**Core Java - Assignment**

**Module - 1**

1. **Multi-threading :**

**Theory : -**

1. Introduction to Threads :

-> Threads in Java allow concurrent execution of two or more parts of a program to maximize the utilization of CPU. Each part of such a program is called a thread. Threads are lightweight processes and are a fundamental part of Java's concurrency model.

-> Creating Threads

-> There are two main ways to create a thread in Java:

1. Extending the Thread class

2. Implementing the Runnable interface

-> Extending the Thread Class

To create a thread by extending the Thread class, you need to create a subclass of Thread and override its run method.

-> Implementing the Runnable Interface

To create a thread by implementing the Runnable interface, you need to implement the run method and pass an instance of the class to a Thread object.

1. Creating Threads by Extending Thread Class or Implementing Runnable Interface

-> In Java, you can create threads by either extending the Thread class or implementing the Runnable interface. Both approaches have their own use cases and advantages. Here's a detailed explanation with examples for each method.

-> Creating Threads by Extending the Thread Class

To create a thread by extending the Thread class, you need to create a subclass of Thread and override its run method. The run method contains the code that constitutes the new thread's task.

-> Creating Threads by Implementing the Runnable Interface

To create a thread by implementing the Runnable interface, you need to implement the run method and pass an instance of the class to a Thread object. This approach is more flexible because your class can extend another class while still implementing Runnable .

1. Thread Life Cycle

-> A thread can be in one of several states during its lifecycle. Understanding these states is crucial for effective thread management and debugging. The thread lifecycle includes the following states:

1. New. A thread that has been created but not yet started.

2. Runnable: A thread that is ready to run and is waiting for CPU time.

3. Blocked: A thread that is blocked and waiting for a monitor lock.

4. Waiting: A thread that is waiting indefinitely for another thread to perform a particular action.

5. Timed Waiting: A thread that is waiting for another thread to perform an action for up to a specified waiting time.

6. Terminated: A thread that has exited.

1. New

-> A thread is in the New state when it is created but not yet started. At this point, the thread is not yet eligible for execution.

Ex. = Thread thread = new Thread(new MyRunnable());

1. Runnable

-> A thread is in the Runnable state when it is ready to run and waiting for CPU time. This state includes both threads that are actually running and those that are waiting to be picked by the thread scheduler.

Ex. = thread.start();

1. Blocked

-> A thread is in the Blocked state when it is waiting for a monitor lock to enter a synchronized block or method. This happens when another thread holds the lock.

Ex. = synchronized (someObject) {

// Thread enters Blocked state if someObject is locked by another thread

}

4. Waiting

-> A thread is in the Waiting state when it is waiting indefinitely for another thread to perform a particular action. This state is entered by calling methods like Object .wait() or Thread. join() without a timeout.

Ex. = synchronized (someObject) {

someObject.wait(); // Thread enters Waiting state

}

5. Timed Waiting

-> A thread is in the Timed Waiting state when it is waiting for another thread to perform an action for up to a specified waiting time. This state is entered by calling methods like Thread. sleep(long millis) , Object.wait(long timeout) , or Thread. join(long millis) .

Ex. = Thread.sleep(1000); // Thread enters Timed Waiting state for 1 second

1. Terminated

-> A thread is in the Terminated state when it has completed its execution or has been explicitly terminated. Once a thread reaches this state, it cannot be restarted.

Ex. = public void run() {

// Thread execution logic

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

// Thread reaches Terminated state after completing run method

}

1. Synchronization and Inter-thread Communication :

-> Synchronization : -

- > Synchronization is used to control the access of multiple threads to shared resources. It ensures that only one thread can access a synchronized block or method at a time, preventing race conditions and ensuring data consistency.

-> Inter-thread Communication

Inter-thread communication in Java is achieved using methods like wait() , notify() , and notifyA11() . These methods are used to coordinate the activities of multiple threads.

-> wait() : Causes the current thread to wait until another thread invokes notify() or notifyA11() on the same object.

-> notify() : Wakes up a single thread that is waiting on the object's monitor.

-> notifyAll() : Wakes up all threads that are waiting on the object's monitor.