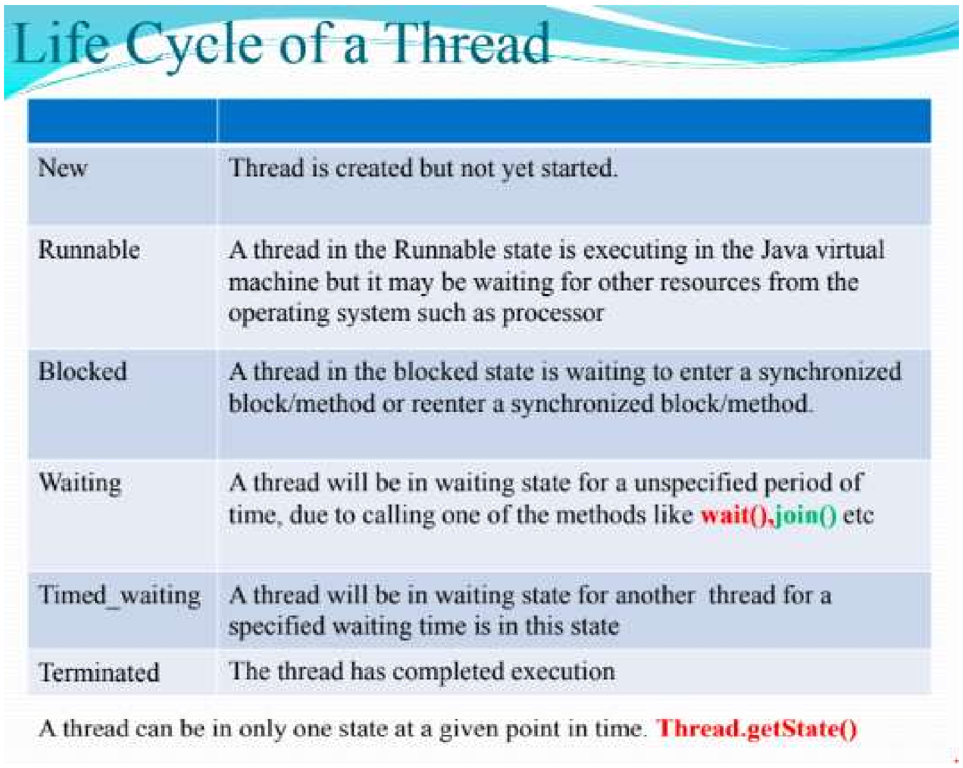
Thread Life Cycle

Once we have created a thread, a thread will be in any of these below mentioned 7 lifecycle stage

1. New stage
2. Runnable Stage
3. Running stage
4. Blocked stage
5. Waiting stage
6. Timed-Waiting stage
7. Terminated stage

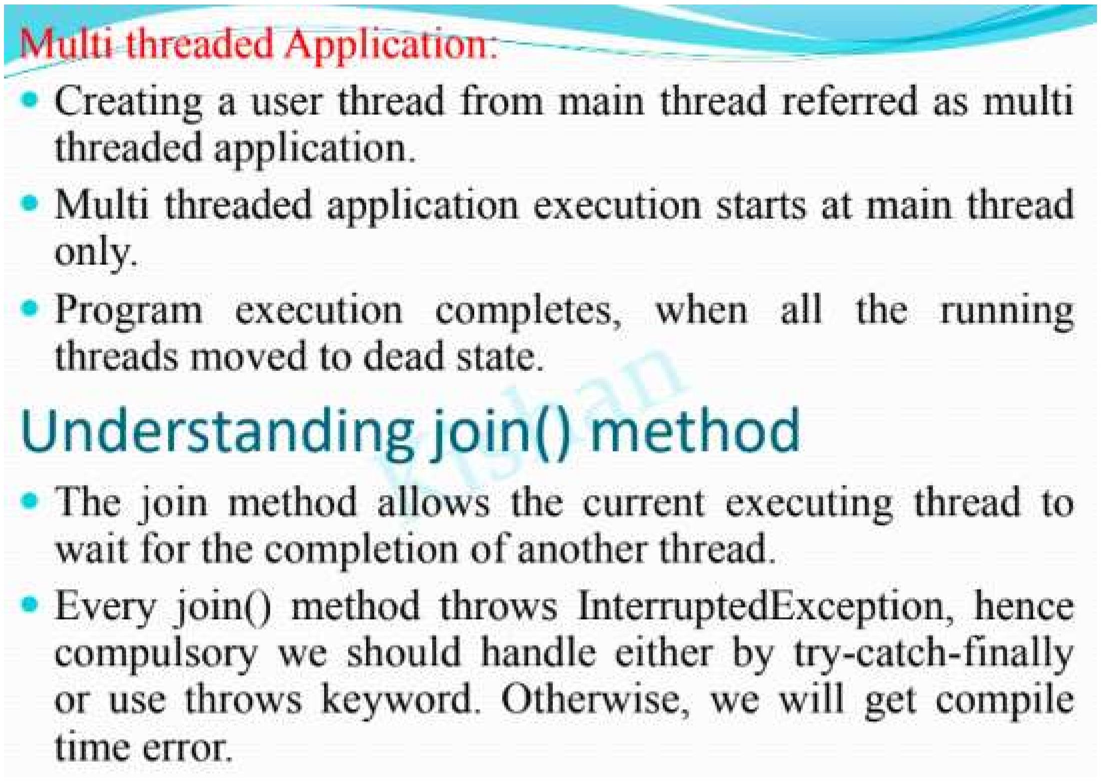


Life Cycle of a Thread

New	Thread is created but not yet started.
Runnable	A thread in the Runnable state is executing in the Java virtual machine but it may be waiting for other resources from the operating system such as processor
Blocked	A thread in the blocked state is waiting to enter a synchronized block/method or reenter a synchronized block/method.
Waiting	A thread will be in waiting state for a unspecified period of time, due to calling one of the methods like wait(),join() etc
Timed_waiting	A thread will be in waiting state for another thread for a specified waiting time is in this state
Terminated	The thread has completed execution

