Programming with Java
(Multithreaded Programming)

1. What is a thread? Write the differences between multithreading and multitasking.
Or
Write the differences between Multithreading and Multiprocessing.

Ans:

Thread:

A thread is similar to a program that has a single flow of control. It has a beginning, a body, and an end, and executes commands sequentially. Threads in Java are subprograms of a main application program and share the same memory space, so they are known as lightweight processes.

Multithreading	Multitasking / Multiprocessing
It is a programming concept in which a program is divided into two or more subprograms or threads that are executed at the same time in parallel.	It is an operating system concept in which multiple tasks are performed simultaneously.
It supports the execution of multiple parts of a single program simultaneously.	It supports the execution of multiple programs simultaneously.
The processor has to switch between different parts or threads of a program.	The processor has to switch between different programs or processes.
It is highly efficient.	It is less efficient in comparison to multithreading.
A thread is the smallest unit in multithreading.	A program or process is the smallest unit in a multitasking environment.
It helps in developing efficient programs.	It helps in developing an efficient operating system.

2. What is the usefulness of using thread?

Ans:

Some of the reasons for using threads are that they can help to:

- Make the User Interface (UI) more responsive.
- Take advantage of multiprocessor systems.
- Simplify modeling.
- Perform asynchronous or background processing.

3. What is multithreading? Explain the advantages of multithreading programs.

Or

Explain the concept of multithreaded programming

Ans:

Multithreading:

Multithreading is a conceptual programming paradigm where a program (process) is divided into two or more subprograms (processes), which can be implemented at the same time in parallel. For example, one subprogram can display an animation on the screen while another may build the next animation.

Advantages of Multithreaded Programs:

- *Improve performance:*
  Multithreading enables programmers to do multiple things at one time. They can divide a long program into threads and execute them in parallel. For example, one can send tasks such as printing into the background and continue to perform some other task in the foreground. That improves program performance of programs.
- *Shared resources:*
  Multiple threads share a single address space and other resources.
- *Potential Simplicity:*
  Multiple threads may reduce the complexity of some applications that are inherently suited for threads.

4. Describe the different ways to create a new thread.

Ans:

A new thread can be created in two ways:

1. **By creating a thread class:** Define a class that extends `Thread` class and override its `run()` method with the code required by the thread.
2. **By converting a class to a thread:** Define a class that implements Runnable interface. The Runnable interface has only one method, `run()`, that is to be defined in the method with the code to be executed by the thread.

5. Show with an example how a thread can be created by extending the Thread Class?
Ans:

We can make our class runnable as a thread by extending the class `java.lang.Thread`. This gives us access to all the thread methods directly. It includes the following steps:

1. Declare the class as extending the `Thread` class.
2. Implement the `run()` method that is responsible for executing the sequence of code that the thread will execute.
3. Create a thread object and call the `start()` method to initiate the thread execution.

```
class A extends Thread{
     public void run(){
          for(int i=1;i<=5;i++){
               System.out.println("From Thread A: " +i);
          }
          System.out.println("Exit from A");
     }
}

class B extends Thread {
     public void run(){
          for(int j=1;j<=5;j++){
               System.out.println("From Thread B: "+j);
          }
          System.out.println("Exit from B");
     }
}

class C extends Thread {
     public void run(){
          for(int k=1;k<=5;k++){
               System.out.println("From Thread C: "+k);
          }
          System.out.println("Exit from C");
     }
}

public class ThreadTest {
     public static void main(String[] args) {
          new A().start();
          new B().start();
          new C().start();
     }
}
```

```
A myThing = new A();
B yourThing = new B();
C herThing = new C();
myThing.start();
yourThing.start();
herThing.start();
```

6. When you will prefer multithreading using interfaces rather than extending the Thread class? Write a Java program that makes multiprogramming using Runnable interface.

Or

How thread can be created using Runnable Interface? Show with example.

Or

Write the steps for implementing the Runnable interface. Give an example.

