Daemon thread in Java:

Daemon thread is a low priority thread that runs in background to perform tasks such as garbage collection.

Properties:

- They cannot prevent the JVM from exiting when all the user threads finish their execution.
- JVM terminates itself when all user threads finish their execution.
- If JVM finds running daemon thread, it terminates the thread and after that shutdown itself. JVM does not care whether Daemon thread is running or not.

Java's sleep() and wait():

sleep() is a method which is used to pause the process for few seconds or the time we want to. But in case of **wait()** method, thread goes in waiting state and it won't come back automatically until we call the notify() or notifyAll().

The major difference is that wait() releases the lock or monitor while sleep() doesn't releases the lock or monitor while waiting. wait() is used for inter-thread communication while sleep() is used to introduce pause on execution, generally.

Difference between wait() and sleep():

wait()	sleep()
wait() method releases the lock	sleep() method doesn't release the lock.
is the method of Object class	is the method of Thread class
is the non-static method	is the static method
should be notified by notify() or notifyAll() methods	after the specified amount of time, sleep is completed.

Why wait(), notify() and notifyAll() methods are defined in Object class not Thread class?

It is because they are related to lock and object has a lock.

Java's notify() and notifyAll():

The method notify() wakes up a single thread that is waiting on this object's monitor.

The method notifyAll() wakes up all threads that are waiting on this object's monitor.

Java's yield():

yield(): Suppose there are three threads t1, t2, and t3. Thread t1 gets the processor and starts its execution and thread t2 and t3 are in Ready/Runnable state. Completion time for thread t1 is 5 hour and completion time for t2 is 5 minutes. Since t1 will complete its execution after 5 hours, t2 has to wait for 5 hours to just finish 5 minutes job. In such scenarios where one thread is taking too much time to complete its execution, we need a way to prevent execution of a thread in between if something important is pending. The method yield() helps us in doing so.

The method **yield()** basically means that the thread is not doing anything particularly important and if any other threads or processes need to be run, they should run. Otherwise, the current thread will continue to run.

yield() vs sleep():

yield() indicates that the thread is not doing anything particularly important and if any other threads or processes need to be run, they can. Otherwise, the current thread will continue to run.

sleep() causes the thread to definitely stop executing for a given amount of time; if no other thread or process needs to be run, the CPU will be idle (and probably enter a power saving mode).

```
// Program for inter-thread communication in Java

class Q
{
 int n;
 boolean valueSet=false;

 synchronized int get()
 {
  while(!valueSet)
  {
   try
   {
     wait();
   }
   catch(InterruptedException e)
   {
   }
  }
  System.out.println("got: "+n);
  valueSet=false;
  notify();
  return n;
 }

 synchronized void put(int n)
 {
    while(valueSet)
  {
   try
   {
     wait();
   }
   catch(InterruptedException e)
   {
   }
  }
  this.n=n;
  valueSet=true;
  System.out.println("put: "+n);
  notify();
 }
}
```

```
class Producer implements Runnable
{
 Q q;
 Producer(Q q)
 {
  this.q=q;
  new Thread(this, "producer").start();
 }
 public void run()
 {
  int i=0;
  int count=5;
  while(count>0)
  {
    q.put(i++);
    count--;
  }
 }
}

class Consumer implements Runnable
{
  Q q;
  Consumer(Q q)
  {
    this.q=q;
    new Thread(this,"consumer").start();
  }

  public void run()
  {
    int count=5;
    while(count>0)
    {
      q.get();
      count--;
    }
  }
}
```

```
class IPC2{
        public static void main(String args[])
       {
        Q q=new Q();
        new Producer(q);
        new Consumer(q);
       }
}
```

```
o/p: put : 0
     got : 0
     put : 1
     got : 1
     put : 2
     got : 2
     put : 3
     got : 3
     put : 4
     got : 4
```

//Program for illustrating thread priorities

```
import java.lang.Thread;
class ThreadPrio extends Thread{
 public static void main(String args[])
 {
  System.out.println("current priority : "+Thread.currentThread().getPriority());

  //setting the priority of the current thread to the MAX_PRIORITY
  Thread.currentThread().setPriority(Thread.MAX_PRIORITY);

  System.out.println("Max priority : "+Thread.currentThread().getPriority());

  //setting the priority of the current thread to the MIN_PRIORITY
  Thread.currentThread().setPriority(Thread.MIN_PRIORITY);

  System.out.println("Min priority : "+Thread.currentThread().getPriority());
 }
}
```

```
o/p: current priority : 5
     Max priority : 10
     Min priority : 1
```