A thread can be in only one state at a given point in time. **Thread.getState()**

Thread class methods



Multi threaded Application:

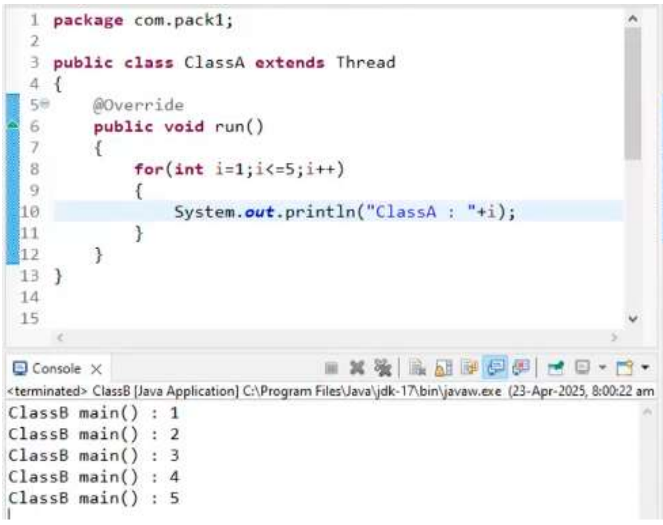
- Creating a user thread from main thread referred as multi threaded application.
- Multi threaded application execution starts at main thread only.
- Program execution completes, when all the running threads moved to dead state.

Understanding join() method

- The join method allows the current executing thread to wait for the completion of another thread.
- Every join() method throws InterruptedException, hence compulsory we should handle either by try-catch-finally or use throws keyword. Otherwise, we will get compile time error.

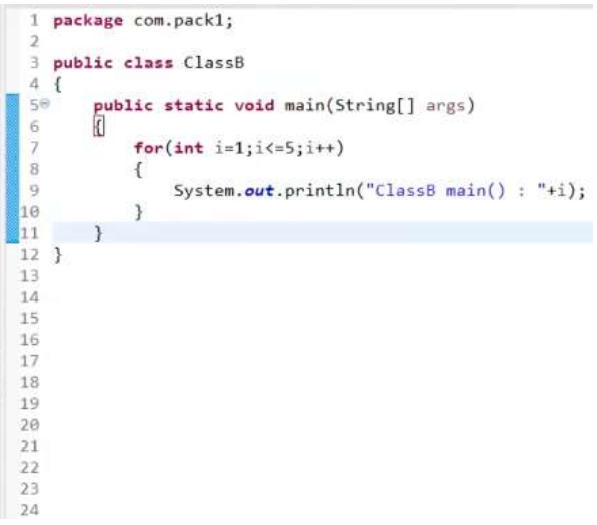
Join() allows the current executing thread to wait for the completion of another thread

Every join() throws interrupted exception, hence compulsory we should handle either by using try, catch, finally or use throws key word otherwise we will be getting a compile time error

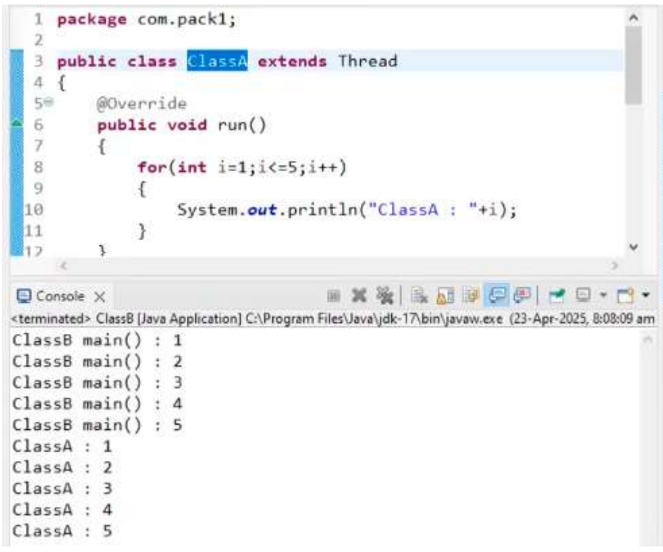


```java
package com.pack1;

public class ClassA extends Thread
{
    @Override
    public void run()
    {
        for(int i=1;i<=5;i++)
        {
            System.out.println("ClassA : "+i);
        }
    }
}
```

```
<terminated> ClassB [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe (23-Apr-2025, 8:00:22 am
ClassB main() : 1
ClassB main() : 2
ClassB main() : 3
ClassB main() : 4
ClassB main() : 5
```

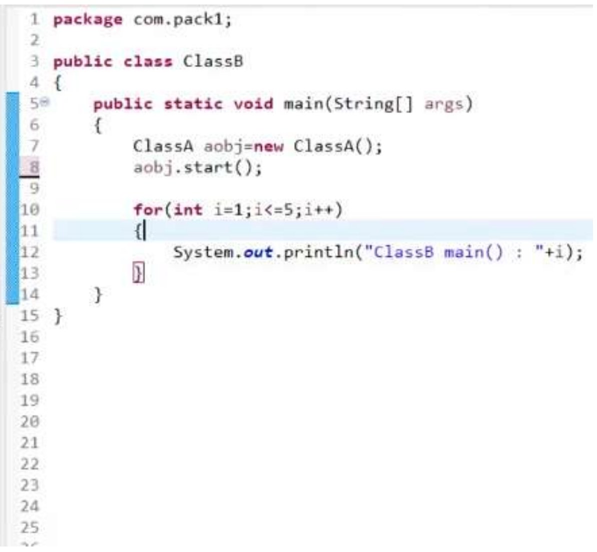


```java
package com.pack1;

public class ClassB
{
    public static void main(String[] args)
    {
        for(int i=1;i<=5;i++)
        {
            System.out.println("ClassB main() : "+i);
        }
    }
}
```