Ans:

We prefer multithreading using interfaces rather than extending the Thread class when it requires extending another class because Java classes cannot have two superclasses.

Implementing the Runnable interface includes the following steps:

1. Declares the `class` as implementing the Runnable interface.
2. Implement the `run()` method.
3. Create a thread by defining an object that is instantiated from this "runnable" class as the target of the thread.
4. Call the threads `start()` method to run the thread.

Here, is a simple program that illustrates the above steps:

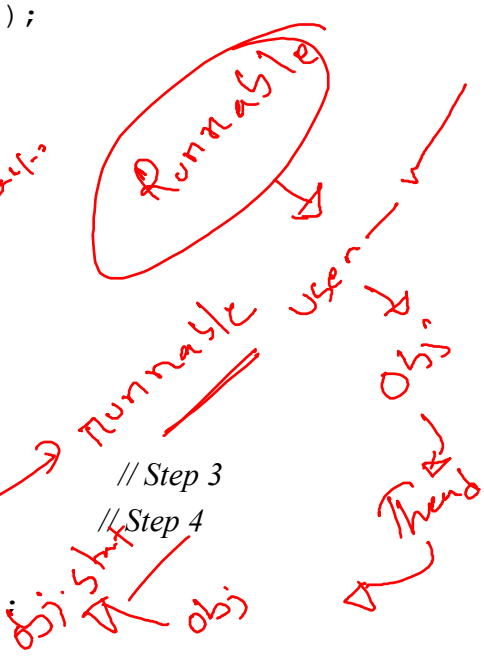
```
class X implements Runnable{                    // Step 1
      public void run(){                        // Step 2
            for (int i=1;i<=10;i++){
                  System.out.println("\tThreadX:"+i);
            }

            System.out.println("End of ThreadX");
      }
}


public class RunnableTest {
      public static void main(String[] args) {

         X runnable=new X();
         Thread threadX=new Thread(runnable);       // Step 3
         threadX.start();                           // Step 4

         System.out.println("End of main Thread");
      }
}
```



7. Suppose, in your Java program you need to create a thread A, but A needs to extend class B. Write some code to show how you can create the thread.

Ans:

We prefer multithreading using Runnable Interfaces when a Thread class requires to extend another class. The code is given below:

```
class B{
     void show(){
          System.out.println("CLASS A");
     }
}


class A extends B implements Runnable{    //Step 1
     public void run(){                   //Step 2
          for (int i=1;i<=10;i++){
               System.out.println("\tThreadX:"+i);
          }
          super.show();
          System.out.println("End of ThreadX");
     }
}


public class RunnableTest {
     public static void main(String[] args){

          A runnable=new A();
          Thread threadX=new Thread(runnable); //Step 3
          threadX.start();                     //step 4

          System.out.println("End of main Thread");
     }
}
```

8. Write a Java program which creates three threads in the main method.
Ans:

```
class NewThread implements Runnable{
      String name;
      Thread t;
      NewThread(String threadname){
            name=threadname;
            t=new Thread(this, name);
            t.start();
      }

      public void run(){
            for (int i=1;i<=10;i++){
                  System.out.println(name+": "+i);
            }
            System.out.println("End of "+name);
      }
}

public class MultiThread {
      public static void main(String[] args){

            new NewThread("A1");
            new NewThread("A2");
            new NewThread("A3");

            System.out.println("End of main Thread");
      }
}
```

9. Discuss the complete life cycle of a thread.
Or
Show the life cycle of a thread. [Only Diagram]
Ans:
During the life-time of a thread, there are many states it can enter. They include:

