is processed concurrently across all of the processors.

3. Multithreading facilitates speeding up computation.

4. The multithreading strategy boosts the application's performance.

5. The memory address space used by threads is the same. Thus, it conserves memory.

6. A multi-threaded software uses the CPU to the fullest extent possible while minimising CPU idle time.

7. Transitioning from one thread to another's context

**Disadvantage of Multithreading in Java**

1.The code gets harder to maintain and debug as the number of threads increases.

2.The process of creating threads uses up memory and CPU resources on the machine.

3.The worker method must contain exception handling since unhandled exceptions can cause the programme to crash.

**The major purpose of threads is multitasking, or performing several activities at once.**

Eg:

Many requests arrive at once to the server. The processing of requests one by one is not possible ( as some may have problem with their connection , pc etc..). One request is handled at the time of a delay in order to increase time efficiency since if it is done one at a time and the first one is delayed, a lot of time would be damaged.

Threads are created with fewer resources than other process elements and share process resources when they are present.

One thread, known as the Main Thread, is present in every Java application.

Even yet, there are a tonne of additional Java threads working in the background, such as those managing the system, signal processing, memory, etc. The initial Java thread, however, is main, and we can construct more threads from it from the perspective of the application.

A programme running multiple threads at once is referred to as multithreading. Time slicing is an OS feature that divides processor time across various programmes and threads because a single core computer processor can only run one thread at a time.

**Java Thread Benefits**

1.Java Threads are more lightweight than processes; they can be created faster and with fewer resources.

2. Threads share the data and code of their parent processes.

3. Switching context between threads is typically less expensive than switching context between processes.

4. Process communication is more difficult than thread intercommunication.

**What is Daemon Thread in Java ?**

In Java, we create threads to perform multitasking, such as one thread processing (perhaps listening to UI actions) and another thread performing background operations like obtaining data from a database.

Let's say we create a game that includes background music. A different thread will be used to play the music. Now, if the game ends, or the main thread's execution is complete, we want the music to end as well. We can designate the music thread as a daemon thread so that the JRE won't have to wait for it to complete execution

Secondary threads include daemon threads. In other words, JRE doesn't wait for a daemon thread to finish. When all non-daemon threads have completed running, JRE will end the programme without worrying about the daemon thread completing.

All threads are non-daemons by default, but we can set any thread to be a daemon before launching it.

Making a thread directly involves creating an instance of the Thread class and calling start() on it. There are two ways to utilise the Thread class, but both ways under the hood create a thread the same way. You can derive you own class from Thread and override the run() method, which is called when the thread is started. Or, you can create an instance of a class that implement the Runnable interface, which has a run() method, and pass that Runnable to the Thread to its constructor, when you create the instance.

However, there is a different approach that makes use of the concurrency utility classes that the Java libraries offer, such as ThreadPool and similar stuff. But within, they're utilizing regular Thread.

# Main thread in Java

Java has multithreaded programming capability built in. A multi-threaded software has two or more components that can work together simultaneously. In such a programme, each component is referred to as a thread, and each thread specifies a unique path of execution.

A Java application starts with one thread operating right away. Because it is the first thread to operate when our programme starts, this is frequently referred to as the main thread of our program.

The following are some of the attributes connected to the main thread:

It is the thread from which more threads will be spawned as "child."

Because it does a variety of shutdown actions, it frequently has to be the last thread to complete execution.

**How to make a class thread-safe in java?**

It's important to keep the following in mind when writing thread-safe Java code since it will help you avoid major concurrency problems like race conditions and deadlocks:

1. Because they cannot have their state changed after being created, immutable objects are thread-safe by default. String is thread-safe by default in Java since it is immutable.
2. Java variables that are read-only or final are thread-safe in Java.
3. In Java, locking is one method of achieving thread safety.
4. Static variables can pose serious thread-safety problems if they are not correctly synchronised.
5. Java thread-safe class examples include String, Vector, Hashtable, and ConcurrentHashMap.
6. Because each thread has its own copy, local variables are likewise thread-safe. Using local variables is a useful method to build Java code that is thread-safe.
7. Atomic operations in Java are thread-safe like reading a 32-bit int from memory because it's an atomic operation it can't interleave with other threads.
8. Reduce the amount of objects that are shared between different threads to prevent thread-safety problems.
9. The Java volatile keyword can also be used to tell a thread not to cache variables or read from main memory. It can also tell the JVM to stop rearranging or streamlining code from a threading perspective.