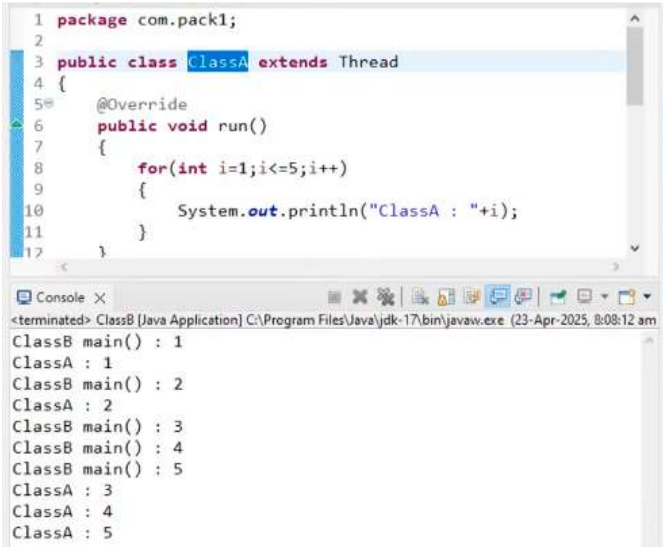


```java
package com.pack1;

public class ClassA extends Thread
{
    @Override
    public void run()
    {
        for(int i=1;i<=5;i++)
        {
            System.out.println("ClassA : "+i);
        }
    }
```

```
<terminated> ClassB [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe (23-Apr-2025, 8:08:09 am
ClassB main() : 1
ClassB main() : 2
ClassB main() : 3
ClassB main() : 4
ClassB main() : 5
ClassA : 1
ClassA : 2
ClassA : 3
ClassA : 4
ClassA : 5
```

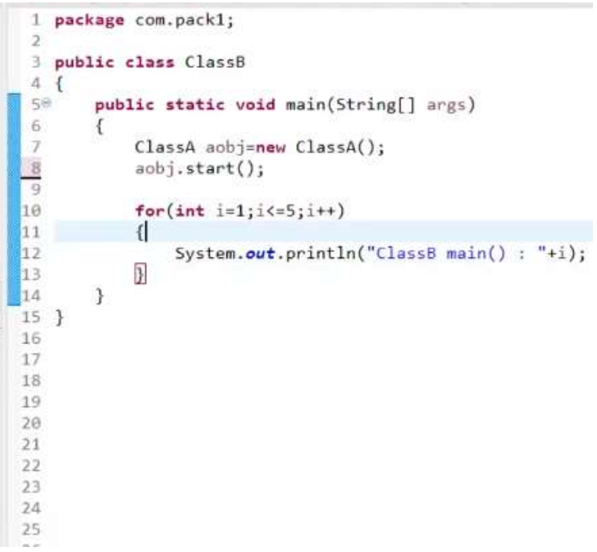


```java
package com.pack1;

public class ClassB
{
    public static void main(String[] args)
    {
        ClassA aobj=new ClassA();
        aobj.start();

        for(int i=1;i<=5;i++)
        {
            System.out.println("ClassB main() : "+i);
        }
    }
}
```

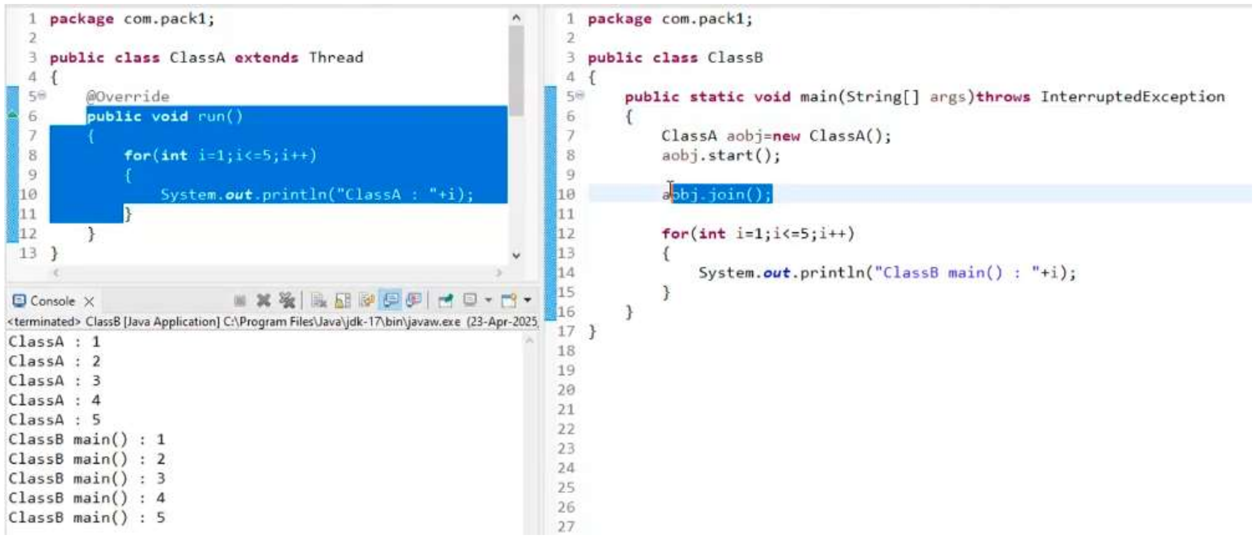


```java
package com.pack1;

public class ClassA extends Thread
{
    @Override
    public void run()
    {
        for(int i=1;i<=5;i++)
        {
            System.out.println("ClassA : "+i);
        }
    }
```

```
<terminated> ClassB [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe (23-Apr-2025, 8:08:12 am
ClassB main() : 1
ClassA : 1
ClassB main() : 2
ClassA : 2
ClassB main() : 3
ClassB main() : 4
ClassB main() : 5
ClassA : 3
ClassA : 4
ClassA : 5
```



```java
package com.pack1;

public class ClassB
{
    public static void main(String[] args)
    {
        ClassA aobj=new ClassA();
        aobj.start();

        for(int i=1;i<=5;i++)
        {
            System.out.println("ClassB main() : "+i);
        }
    }
}
```