1. Newborn state.
2. Runnable state.
3. Running state.
4. Blocked state.
5. Dead state.

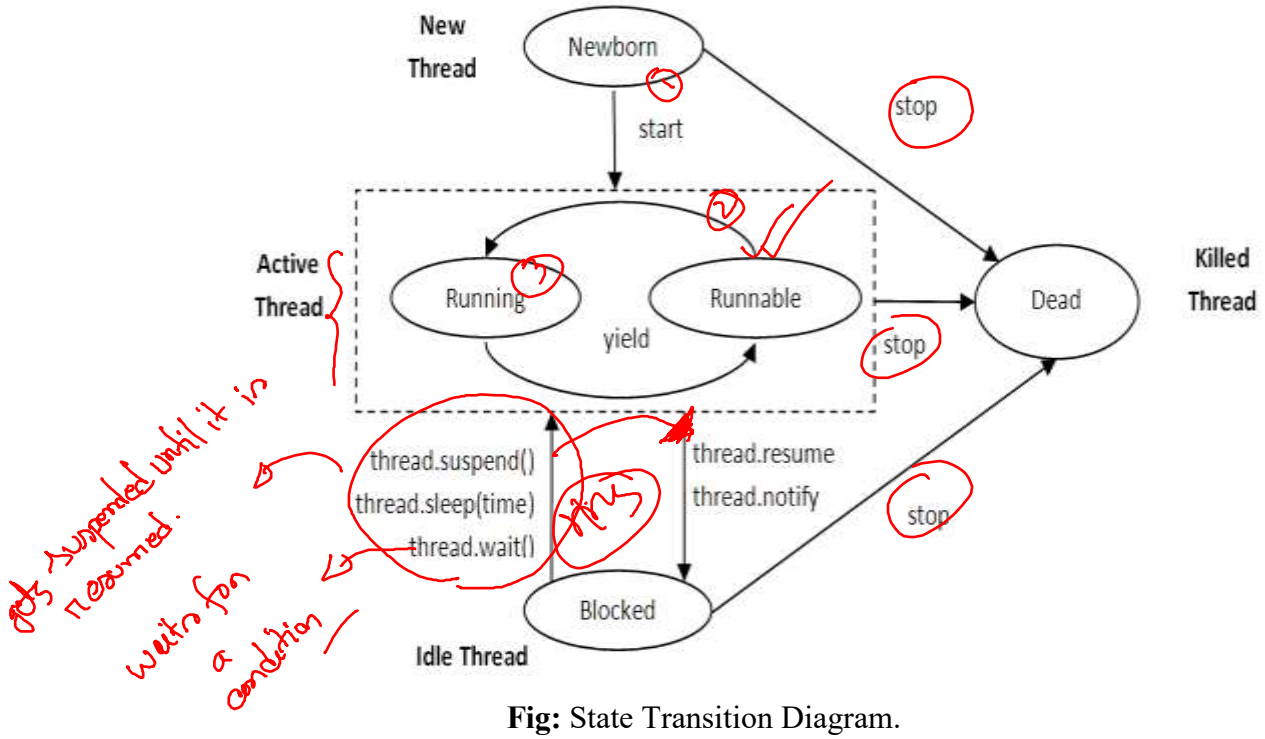


Fig: State Transition Diagram.

Newborn State:
When we create a thread object, the thread is born and is said to be in a *newborn* state. At this state, we can do only one of the following things with it:

- Schedule it for running using `start()` method.
- Kill it using `stop()` method.

Runnable State:
The *runnable* state means that the thread is ready for execution and is waiting for the availability of the processor.

Running State:
Running means that the processor has given its time to the thread for its execution. The thread runs until it relinquishes control on its own or it is preempted by a higher priority thread.

Blocked State:
A thread is said to be *blocked* when it is prevented from entering into the runnable state and subsequently the running state. This happens when the thread is suspended, sleeping, or waiting in order to satisfy certain requirements.

Dead State:
Every thread has a life cycle. A running thread ends its life when it has completed executing its `run()` method. It is a natural death. However, we can kill it by sending the stop message to it at any state thus causing a premature death to it.

