1. **What is multitasking and what are its types?**

Ans:-

Multitasking means performing more than one task simultaneously. There are two types of multitasking:

a) process-based multi-tasking

b) thread-based multi-tasking

1. **Explain process-based and thread-based multitasking.**

Ans:

process - is a program in execution.

thread - is one of the part of program in execution.

process-based multitasking:- more than one processes are running simultaneously. e.g. word and excel applications are running simultaneously.

thread-based multitasking :- more than one threads are running simultaneously. e.g. within a word application, you can start formatting as well as printing.

1. **What is the difference between thread-based and process-based multitasking?**

Ans:

a) threads can share the memory , processes cannot.

b) context switching bet'n threads is relatively cheaper as compare to that bet'n processes.

c) cost of communication between threads is also low as compare to that bet'n processes.

( cheaper or cost is low actually means less no. of system resources are used.)

So that means thread based multitasking is light weight as compare to process based.

1. **Explain the difference between pre-emptive and time slicing schedulers.**

Ans:

**Preemptive vs. Time-Slicing Scheduling**

| **Feature** | **Preemptive Scheduling** | **Time-Slicing Scheduling** |
| --- | --- | --- |
| Execution Order | Higher-priority threads execute first | Threads execute in cyclic order |
| CPU Allocation | Depends on thread priority | Each thread gets a fixed time slice |
| Context Switching | Happens when a higher-priority thread arrives | Happens when the time slice expires |
| Fairness | Not fair, as low-priority threads may starve | Fair, as all threads get CPU time |
|  |  |  |

1. **What are the steps required to create multithreading application?**

Ans:

Following are the imp. steps required for multithreading application.

a) create thread/s

b) define thread execution body

c) register thread with the thread schedular

d) thread schedular will execute the thread/s

1. **What is the difference between start and run methods of Thread class?**

Ans:- start method allows you to register the thread with thread scheduler where as run() method allows you to write thread execution body i.e. the code which needs to be executed by a particular thread or threads. Start() method is called by a programmer whereas run() method is called internally by a thread scheduler.

1. **Can we call run() method directly?**

Ans: yes , but that won’t be thread execution i.e. we don’t get different call stacks.

1. **What are the two ways to create a thread in Java? Which one is better way?**

Ans:

There are two ways to create multithreading in java

1. extends Thread
2. implements Runnable

In Java, when creating a new thread, you have two primary options:

1. **Extending the Thread class**
2. **Implementing the Runnable interface**

**Why implements Runnable is generally better than extends Thread:**

**1. Java supports single inheritance**

* Java does not support multiple inheritance, meaning a class can only extend **one** other class.
* If you extend Thread, your class **cannot** extend any other class, which limits its flexibility.
* However, by implementing Runnable, your class can still extend another class and gain additional functionalities.

**Example** (Problem with extends Thread):

class MyThread extends Thread {

// Custom thread implementation

}

class MyOtherClass extends MyThread { // This is fine

}

Now, if MyThread was already extending another class:

class SomeBaseClass {

// Some functionality

}

class MyThread extends SomeBaseClass { // Now it can't extend Thread

}

class MyOtherClass extends MyThread { // Cannot extend Thread anymore

}

**Solution: Use Runnable**

class SomeBaseClass {

// Some functionality

}

class MyRunnable implements Runnable {

public void run() {

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

}

}

class MyOtherClass extends SomeBaseClass { // Can still extend another class

}

**2. Better separation of concerns (Encapsulation & Code Reusability)**

* Using Runnable keeps thread behavior separate from the actual Thread class, leading to better code organization.
* It allows **different threads to execute the same Runnable instance**, which is not possible with Thread.

**Example (Using extends Thread)**:

class MyThread extends Thread {

public void run() {

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

}

}

public class Main {

public static void main(String[] args) {

MyThread t1 = new MyThread();

MyThread t2 = new MyThread();

t1.start();

t2.start();

}

}

Here, t1 and t2 are **two separate objects**, meaning they do not share any state.

**Example (Using implements Runnable)**:

class MyRunnable implements Runnable {

public void run() {

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

}

}

public class Main {

public static void main(String[] args) {

MyRunnable r = new MyRunnable();

Thread t1 = new Thread(r);

Thread t2 = new Thread(r);

t1.start();

t2.start();

}

}

Here, both t1 and t2 **share the same instance of MyRunnable**, meaning they can access shared resources efficiently.

**3. Thread Pooling (Recommended for large-scale applications)**

* Using Runnable, you can submit tasks to a thread pool (ExecutorService), which is **not possible with Thread**.
* Thread pools **reuse** threads, making them more efficient than creating new Thread objects each time.