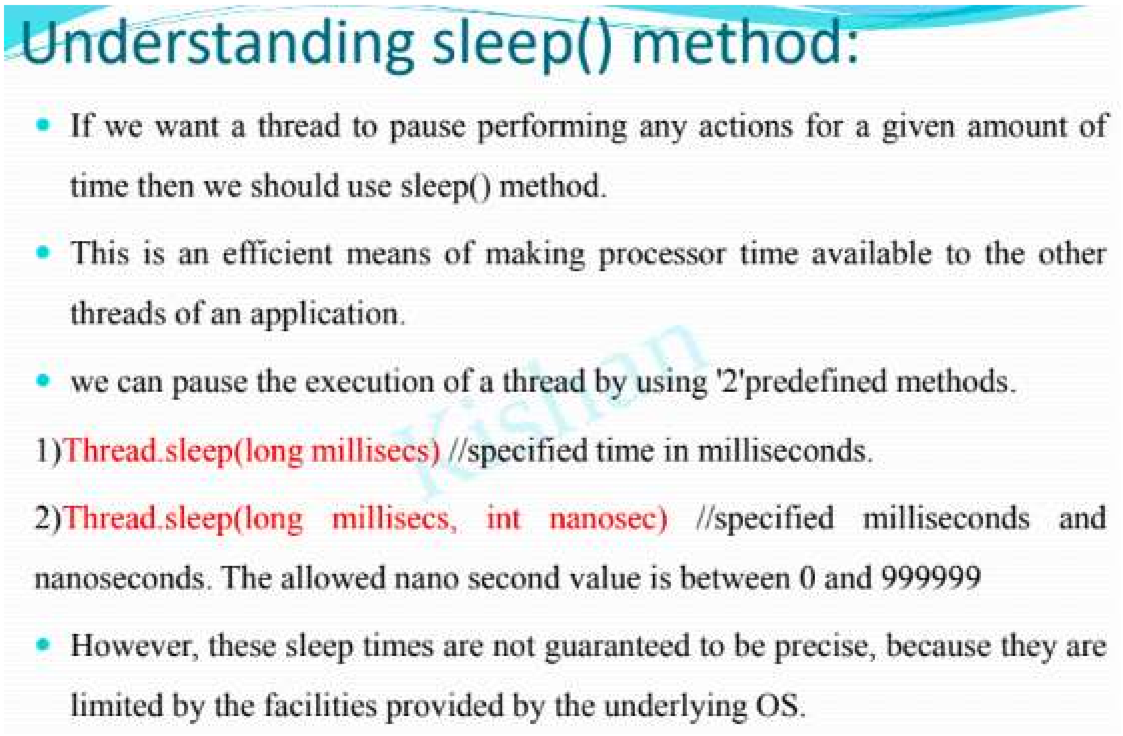


```java
package com.pack1;

public class ClassA extends Thread
{
    @Override
    public void run()
    {
        for(int i=1;i<=5;i++)
        {
            System.out.println("ClassA : "+i);
        }
    }
}
```

```
<terminated> ClassB [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe (23-Apr-2025
ClassA : 1
ClassA : 2
ClassA : 3
ClassA : 4
ClassA : 5
ClassB main() : 1
ClassB main() : 2
ClassB main() : 3
ClassB main() : 4
ClassB main() : 5
```

```java
package com.pack1;

public class ClassB
{
    public static void main(String[] args)throws InterruptedException
    {
        ClassA aobj=new ClassA();
        aobj.start();

        aobj.join();

        for(int i=1;i<=5;i++)
        {
            System.out.println("ClassB main() : "+i);
        }
    }
}
```

Understanding sleep()



Understanding sleep() method:

- If we want a thread to pause performing any actions for a given amount of time then we should use sleep() method.
- This is an efficient means of making processor time available to the other threads of an application.
- we can pause the execution of a thread by using '2'predefined methods.

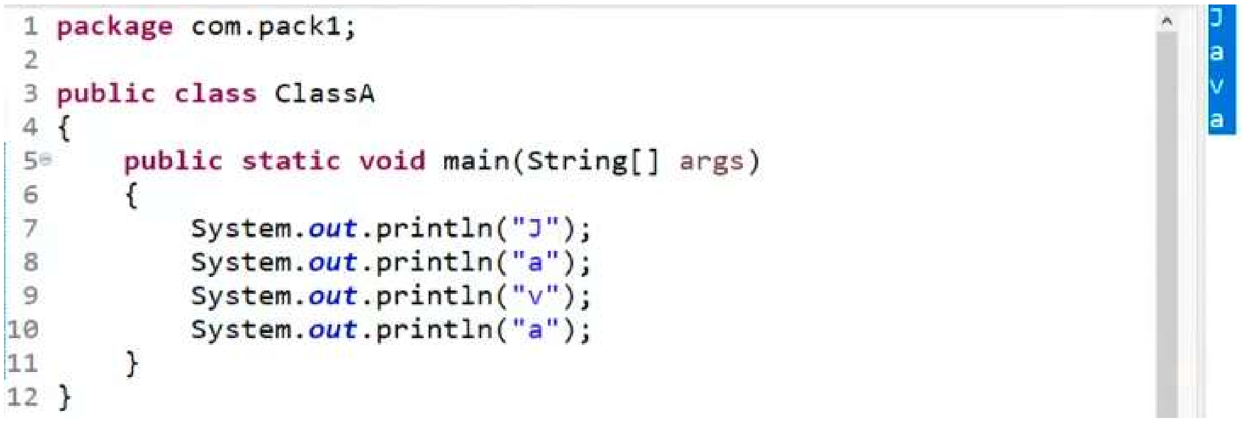
1)Thread.sleep(long millisecs) //specified time in milliseconds.