10. Describe how a thread can be stopped and blocked?
Ans:

Stopping a Thread:
Whenever we want to stop a thread from running further, we may do so by calling its `stop()` method, like:

```
aThread.stop();
```

This statement causes the thread to move to the *dead* state. A thread will also move to the dead state automatically when it reaches the end of its method. The `stop()` method may be used when the *premature death* of a thread is desired.

Blocking a Thread:
A thread can also be temporarily suspended or blocked from entering into the runnable and subsequently running state by using either of the following thread methods.

```
sleep()      //blocked for a specified time
suspend()    //blocked until further orders
wait()       //blocked until certain condition occurs
```

The thread will return to the runnable state when the specified time is elapsed in the case of `sleep()`, the `resume()` method is invoked in the case of `suspend()` and the `notify()` method is called in the case of `wait()`.

11. Explain the functionality of the following methods of Thread class:
i) join( ) ii) sleep( ) iii) notify( )
Or
Write short notes on: i) run ii) sleep iii) wait iv) yield
Ans:

`yield()` method:
If we want a thread to relinquish control to another thread of equal priority before its turn comes, we can do so by using the `yield()` method.

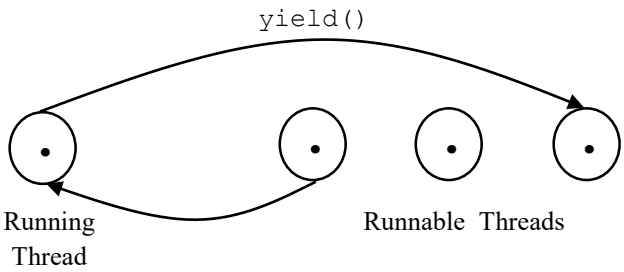


Fig: Relinquish control using `yield()` method.

`suspend()` and `resume()` method:

A running thread has been suspended by using `suspend()` method. A suspended thread can be revived by using the `resume()` method. Useful when we want to suspend a thread for some reason but don't want to kill it.

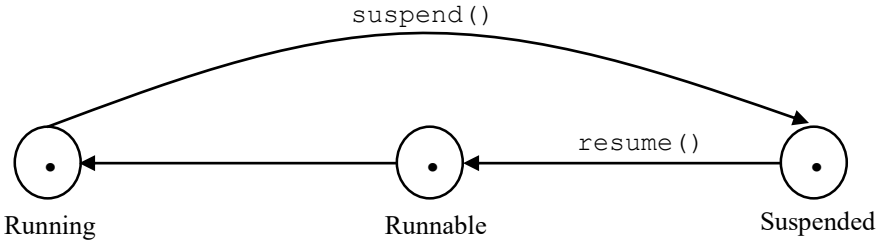


Fig: Relinquish control using `suspend()` method.

`sleep()` method:

We can put a running thread to sleep for a specified time period using the method `sleep(`*time*`)`, where *time* is in milliseconds. This means that the thread is out of the queue during this time period. The thread re-enters the runnable state as soon as this time period is elapsed.

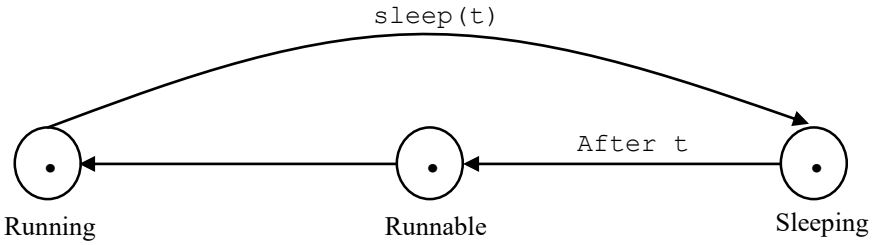


Fig: Relinquish control using `sleep()` method.

`wait()` and `notify()` method:

A running thread has been told to wait until some event occurs. This is done using the `wait()` method. The thread can be scheduled to run again using the `notify()` method.

