Multithreading in Java

1.Multithreading

DEFINITION/WHAT

Multithreading is a model of program execution that allows for multiple threads to be created within a process, executing independently but sharing process resources at the same time.

NEED/WHY

- ✓ *Processing of large data* where you can divide it in chunks. Threads can be assigned to complete the task in parallel.

IMPLEMENTATION/HOW IT WORKS

Thread Creation:

Declaring a class, which implements **Runnable Interface or extend Thread class**

REAL TIME EXAMPLE

For Example in a web browser, we can have one thread which handles the user interface, and in parallel, we can have another thread, which fetches the data to be displayed. So multithreading improves the responsiveness of a system.

ADDITIONAL INFORMATION

Java Multithreading is mostly used in games, animation, etc.

2.Runnable

DEFINITION/WHAT

Runnable is an interface that is to be implemented by a class whose instances are intended to be executed by a thread.

NEED/WHY

- ✓ Need 1:Runnable interface is always preferred because, the class implementing it can implement as many interfaces as a developer can, and also extend another class.

IMPLEMENTATION/HOW IT WORKS

Declaring a class which implements Runnable Interface

REAL TIME EXAMPLE

Can be used in MultiThreaded Environments

ADDITIONAL INFORMATION

This interface defines a single method of no arguments called run() which holds the code that needs to be executed by the thread. Thus the classes implement the Runnable interface needs to override the run().

3.Thread

DEFINITION/WHAT

Thread class is used to create and run threads for utilizing Multithreading feature of Java. It provides constructors and methods to support multithreading.

NEED/WHY

- ✓ Need 1: Concurrency.

IMPLEMENTATION/HOW IT WORKS

Declaring a class which extends Thread class and override run method

REAL TIME EXAMPLE

Can be used in MultiThreaded Environments

ADDITIONAL INFORMATION

Every thread has a priority. Threads with higher priority are executed in preference to threads with lower priority.

Main thread has priority 5.Priority can be set for the thread using setPriority() also we can set name for thread using setName() method

4.Thread LifeCycle

DEFINITION/WHAT

Thread life cycle denotes the various phases of a thread. Following are its states

1. ***NEW*** **–** a newly created thread that has not yet started the execution
2. ***RUNNABLE*** **–** either running or ready for execution but it's waiting for resource allocation
3. ***BLOCKED*** **–** waiting to acquire a monitor lock to enter or re-enter a synchronized block/method
4. ***WAITING*** **–** waiting for some other thread to perform a particular action without any time limit
5. ***TIMED_WAITING*** **–** waiting for some other thread to perform a specific action for a specified period
6. ***TERMINATED*** **–** has completed its execution

ADDITIONAL INFORMATION

Some methods of Thread class

Method	Description
setName()	to give thread a name
getName()	return thread's name
getPriority()	return thread's priority
isAlive()	checks if thread is still running or not
join()	Wait for a thread to end

run()	Entry point for a thread
sleep()	suspend thread for a specified time
start()	start a thread by calling run() method

5. Synchronization

DEFINITION/WHAT

Synchronization is a process of handling resource accessibility by multiple thread requests. The main purpose of synchronization is to avoid thread interference.

At times when more than one thread try to access a shared resource, we need to ensure that resource will be used by only one thread at a time. The process by which this is achieved is called synchronization.

NEED/WHY

- ✓ Need 1: Overcome distortion results when two or more threads access shared resource.
- ✓ *To overcome thread interference*

IMPLEMENTATION/HOW IT WORKS

Using keyword synchronized

REAL TIME EXAMPLE

Two persons accessing joint account at the same time

ADDITIONAL INFORMATION

Synchronization is built around an internal entity known as the lock or monitor.

Every object has a lock associated with it. By convention, a thread that needs consistent access to an object's fields has to acquire the object's lock before accessing them, and then release the lock when it's done with them.