2)Thread.sleep(long millisecs, int nanosec) //specified milliseconds and nanoseconds. The allowed nano second value is between 0 and 999999

- However, these sleep times are not guaranteed to be precise, because they are limited by the facilities provided by the underlying OS.

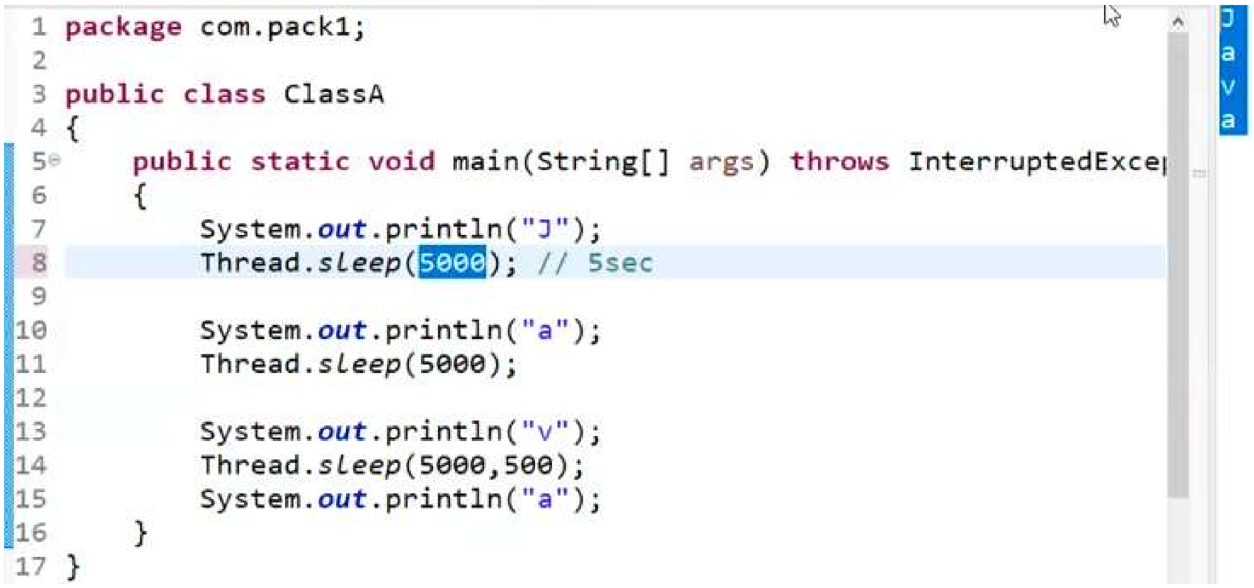
Sleep() is a static method which is present in thread class.

However, the sleep timings of a thread are highly system dependent.



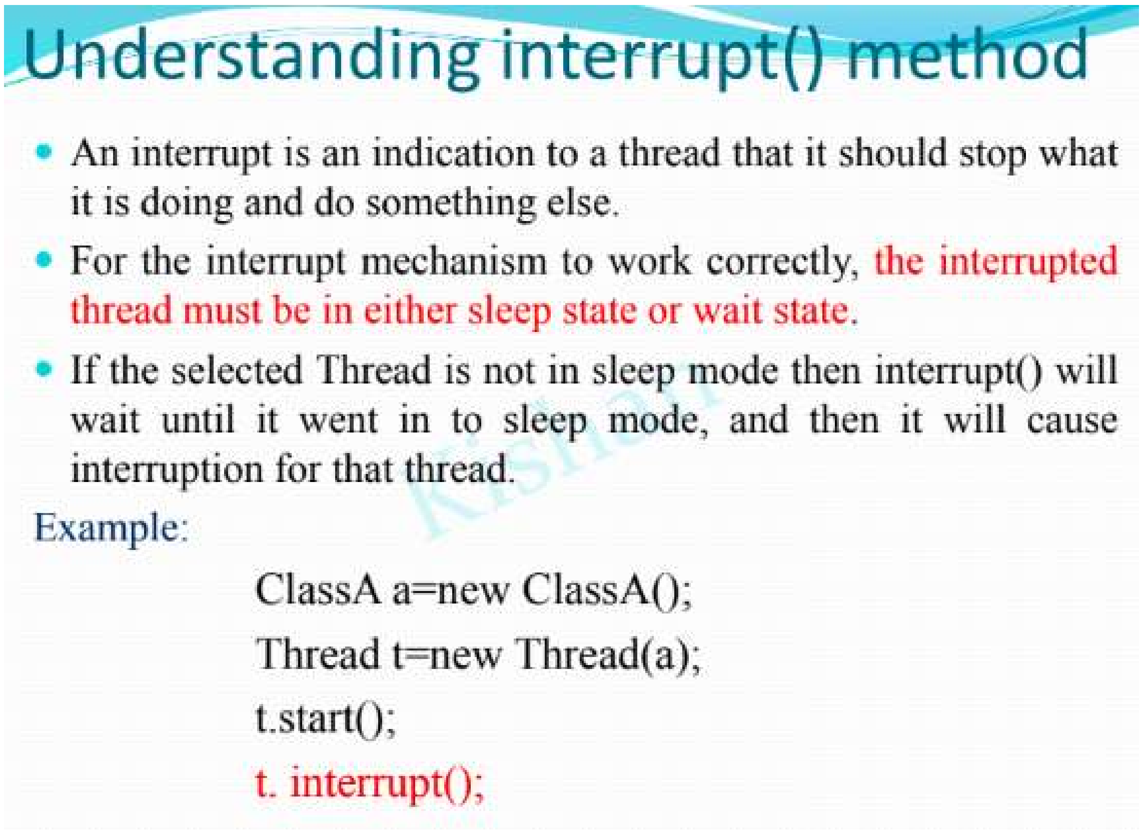
```
package com.pack1;

public class ClassA
{
    public static void main(String[] args)
    {
        System.out.println("J");
        System.out.println("a");
        System.out.println("v");
        System.out.println("a");
    }
}
```

```
J
a
v
a
```



```
package com.pack1;

public class ClassA
{
    public static void main(String[] args) throws InterruptedExce
    {
        System.out.println("J");
        Thread.sleep(5000); // 5sec

        System.out.println("a");
        Thread.sleep(5000);

        System.out.println("v");
        Thread.sleep(5000,500);
        System.out.println("a");
    }
}
```

```
J
a
v
a
```