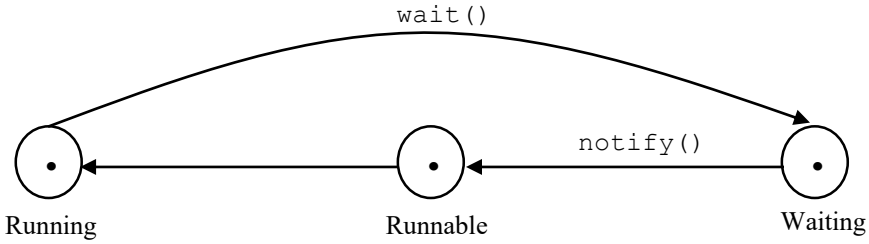


Fig: Relinquish control using `wait()` method.

`stop()` method:

Whenever we want to stop a thread from running further, we may do so by calling its `stop()` method, like:

```
aThread.stop();
```

This statement causes the thread to move to the *dead* state. . A thread will also move to the dead state automatically when it reaches the end of its method. The `stop()` method may be used when the *premature death* of a thread is desired.

`isAlive()` and `join()` method:

`isAlive()` and `join()` methods are two ways that exist to determine whether a thread has finished.

The `isAlive()` method returns true if the thread upon which it is called is still running, it returns false otherwise. Its general form is:

```
final boolean isAlive();
```

While `isAlive()` is occasionally useful, the method that you will more commonly use to wait for a thread to finish is called `join()`, shown here:

```
final void join() throws InterruptedException
```

This method waits until the thread on which it is called terminates. For example, if a thread (name it `T1`) calls the `join()` method on another Thread named `T2`, then `T1` waits for `T2` to complete its execution before it continues from that point. Additional forms of `join()` allow you to specify a maximum amount of time that you want to wait for the specified thread to terminate.

12. Describe all the methods to block, unblock or kill a thread.

Ans:

To block a thread `sleep()`, `suspend()` and `wait()` methods are used. A blocked thread will return to the runnable state when the specified time is elapsed in the case of `sleep()`, the `resume()` method is invoked in the case of `suspend()` and the `notify()` method is called in the case of `wait()`. We can kill a thread by calling `stop()` method.

Then, write the following sections of the previous question.

`sleep()` method:

`suspend()` and `resume()` method:

`wait()` and `notify()` method:

`stop()` method:

13. What are the differences between the blocked state and the dead state of a thread?
Ans:

Blocked State:
A thread is said to be blocked when it is prevented from entering into the runnable state and subsequently the running state. This happens when the thread is suspended, sleeping, or waiting in order to satisfy certain requirements. A blocked thread is considered "not runnable" but not dead and therefore fully qualified to run again.

Dead State:
Every thread has a life cycle. A running thread ends its life when it has completed executing its `run()` method. It is a natural death. However, we can kill it by sending the stop message to it at any state thus causing a premature death to it. A thread can be killed as soon as it is born, or while it is running or even when it is in "not runnable" (blocked) condition.

14. What is the usage of synchronized? Explain with an example.
Or
What is synchronization? When do we use it?
Or
What is synchronization with threads? Why it is necessary in a Java program?
Or
Write short notes on Synchronized keyword.
Ans:

Synchronization is a process of controlling the access of shared resources by multiple threads in such a manner that only one thread can access one resource at a time. In a non-synchronized multithreaded application, it is possible for one thread to modify a shared object while another thread is in the process of using or updating the object's value. The synchronization prevents such type of data corruption. E.g. Synchronizing a function:

```
public synchronized void Method1 () {
    // Appropriate method-related code.
}
E.g. Synchronizing a block of code inside a function:
public myFunction (){
    synchronized (this) {
    // Synchronized code here.
  }
```

Synchronization:
When two or more threads need access to a shared resource, they need some way to ensure that the resource will be used by only one thread at a time. The process by which this is achieved is called synchronization.

