## Ways to create threads

1. By extending the thread class
2. By implementing the runnable interface

- Creating a simple thread by extending thread class

```java
package threadsEx;

public class MyThread extends Thread{

    @Override
    public void run() {
        System.out.println("Thread is running!!!!!");
    }

    public static void main(String[] args) {
        MyThread t1 = new MyThread();
        t1.start();
    }
}
```

- Creating simple thread by implementing runnable interface

```java
package runnableEx;
//Implementing runnable interface
public class MyRunnable implements Runnable{

    @Override
    public void run() {
        System.out.println("Thread is Running!!!!!");
    }
}
```

```java
package runnableEx;

public class RunnableExample {
    public static void main(String[] args) {
        //Creating the instance of MyRunnable
        MyRunnable myRunnable = new MyRunnable();

        // Creating the thread object and passing MyRunnable
instance
        Thread thread = new Thread(myRunnable);

        //starting the thread
        thread.start();
    }
}
```

Explanation:

- Created a class MyRunnable which implements Runnable interface
- Overriden the run() method, where we can define what a thread will do.
- In main() method
  We created the instance of MyRunnable

  Passed it to the thread object

  Started the thread with start()
method

## Thread Methods:

1. start(): It begins the execution of the thread
2. run(): It contains the code that a thread executes
3. sleep(milliseconds): pauses the thread for a specific amount of time
4. join(): waits for the thread to complete its execution
5. setName() & getName(): Used to set and get the name of thread
6. getState(): Returns the state of this thread.
7. isAlive(): checks if the thread is still running or not

## New State

```java
package threadsEx;

public class MyThread extends Thread{

    @Override
    public void run() {
        System.out.println("Thread is running!!!!!");
    }

    public static void main(String[] args) {
        MyThread t1 = new MyThread();    //Thread is in new
state
        System.out.println("Thread State: "+ t1.getState());

    }
}
```

o/p:

```
Thread State: NEW
```

## Runnable State

A thread enters the Runnable state when we start the thread( by using start() method).

```java
package threadsEx;

public class MyThread extends Thread{

    @Override
    public void run() {
        System.out.println("Thread is running!!!!!");
    }

    public static void main(String[] args) {
        MyThread t1 = new MyThread();    //Thread is in new
state
        t1.start();                      //Thread moves to the
runnable state
        System.out.println("Thread State: "+ t1.getState());

    }
}
```

o/p:

```
Thread State: RUNNABLE

Thread is running!!!!!
```

## Blocked/Waiting State

A thread enters this state when it is waiting for a resource or another thread to complete.

```java
package blockedExample;

public class SharedResource {
   synchronized void doWork(){
        try {
            Thread.sleep(2000); // simulating some work for
thread
        } catch (InterruptedException e) {
            System.out.println(e);
        }
    }
}
```

- The SharedResource class has a method doWork(), it is synchronized(only one thread can access this method at a time)
- Inside the method, the current running thread sleeps for 2 seconds to simulate some work.

```java
package blockedExample;

public class MyThread extends Thread {
    SharedResource resource;

    MyThread(SharedResource resource){
        this.resource=resource;
    }

    @Override
    public void run() {
        resource.doWork();
    }
}
```

- MyThread extends the thread class
- It takes SharedResource object as a parameter in its constructor
- In run() method, the thread calls the doWork() method on SharedResource object

```java
package blockedExample;

public class Main {
    public static void main(String[] args) {
        SharedResource resource = new SharedResource();
        MyThread t1 = new MyThread(resource);
        MyThread t2 = new MyThread(resource);

        t1.start();
        t2.start();

        try {
            Thread.sleep(100); // waits for thread to start
        } catch (InterruptedException e) {
            System.out.println(e);
        }

        System.out.println("Thread t1 state: "+t1.getState());

        System.out.println("Thread t2 state: "+
t2.getState());
    }
}
```

In the main method

- A ShareResource Object is created
- Two threads t1 and t2 are created and both are sharing same SharedResource object.
- Both threads are started using start() method
- The main thread sleeps for 100 milliseconds to ensure t1 and t2 have started its execution
- We are printing the states of t1 and t2

o/p:

```
Thread t1 state: TIMED_WAITING

Thread t2 state: BLOCKED
```

Synchronization:

It ensures the thread safety

States:

