Thread

- A thread is a light weight sub-process which runs independently within a program
- Threads are used for multitasking and for improving performance.

Example: Restaurant Kitchen

How to create threads:

There are 2 ways to create threads:

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

What is thread?

- A thread is smallest unit of process
- It can be executed independently
- It is a lightweight sub-process
- Each thread has its own path of execution
- It means it can perform tasks concurrently with other threads.

Lifecycle of thread:

New: the thread is created but not yet started

Runnable: The thread is ready to run, waiting for CPU to execute it.

Blocked/Waiting: The thread is temporarily inactive, waiting for a resource or another thread to complete its execution.

Timed waiting: The thread is waiting for specific amount of time(using sleep())

Terminated: The thread has completed its execution and is no longer running

There are 2 ways to create threads:

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

- Creating thread by extending thread class.

```java
package threadEx;

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;

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


}





package runnableEx;

public class RunnableExample {
    public static void main(String[] args) {
        //Creating 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();
    }
}
```

o/p:

Thread is running!!!!

Explanation:

- Created a class MyRunnable which implements Runnable interface.
- Overriden a method run(), where we can define what a thread will do.
- In main method
  Created an instance of MyRunnable
  Passed it to 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 which thread is going to execute
3. sleep(milliseconds): pauses the thread for specific amount of time.
4. join(): waits for the thread to complete its execution
5. setName() & getName(): used for set and get the name of thread

6. getState(): returns the state of the thread.
7. isAlive(): to check if the thread is running or not.

Complete example of thread lifecycle:

```java
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 after using start():
"+t1.getState());

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

        t1.join(); // wait for t1 to finish
        System.out.println("Thread state after completion: "+
t1.getState());

    }
}
```

- Blocked/Waiting State

```
• package threadEx;

  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 it at a time)

```
package threadEx;

public class SharedResource {

    synchronized void doWork(){
        try {
            Thread.sleep(2000); //simulating some work for
thread

        } catch (InterruptedException e) {
            System.out.println(e);
        }
```

```java
    }
}


package threadEx;

public class MyThread extends Thread{
    SharedResource resource;

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

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


package threadEx;

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);
        } catch (InterruptedException e) {
            System.out.println(e);
        }

        System.out.println("Thread t1 state: "+
t1.getState());
        System.out.println("Thread t2 state: "+
t2.getState());
    }
}
```

In main method:

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

o/p:

```
Thread t1 state: TIMED_WAITING
Thread t2 state: BLOCKED
```

Synchronization ensures thread safety.

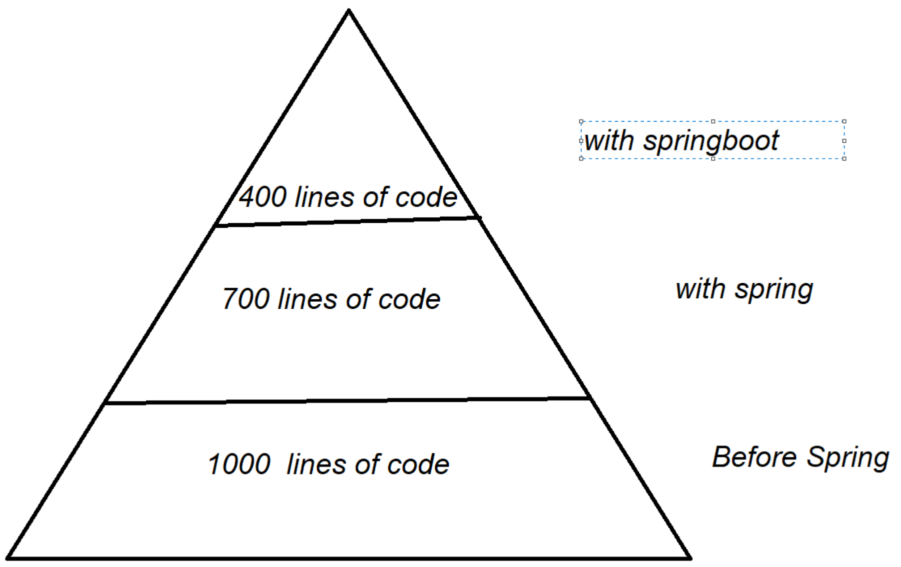
T1 -> acquires the lock ->
sleeps for 2 secs

T2 -> tries to acquire the lock
-> lock is held by t1 ->
BLOCKED

MAIN THREAD:

Sleeps for 100ms -> checks the
states of t1 and t2 -> printing
them.

SpringBoot



Pre-requisites:
OOP, interfaces, collection framework, exception handling, inheritance

Purpose: to build java applications

Must's:

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

IntelliJ

*The Problem with spring: