Q2/ What is a ThreadPool, how to create a thread pool of 4 threads, and write down the difference between the Callable and Runnable interface.
A thread pool reuses previously created threads to execute current tasks and offers a solution to the problem of thread cycle overhead and resource thrashing.

Creating thread pool with four threads:

```
class PrintJob implements Runnable{
            String name;
            PrintJob(Sting name){
                  this.name=name;
            }
            public void run(){
                    System.out.println(name +" job started by
Thread:"+Thread.currentThread().getName());

           try{
                 Thread.sleep(5000);
              }catch(InturreptedException e){
                        e.printStackTrace();
              }

                 System.out.println(name +"..job completed by Thread
:"+Thread.currentThread().getName()); }
}

 class Main{
public static void main(String[] args){
            PrintJob[] jobs={
                            new PrintJob("Ram"),
                            new PrintJob("Ravi"),
                            new PrintJob("Anil"),
                            new PrintJob("Shiva"),
                            new PrintJob("Pawan"),
                            new PrintJob("Suresh")
              };
 ExecutorService service = Executors.newFixedThreadPool(4);
for(PrintJob job:jobs){
        service.submit(job);
}

 service.shutdown();//to shutdown the executorService.
 }
 }
```

Difference between runnable and callable interface:

| Callable | Runnable |
|---|---|
| • It is a part of java.lang package since java 1.0<br><br>• It cannot return the result of computation.<br><br>• It cannot throw a checked exception.<br><br>• In a runnable interface, one needs to override the run() method in java. | • It is a part of the java.util.concurrent package since java 1.5.<br><br>• It can return the result of the parallel processing of a task.<br><br>• It can throw a checked exception.<br><br>• In order to use callable, you need to override the call() |

Q3/- What do you mean by a Race condition, How to solve a race condition, give a proper example.
Java is a multi-threaded programming language and there is a higher risk to occur race conditions. Because the same resource may be accessed by multiple threads at the same time and may change the data.

A race-condition is a condition in which the critical section (a part of the program where shared memory is accessed) is concurrently executed by two or more threads. It leads to incorrect behavior of a program.

Race condition can be solved by synchronizing the process.
Example :

```
class Common{
          public void fun1(Stirng name){
          System.out.print("Welcome");
          try{
              Thread.sleep(1000);
          }catch(Exception ee){
       } System.out.println(name);
}

 class ThreadA extends Thread{
          Common c;
          String name;
          public ThreadA(Common c,String name) {
          this.c=c; this.name=name;
         }

         @Override
          public void run() {
                c.fun1(name);
          }
      }

class ThreadB extends Thread{
         Common c;
         String name;
         public ThreadB(Common c,String name) {
                this.c=c; this.name=name;
         }

         @Override
          public void run() {
                c.fun1(name);
          }
 }

 class Main{
        public static void main(String[] args){
                Common c=new Common();
                //sharing same Common object to two thread
                ThreadA t1=new ThreadA(c,"Ram");
                ThreadB t2=new ThreadB(c,"Shyam");

                t1.start();
               t2.start();
         }
 }
```

Q4/ Explain about thread synchronization(inter-thread communication).
It means two synchronized threads communicate each other.

Two synchronized thread can communicate each other by using some methods present in Object class, those methods are wait(), notify(), notifyAll().

By using above methods we can gain partial control on the scheduling mechanism which is supervised by the thread-scheduler.

To gain this partial control the threads should have a sign of mutual understanding between them .they should be able to communicate with each other.

Whenever we need to suspend a synchronized thread unconditionally then we use wait() method.

Whenever we need to resume a suspended(waiting) thread then we use notify() method
this is known as thread-synchronization or inter-thread communication.

In the inter-thread communication the thread which require updation it has to call wait() method. The thread which performing updation it will call notify() method, so that waiting thread will gets the notification and it continues its execution with those updation.

To call wait() or notify() method on any object we must have that particular object lock otherwise we will get a runtime exception called IllegalMonitorStateException.

Once a thread calls wait() method on any object, first it releases the lock immediately of that particular object and then it enters into the waiting state immediately.
Once a thread calls notify() method on any object it also releases the lock of that object but not immediately.

Wait and notify or notifyAll method belongs from Object class because "a thread" can call these methods on any java object.

Q5/- What is the difference between the sleep and wait method?

| **Sleep** | **Wait** |
|---|---|
| ● Sleep() method belongs to thread class.<br><br>● Sleep() method does not release the lock on object during synchronization.<br><br>● There is no need to call sleep() from synchronized context.<br><br>● Sleep() is a static method.<br><br>● Sleep() has two overloaded methods:<br>1. sleep(long millis)millis: millilseconds<br>2. sleep(long millis,int nanos) nanos: Nanoseconds | ● Wait() method belongs to object class.<br><br>● Wait() method releases lock during synchronization.<br><br>● Wait() should be called only from synchronized context.<br><br>● Wait() is not a static method.<br><br>● Wait() has three overloaded methods:<br>1. Wait()<br>2. Wait(long timeout)<br>3. Wait(long timeout, int nanos) |