1. TIMED_WAITING: a thread is waiting for specific amount of time(using sleep())

2. BLOCKED: A thread is waiting to acquire a lock held by another thread

t1 -> acquires the lock -> enters doWork() -> sleeps for 2 secs

t2 -> tries to acquire the lock -> lock is held by t1 -> BLOCKED

MAIN THREAD:
Sleeps for 100ms -> Checks the states of t1 and t2 -> prints states

| Terminated State |
|---|

## Complete Example of Thread life cycle

```
package threadLifecycle;

public class MyThread extends  Thread{

    @Override
    public void run() {
        try {
            System.out.println("Thread is running!!!!");
            Thread.sleep(1000); // Timed_Waiting state
        } catch (InterruptedException e) {
            System.out.println(e);
        }
    }
}
```

```
package threadLifecycle;

public class Main {
    public static void main(String[] args) throws
InterruptedException{
        MyThread t1 = new MyThread();
        System.out.println("Thread state after creation: "+
t1.getState());

        t1.start();
        System.out.println("Thread state after start(): "+
t1.getState());

        Thread.sleep(100); // Wait for t1 to enter the
timed_Waiting_state
        System.out.println("Thread state during sleep(): "+
t1.getState());

        t1.join(); //wait for t1 to finish

        System.out.println("Thread State after completion: "+
t1.getState());

    }
}
```

o/p:

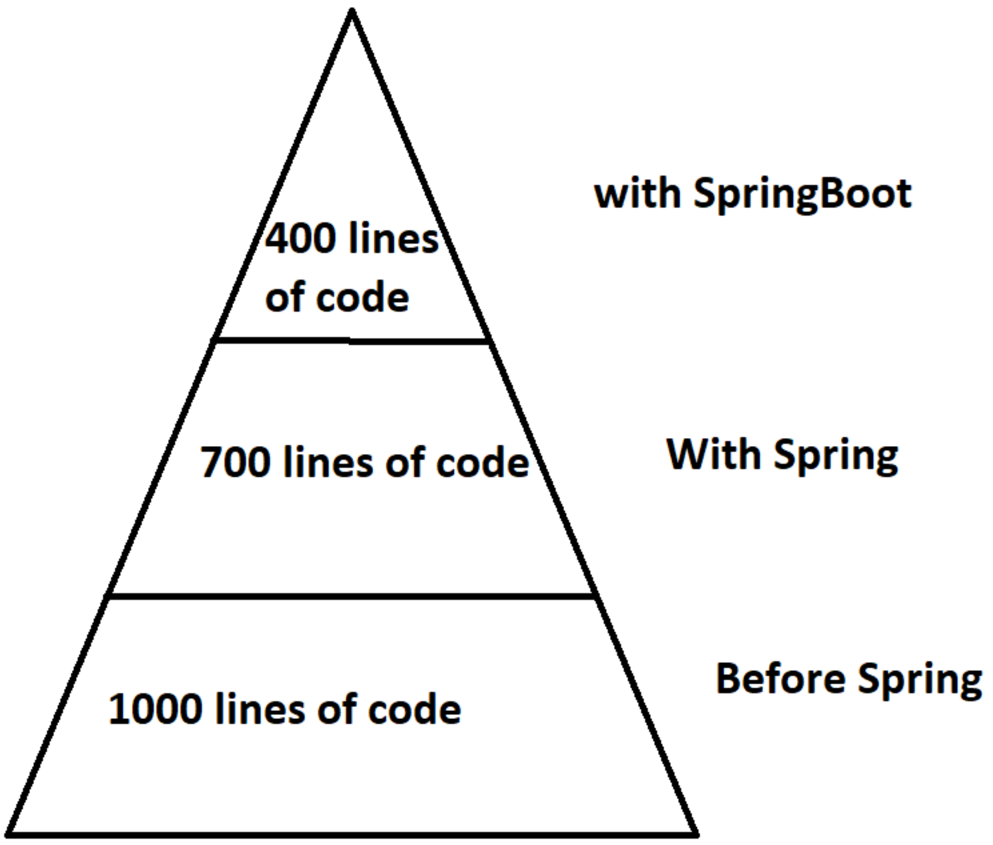
```
Thread state after creation: NEW

Thread state after start(): RUNNABLE

Thread is running!!!!

Thread state during sleep(): TIMED_WAITING

Thread State after completion: TERMINATED
```



SpringBoot

Pre-requisites:

OOP, classes, interfaces, exception handling, collection framework ,inheritance

Purpose: To build java applications

MUST’s:

JDK -> JDK 17 or higher because we are going to use springboot 3.

IntelliJ

The Problem with Spring:

Difference between spring and springboot: