**Day 46**

**Java join() method**

The join() method in Java is provided by the java.lang.Thread class that permits one thread to wait until the other thread to finish its execution. Suppose *th* be the object the class Thread whose thread is doing its execution currently, then the *th.join();* statement ensures that *th* is finished before the program does the execution of the next statement. When there are more than one thread invoking the join() method, then it leads to overloading on the join() method that permits the developer or programmer to mention the waiting period. However, similar to the sleep() method in Java, the join() method is also dependent on the operating system for the timing, so we should not assume that the join() method waits equal to the time we mention in the parameters. The following are the three overloaded join() methods.

Description of The Overloaded join() Method

**join():** When the join() method is invoked, the current thread stops its execution and the thread goes into the wait state. The current thread remains in the wait state until the thread on which the join() method is invoked has achieved its dead state. If interruption of the thread occurs, then it throws the InterruptedException.

**Syntax:**

1. **public** **final** **void** join() **throws** InterruptedException

**join(long mls):** When the join() method is invoked, the current thread stops its execution and the thread goes into the wait state. The current thread remains in the wait state until the thread on which the join() method is invoked called is dead or the wait for the specified time frame(in milliseconds) is over.

**Syntax:**

1. **public** **final** **synchronized** **void** join(**long** mls) **throws** InterruptedException, where mls is in milliseconds

**join(long mls, int nanos):** When the join() method is invoked, the current thread stops its execution and go into the wait state. The current thread remains in the wait state until the thread on which the join() method is invoked called is dead or the wait for the specified time frame(in milliseconds + nanos) is over.

**Syntax:**

1. **public** **final** **synchronized** **void** join(**long** mls, **int** nanos) **throws** InterruptedException, where mls is in milliseconds.

Example of join() Method in Java

The following program shows the usage of the join() method.

**FileName:** ThreadJoinExample.java

1. // A Java program for understanding
2. // the joining of threads
4. // import statement
5. **import** java.io.\*;
7. // The ThreadJoin class is the child class of the class Thread
8. **class** ThreadJoin **extends** Thread
9. {
10. // overriding the run method
11. **public** **void** run()
12. {
13. **for** (**int** j = 0; j < 2; j++)
14. {
15. **try**
16. {
17. // sleeping the thread for 300 milli seconds
18. Thread.sleep(300);
19. System.out.println("The current thread name is: " + Thread.currentThread().getName());
20. }
21. // catch block for catching the raised exception
22. **catch**(Exception e)
23. {
24. System.out.println("The exception has been caught: " + e);
25. }
26. System.out.println( j );
27. }
28. }
29. }
31. **public** **class** ThreadJoinExample
32. {
33. // main method
34. **public** **static** **void** main (String argvs[])
35. {
37. // creating 3 threads
38. ThreadJoin th1 = **new** ThreadJoin();
39. ThreadJoin th2 = **new** ThreadJoin();
40. ThreadJoin th3 = **new** ThreadJoin();
42. // thread th1 starts
43. th1.start();
45. // starting the second thread after when
46. // the first thread th1 has ended or died.
47. **try**
48. {
49. System.out.println("The current thread name is: "+ Thread.currentThread().getName());
51. // invoking the join() method
52. th1.join();
53. }
55. // catch block for catching the raised exception
56. **catch**(Exception e)
57. {
58. System.out.println("The exception has been caught " + e);
59. }
61. // thread th2 starts
62. th2.start();
64. // starting the th3 thread after when the thread th2 has ended or died.
65. **try**
66. {
67. System.out.println("The current thread name is: " + Thread.currentThread().getName());
68. th2.join();
69. }
71. // catch block for catching the raised exception
72. **catch**(Exception e)
73. {
74. System.out.println("The exception has been caught " + e);
75. }
77. // thread th3 starts
78. th3.start();
79. }
80. }

**Output:**

The current thread name is: main

The current thread name is: Thread - 0

0

The current thread name is: Thread - 0

1

The current thread name is: main

The current thread name is: Thread - 1

0

The current thread name is: Thread - 1

1

The current thread name is: Thread - 2

0

The current thread name is: Thread - 2

1

**Explanation:** The above program shows that the second thread th2 begins after the first thread th1 has ended, and the thread th3 starts its work after the second thread th2 has ended or died.

The Join() Method: InterruptedException

We have learnt in the description of the join() method that whenever the interruption of the thread occurs, it leads to the throwing of InterruptedException. The following example shows the same.

**FileName:** ThreadJoinExample1.java

1. **class** ABC **extends** Thread
2. {
3. Thread threadToInterrupt;
4. // overriding the run() method
5. **public** **void** run()
6. {
7. // invoking the method interrupt
8. threadToInterrupt.interrupt();
9. }
10. }

13. **public** **class** ThreadJoinExample1
14. {
15. // main method
16. **public** **static** **void** main(String[] argvs)
17. {
18. **try**
19. {
20. // creating an object of the class ABC
21. ABC th1 = **new** ABC();
23. th1.threadToInterrupt = Thread.currentThread();
24. th1.start();
26. // invoking the join() method leads
27. // to the generation of InterruptedException
28. th1.join();
29. }
30. **catch** (InterruptedException ex)
31. {
32. System.out.println("The exception has been caught. " + ex);
33. }
34. }
35. }

**Output:**

The exception has been caught. java.lang.InterruptedException

Some More Examples of the join() Method

Let' see some other examples.

**Filename:** TestJoinMethod1.java

1. **class** TestJoinMethod1 **extends** Thread{
2. **public** **void** run(){
3. **for**(**int** i=1;i<=5;i++){
4. **try**{
5. Thread.sleep(500);
6. }**catch**(Exception e){System.out.println(e);}
7. System.out.println(i);
8. }
9. }
10. **public** **static** **void** main(String args[]){
11. TestJoinMethod1 t1=**new** TestJoinMethod1();
12. TestJoinMethod1 t2=**new** TestJoinMethod1();
13. TestJoinMethod1 t3=**new** TestJoinMethod1();
14. t1.start();
15. **try**{
16. t1.join();
17. }**catch**(Exception e){System.out.println(e);}
19. t2.start();
20. t3.start();
21. }
22. }

**Output:**

1

2

3

4

5

1

1

2

2

3

3

4

4

5

5

We can see in the above example, when t1 completes its task then t2 and t3 starts executing.

join(long miliseconds) Method Example

**Filename:** TestJoinMethod2.jav

1. **class** TestJoinMethod2 **extends** Thread{
2. **public** **void** run(){
3. **for**(**int** i=1;i<=5;i++){
4. **try**{
5. Thread.sleep(500);
6. }**catch**(Exception e){System.out.println(e);}
7. System.out.println(i);
8. }
9. }
10. **public** **static** **void** main(String args[]){
11. TestJoinMethod2 t1=**new** TestJoinMethod2();
12. TestJoinMethod2 t2=**new** TestJoinMethod2();
13. TestJoinMethod2 t3=**new** TestJoinMethod2();
14. t1.start();
15. **try**{
16. t1.join(1500);
17. }**catch**(Exception e){System.out.println(e);}
19. t2.start();
20. t3.start();
21. }
22. }

**Output:**

1

2

3

1

4

1

2

5

2

3

3

4

4

5

5

Naming Thread

The Thread class provides methods to change and get the name of a thread. By default, each thread has a name, i.e. thread-0, thread-1 and so on. By we can change the name of the thread by using the setName() method. The syntax of setName() and getName() methods are given below:

1. **public** String getName(): is used to **return** the name of a thread.
2. **public** **void** setName(String name): is used to change the name of a thread.

We can also set the name of a thread directly when we create a new thread using the constructor of the class.