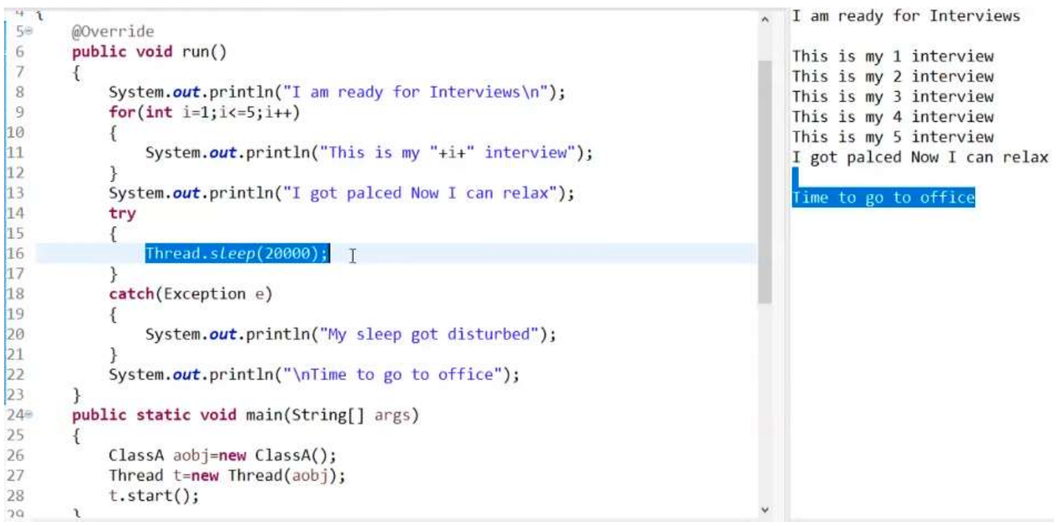


Understanding interrupt() method

- An interrupt is an indication to a thread that it should stop what it is doing and do something else.
- For the interrupt mechanism to work correctly, the interrupted thread must be in either sleep state or wait state.
- If the selected Thread is not in sleep mode then interrupt() will wait until it went in to sleep mode, and then it will cause interruption for that thread.

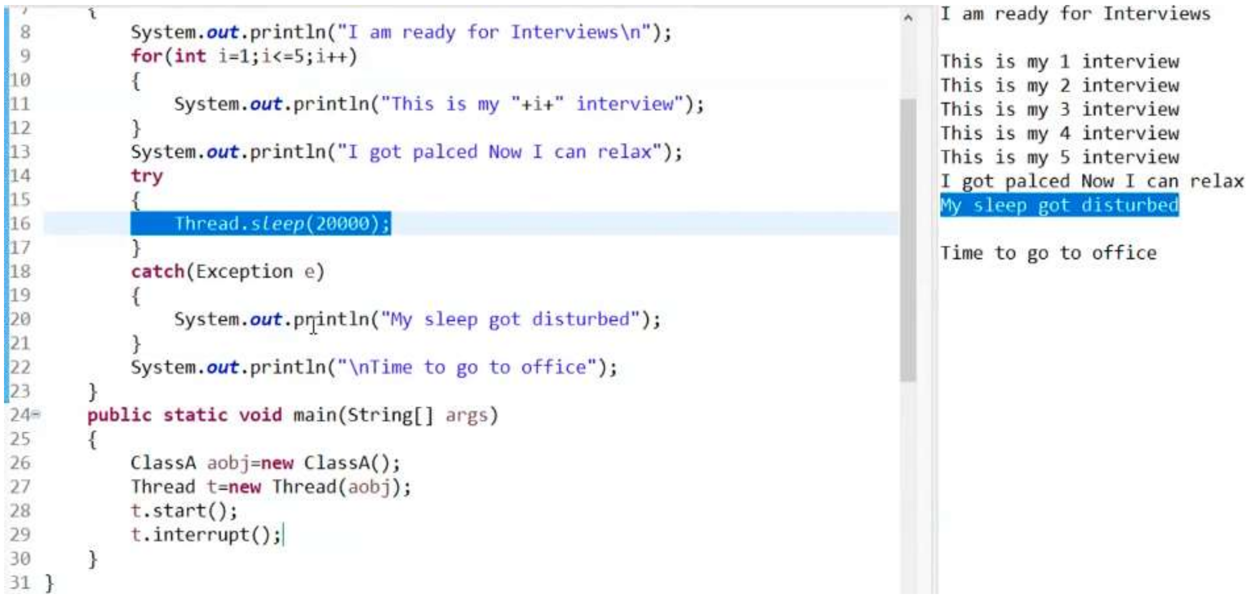
Example:

```
ClassA a=new ClassA();
Thread t=new Thread(a);
t.start();
t. interrupt();
```