**Example (Using ExecutorService)**:

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

class MyRunnable implements Runnable {

public void run() {

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

}

}

public class Main {

public static void main(String[] args) {

ExecutorService executor = Executors.newFixedThreadPool(5); // Thread pool of 5 threads

for (int i = 0; i < 10; i++) {

executor.execute(new MyRunnable());

}

executor.shutdown();

}

}

* This **improves performance** because it **reuses** threads instead of creating and destroying them repeatedly.

**4. More Flexible Object-Oriented Design**

* When extending Thread, each instance is a **separate thread**. This is **not reusable**.
* With Runnable, you can pass the same Runnable instance to **multiple Thread objects**, making it more reusable and manageable.

**5. Avoids Unnecessary Overhead**

* When you extend Thread, your class **inherits all properties and methods** from Thread, which may not always be needed.
* This increases memory overhead since it loads unnecessary members.
* With Runnable, your class only implements run(), keeping it **lightweight**.

**When to use extends Thread?**

Though implements Runnable is generally preferred, extends Thread can be useful when:

1. You **need to override** other Thread class methods such as start(), interrupt(), etc.
2. You are working on **very small, simple applications** where thread reusability and flexibility are not a concern.
3. **What is synchronization ?**

Ans:

Synchronization is used to prevent thread interference by ensuring that only one thread can access a critical section at a time.

1. **What is the difference between synchronized keyword applied to a method and synchronized block inside a method?**

Ans:

**Comparison: Synchronized Method vs Synchronized Block**

| **Feature** | **synchronized Method** | **synchronized Block** |
| --- | --- | --- |
| **Scope** | Locks the **entire method** | Locks only a specific block of code |
| **Performance** | Less efficient (broader locking) | More efficient (fine-grained locking) |
| **Flexibility** | Cannot control which object is locked | Can choose which object to lock |
|  |  |  |
| **Usage** | Used when entire method needs synchronization | Used when only part of the method needs synchronization |

1. **What are the steps need to be performed in case of “implements Runnable” way of creating multithreading application?**

Ans:

a) define a class which implements Runnable

b) define run()

c) instantiate the class which impl. Runnable

d) instantiate Thread class by passing above instance (child of Runnable)

e) register Thread class instance/s

1. **What is the meaning of race condition? Explain it with the example.**

Ans:

when threads share the memory there is a risk of "race condition".

e.g.

There are 2 threads.

one thread is reading from a file

other thread is writing to a file.

Race condition means

while one thread is reading from a file, other thread might write in a file or vice-versa.

Race condition always leads to Data Corruption.

1. **How do we avoid Race condition ?**

Ans:

We will have to make sure that while one thread is working on a data, other thread should not run. Only after first thread completes its job, other thread should start its execution. In java we can achieve this by using "synchronization".

1. **What is the meaning of thread-safe class?**

Ans:

A **thread-safe class** in Java is a class that **functions correctly** when accessed **by multiple threads simultaneously**. This means that the class ensures **no race conditions, data inconsistencies, or unexpected behavior** occur, even when multiple threads try to read and write its data concurrently.

This can be achieved by either writing non-static synchronized methods or blocks inside the class.

1. **What exactly happens when we use synchronization?**

Ans:

There is a concept of object lock.In java every object has a lock. This lock can be accessed by only one thread at a time. The lock gets released as soon as the thread completes as soon as thread completes the synchronized method or block and then another thread which is in a seeking lock state can acquire the lock. However it is important to note that the same thread which has realeased the lock may reacquire again. It all depends on the underlying thread scheduler.

This lock comes into picture only when object has got non-static synchronized method/s or block. Whichever thread executes the synchronized method first, it acquires the lock. Other thread/s have to be in "seeking lock state".

Acquiring and releasing the lock happens automatically.

Once a thread acquires a lock on an object, it can have control on all the non-static synchronized methods of that object.

1. **When we write synchronized block, what is the argument we need to write inside it?**

Ans:

Inside synchronized block we must write the argument as reference to the object on which threads would like to acquire the lock.

1. **What is the need to have thread communication in case of synchronization?**

Ans: Even though synchronized method or block is used to avoid "Race Condition", there can be danger of "DeadLock" inside it.

e.g. if one thread is working inside synchronized block or method and if it gets stuck up ! imagine what will happen ?

neither this thread can complete and release the lock, nor other thread can acquire the lock.

Solution to this is to have a Communication between threads.

i.e. if the thread realizes it cannot continue, it should come out of synchronized method or block and release the lock. Now other thread will acquire the lock, execute the code and allow the first thread to resume.