Synchronized void update(){
.....
//code here is synchronized
}
	Synchronized(lock-object){
	.....
	//code here is synchronized
	}

We need to use Synchronization if work in a multi-threaded environment. Local variables inside methods are thread-safe. If using global variables (including class-level variables), we need to synchronize the method.

1. Synchronized keyword in java provides locking which ensures mutual exclusive access of shared resource and prevents data race.
2. Synchronized keyword also prevents the reordering of code statement by the compiler which can cause a subtle concurrent issue.
3. Synchronized keywords involve locking and unlocking. before entering into the synchronized method or block thread needs to acquire the lock at this point it reads data from the main memory than the cache and when it releases the lock it flushes the write operation into the main memory which eliminates memory inconsistency errors.

15. Define deadlock along with an example using Thread.
Ans:

Deadlock: A special type of error that relates specifically to multitasking is a deadlock, which occurs when two threads have a circular dependency on a pair of synchronized objects.

For example, assume that thread *A* must access *Method1* before it can release *Method2*, but thread *B* cannot release *Method1* until it gets hold of *Method2*. As these are mutually exclusive conditions, a deadlock occurs.

```
Thread A
synchronized method2(){
      synchronized method1{
            .....
            .....
      }
}

Thread B
synchronized method1(){
      synchronized method2{
            .....
            .....
      }
}
```

16. How do we set priorities for threads?
Ans:

To set a thread's priority use the `setPriority()` method as follows:

```
ThreadName.setPriority(intNumber);
```

The `intNumber` is an integer value to which the thread's priority is set. The thread class defines several priority constants:

```
MIN_PRIORITY    = 1
NORM_PRIORITY   = 5
MAX_PRIORITY    = 10
```

The `intNumber` may assume one of these constants or any value between 1 and 10. The default setting is NORM_PRIORITY.

17. Write a program in Java to explain how different priorities can be assigned to different threads?
Ans:

```
class At extends Thread {
     public void run() {
          System.out.println("threadA started");
          for (int i = 1; i <= 4; i++) {
               System.out.println("\tFrom Thread A : i=" + i);
          }
          System.out.println("Exit from A");
     }
}

class Bt extends Thread {
     public void run() {
          System.out.println("threadB started");
          for (int j = 1; j <= 4; j++) {
               System.out.println("\tFrom Thread B : j=" + j);
          }
          System.out.println("Exit from B");
     }
}
```

```
class Ct extends Thread {
     public void run() {
          System.out.println("threadC started");
          for (int k = 1; k <= 4; k++) {
               System.out.println("\tFrom Thread C : k= " + k);
          }
          System.out.println("Exit from C");
     }
}

public class ThreadPriority {
     public static void main(String[] args) {

          At threadA = new At();
          Bt threadB = new Bt();
          Ct threadC = new Ct();

          threadC.setPriority(Thread.MAX_PRIORITY);
          threadB.setPriority(threadA.getPriority() + 1);
          threadA.setPriority(Thread.MIN_PRIORITY);

          System.out.println("Start Thread A");
          threadA.start();

          System.out.println("Start Thread B");
          threadB.start();

          System.out.println("Start Thread C");
          threadC.start();

          System.out.println("End of main Thread ");
     }
}
```

18. Write a Java program to change the priorities of its main thread as follows:

i) Set its priority below normal.

ii) Give it the name: CSE

Or

Write a simple Java program to change the properties of Main thread.

Ans:

The below program changes the name and priority of the Main thread.

```
public class MainThreadPriority {
 public static void main (String args[]){
   Thread t = Thread.currentThread();
   System.out.println("Current Thread "+t);

   t.setName("CSE");
   System.out.println("After name change: "+t);

   t.setPriority(t.getPriority()-2);
   System.out.println("After priority change:"+t.getPriority());
  }
}
```

Output:

```
Current Thread Thread[main,5,main]
After name change: Thread[CSE,5,main]
Priority after change: 3
```