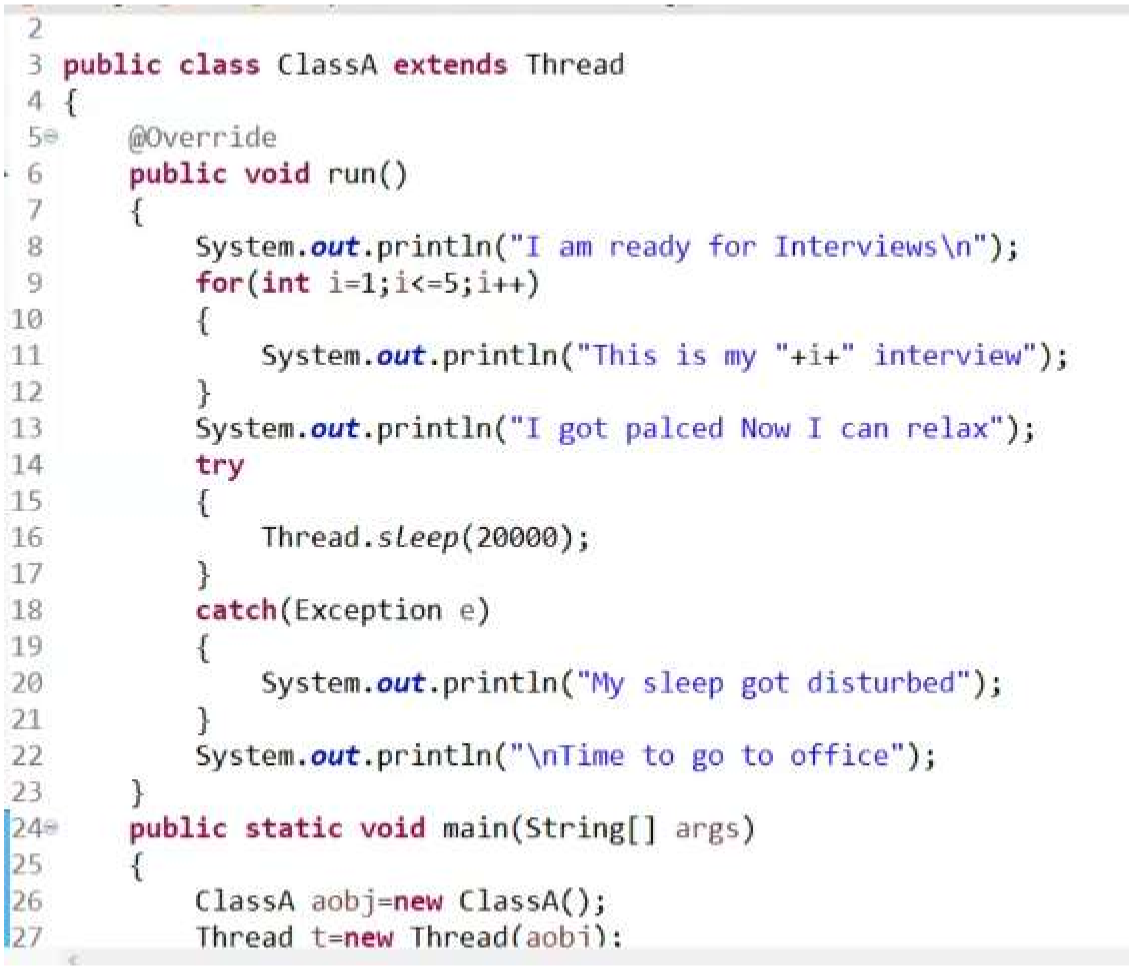
```
    }
    @Override
    public void run()
    {
        System.out.println("I am ready for Interviews\n");
        for(int i=1;i<=5;i++)
        {
            System.out.println("This is my "+i+" interview");
        }
        System.out.println("I got palced Now I can relax");
        try
        {
            Thread.sleep(20000);
        }
        catch(Exception e)
        {
            System.out.println("My sleep got disturbed");
        }
        System.out.println("\nTime to go to office");
    }
    public static void main(String[] args)
    {
        ClassA aobj=new ClassA();
        Thread t=new Thread(aobj);
        t.start();
    }
```

```
I am ready for Interviews

This is my 1 interview
This is my 2 interview
This is my 3 interview
This is my 4 interview
This is my 5 interview
I got palced Now I can relax

Time to go to office
```

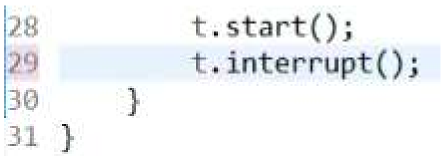


```java
        {
        System.out.println("I am ready for Interviews\n");
        for(int i=1;i<=5;i++)
        {
            System.out.println("This is my "+i+" interview");
        }
        System.out.println("I got palced Now I can relax");
        try
        {
            Thread.sleep(20000);
        }
        catch(Exception e)
        {
            System.out.println("My sleep got disturbed");
        }
        System.out.println("\nTime to go to office");
    }
    public static void main(String[] args)
    {
        ClassA aobj=new ClassA();
        Thread t=new Thread(aobj);
        t.start();
        t.interrupt();
    }
}
```