**18) What are wait ,notify and notifyAll methods?**

Ans:-

wait, notify and notifyAll methods belong to java.lang.Object class and are used to have a communication between threads in case of synchrnozation.

a) wait

it will make thread release the lock and go to wait pool.

b) notify

it will make the thread to move from wait pool to seeking lock state.

c) notifyAll

it will make all the threads to move from wait pool to seeking lock state.

1. **Is there any rule about from where wait ,notify and notifyAll methods should be called ? or they can be called from any method?**

Ans:

No they cannot be called from any method. These methods must be called only from synchronized method or block.

1. **Why wait , notify and notifyAll methods can be called only from synchronized method or block?**

Ans:

It’s because wait(),notify() and notifyAll() methods are used to control the lock ( release and acquire the lock) and the concept of lock comes into picture only in case of synchronized method or block.

1. **What is the main difference between wait and sleep methods?**

Ans:

wait method releases the lock , sleep does not.

1. **What is class lock ?**

Ans:

every class has a lock. It is actually a lock on an instance of class Class. This is because , whenever any class is loaded in java, it is represented by instance of class Class.

The class lock comes into picture in case of synchronized static methods.

Thread which gives a call to synchronized static method can acquire a class lock. Only after thread complete that static method, lock is released.

1. **What is the use of “join()” method?**

Ans:

join method makes caller thread (main thread) to wait for called thread/threads to complete.

1. **What is InterruptedException?**

Ans:

Whenever thread is in a blocked state ie. due to sleep, join or wait methods, it can get interrupted by other threads. Whenever blocked thread gets interrupted, it throws "InterruptedException".

But this cannot be predictable, hence java enforces you to either handle or declare InterruptedException whenever you invoke the above methods.

1. **What is the lifecycle of a thread?**

Ans:

Born

When we create an instance of a class which extends Thread or explicitly instantiate Thread class.

Runnable

When we invoke “start()” method.

Running

When scheduler invokes “run()” method.

Blocked

When either sleep or wait or join is invoked.

Dead

Either run method gets over or an unhandled exception occurs when thread is executing or it’s in a blocked state.

1. **What is main thread?**

Ans:

Main thread is the thread which is executed by JVM. The job of main thread is to execute main method.

1. **What is Daemon thread in java ? what is its example?**

Ans:

A daemon thread in Java is a low-priority background thread that runs in the JVM (Java Virtual Machine) to provide services to user threads. It is automatically terminated when all non-daemon (user) threads finish execution.

The garbage collection thread is indeed an example of a daemon thread. It operates in the background, helping to manage memory without interfering with the execution of user threads, and it automatically terminates when there are no active user threads left.

1. **When you use class lock in java , inside the synchronized block which reference is needed?**

Ans:

Reference of class Class on which threads would like to acquire the lock.

1. **Explain the concept of thread pool.**

Ans:

Thread pool is the one which contains already created threads in the idle state. Whenever requirement comes any randomly selected thread from the thread pool will be made active and that thread is supposed to perform the given task. Once task gets completed the thread is not killed rather it is sent back to the thread pool in order to serve any new requirement. This approach ( instead of creating new thread from the scratch when requirement comes , use the existing thread from the thread pool) improves the performance especially on the server side applications.

1. **What is the API used in java in order to implement thread pool?**

Ans:

ExecutorService interface and Executors class from java.util.concurrent package.

1. **What is the use of “execute()” method in the thread pool concept?**

Ans:

execute() method is used to submit the task to the thread pool.

1. **What is the use of “shutdown()” method in thread pool concept?**

Ans:

shutdown() method is used to indicate JVM that we are done with submitting the task/s to the thread pool.

1. **What is the use of ReentrantLock class in multithreading?**

Ans:

ReentrantLock class is used to apply explicit lock ( with lock() and unlock() ) on an object.

It has two important features:

* 1. Lock fairness:- lock can be acquired by the longest waiting thread
  2. tryLock() with or without time is used to ensure that thread won’t be infinitely waiting if lock is not available and can perform some other job.

1. **Which is the ideal location for invoking “unlock()” method of ReentrantLock class?**

Ans:

“unlock()” method is used to release the lock acquired by “lock()” or “tryLock()” method of ReentrantLock class. The ideal location for invoking “unlock()” method is finally block just to ensure that lock should be released in any case.

1. **In case of ReentrantLock class, how do you make threads communicate with each other?**

Ans:

In case of ReentrantLock class, we can make threads communicate with each other using Condition API and its “await” and “signalAll” methods.

1. **What happens when any unhandled exception is raised in case of implicit synchronization (synchronized keyword) and explicit synchronization (ReentrantLock class)?**

Ans:

In case of implicit lock, when exception is raised in a synchronized code, lock is automatically released.

But in case of ReentrantLock when exception is raised, lock is not released. That’s the reason it is highly recommended that “unlock()” method should be called inside finally block.