Joining Threads in Java

**java.lang.Thread** class provides the join() method which allows one thread to wait until another thread completes its execution. If **t** is a Thread object whose thread is currently executing, then **t.join();** it causes the current thread to pause its execution until thread it join completes its execution.  
If there are multiple threads calling the join() methods that means overloading on join allows the programmer to specify a waiting period. However, as with sleep, join is dependent on the OS for timing, so you should not assume that join will wait exactly as long as you specify.  
There are three overloaded join functions.

1. **join():**It will put the current thread on wait until the thread on which it is called is dead. If thread is interrupted then it will throw InterruptedException.  
   **Syntax:**
2. public final void join()
3. **join(long millis)**:It will put the current thread on wait until the thread on which it is called is dead or wait for specified time (milliseconds).  
   **Syntax:**
4. public final synchronized void join(long millis)
5. **join(long millis, int nanos):**It will put the current thread on wait until the thread on which it is called is dead or wait for specified time (milliseconds + nanos).  
   **Syntax:**
6. public final synchronized void join(long millis, int nanos)

|  |
| --- |
| // Java program to explain the  // concept of joining a thread.  import java.io.\*;    // Creating thread by creating the  // objects of that class  class ThreadJoining extends Thread  {      @Override      public void run()      {          for (int i = 0; i < 2; i++)          {              try              {                  Thread.sleep(500);                  System.out.println("Current Thread: "                          + Thread.currentThread().getName());              }                catch(Exception ex)              {                  System.out.println("Exception has" +                                  " been caught" + ex);              }              System.out.println(i);          }      }  }    class GFG  {      public static void main (String[] args)      {            // creating two threads          ThreadJoining t1 = new ThreadJoining();          ThreadJoining t2 = new ThreadJoining();          ThreadJoining t3 = new ThreadJoining();            // thread t1 starts          t1.start();            // starts second thread after when          // first thread t1 is died.          try          {              System.out.println("Current Thread: "                    + Thread.currentThread().getName());              t1.join();          }            catch(Exception ex)          {              System.out.println("Exception has " +                                  "been caught" + ex);          }            // t2 starts          t2.start();            // starts t3 after when thread t2 is died.          try          {              System.out.println("Current Thread: "                   + Thread.currentThread().getName());              t2.join();          }            catch(Exception ex)          {              System.out.println("Exception has been" +                                      " caught" + ex);          }            t3.start();      }  } |

1. Run on IDE
2. output:
3. Current Thread: main
4. Current Thread: Thread-0
5. 0
6. Current Thread: Thread-0
7. 1
8. Current Thread: main
9. Current Thread: Thread-1
10. 0
11. Current Thread: Thread-1
12. 1
13. Current Thread: Thread-2
14. 0
15. Current Thread: Thread-2
16. 1

In the above example we can see clearly second thread t2 starts after first thread t1 is died and t3 will start its execution after second thread t2 is died.