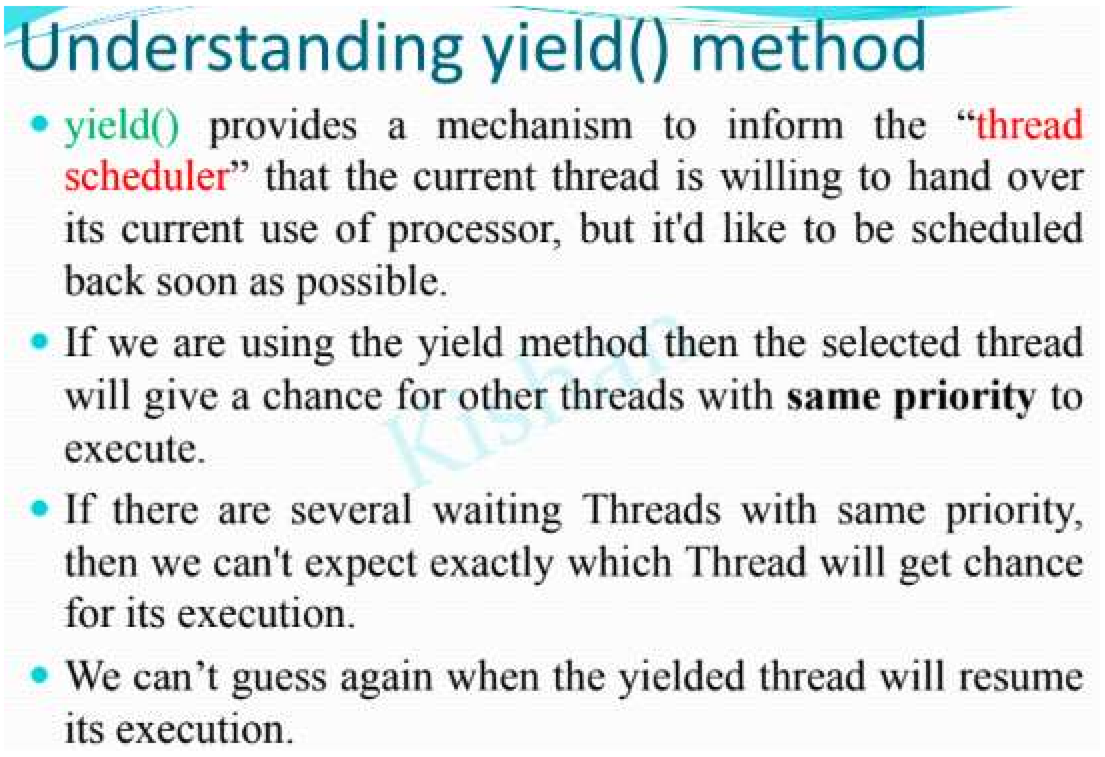
```
I am ready for Interviews

This is my 1 interview
This is my 2 interview
This is my 3 interview
This is my 4 interview
This is my 5 interview
I got palced Now I can relax
My sleep got disturbed

Time to go to office
```



```java
public class ClassA extends Thread
{
    @Override
    public void run()
    {
        System.out.println("I am ready for Interviews\n");
        for(int i=1;i<=5;i++)
        {
            System.out.println("This is my "+i+" interview");
        }
        System.out.println("I got palced Now I can relax");
        try
        {
            Thread.sleep(20000);
        }
        catch(Exception e)
        {
            System.out.println("My sleep got disturbed");
        }
        System.out.println("\nTime to go to office");
    }
    public static void main(String[] args)
    {
        ClassA aobj=new ClassA();
        Thread t=new Thread(aobj);
```



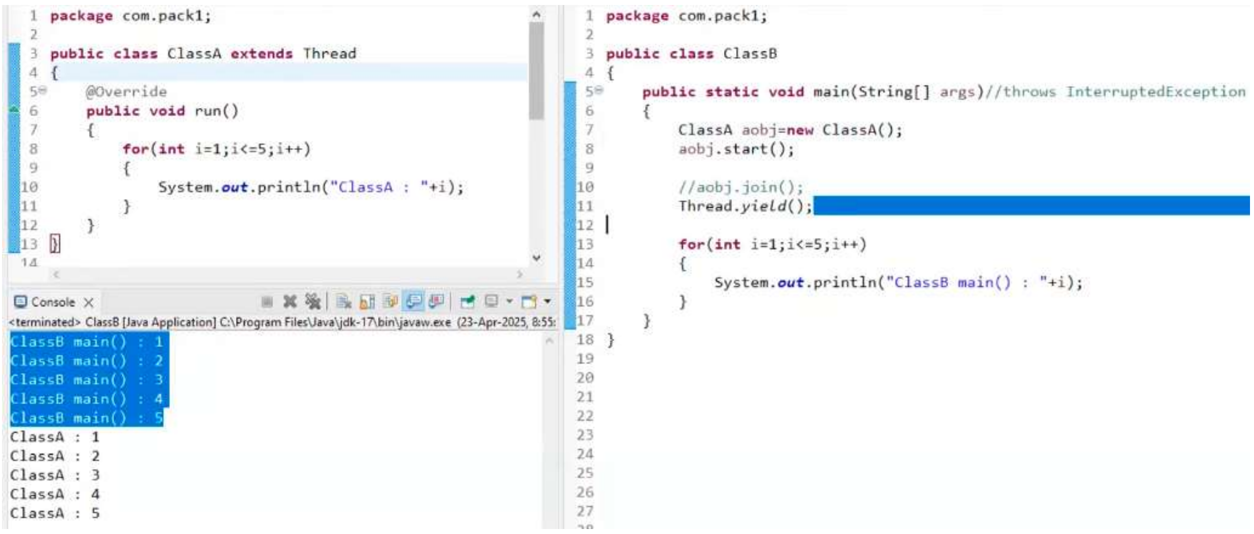
```
        t.start();
        t.interrupt();
    }
}
```



Understanding yield() method

- yield() provides a mechanism to inform the "thread scheduler" that the current thread is willing to hand over its current use of processor, but it'd like to be scheduled back soon as possible.
- If we are using the yield method then the selected thread will give a chance for other threads with **same priority** to execute.
- If there are several waiting Threads with same priority, then we can't expect exactly which Thread will get chance for its execution.
- We can't guess again when the yielded thread will resume its execution.

Yield() provides a mechanism to inform the thread scheduler that the current thread is willing to hand over its current use of processor to the other waiting threads with the same priority but the decision which thread should start it execution will be taken care by thread scheduler.



```java
package com.pack1;

public class ClassA extends Thread
{
    @Override
    public void run()
    {
        for(int i=1;i<=5;i++)
        {
            System.out.println("ClassA : "+i);
        }
    }
}
```

```
<terminated> ClassB [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe (23-Apr-2025, 8:55:
ClassB main() : 1
ClassB main() : 2
ClassB main() : 3
ClassB main() : 4
ClassB main() : 5
ClassA : 1
ClassA : 2
ClassA : 3
ClassA : 4
ClassA : 5
```

```java
package com.pack1;

public class ClassB
{
    public static void main(String[] args)//throws InterruptedException
    {
        ClassA aobj=new ClassA();
        aobj.start();

        //aobj.join();
        Thread.yield();

        for(int i=1;i<=5;i++)
        {
            System.out.println("ClassB main() : "+i);
        }
    }
}
```