MulitThreading
   1. Different ways of creating a thread
                a. Extending Thread class
                b. Implements Runnable interface
   2. Setting a name and getting name from the Thread
                a. public void setName(String name)
                b. public String getName()

Lifecycle of a Thread
          new/born -----start()--------> ready/runnable ---ts allocates cpu time---------> running----run() complets---------> deadstate

   3. Methods to prevent a Thread from execution
                a. join() -> To make another thread to wait till it finsihes the execution.
                b. sleep() -> To stop/pause the execution of a thread for sometime
   4. Synchronization
                => this concept is applicable at method level and block level.
                => if we apply synchronization at block level or at method level then only one thread
                        is allowed to execute the block or a method.
                => Advantage -> it resolves the problem of "Data Incosistencey/race condition".
                => DisAdvantage->It increase the waiting time for other threads so it affects the peformance of the
                                                system.

Note: In java we have 2 levels of lock
                a. class level lock => A thread which needs to execute static synchronized block/method needs
                                                        class level lock.
                                                        This lock is very unique at the class level.
                b. object level lock => A thread which needs to execute synchronized block/method needs
                                                         object level lock.
                                                         This lock is very unique at the Object level.

InterThread Communication

      Two threads should interact with each other, how?

eg: Producer Consumer Problem

ProducerEnd
      => Producer duty is to produce the data and once the data is produced update the variable
           called "DataProvider" to true
      => This action should be done by "Producer Thread"

```
for (int i = 1; i <= 10; i++) {
                try {
                      sb.append(i + ": ");
                      Thread.sleep(100);
                      System.out.println("appending");
                } catch (InterruptedException e) {
```

```
                        e.printStackTrace();
                }

}
dataProvider = true;


Consumer End
      => Consumer Thread should consume the data produced by the Producer
      => Consumer  Thread should check the dataprovider status,if it is true
consume the data
           otherwise sleep for some time and again check for the dataprovider
status

while(producer.dataProvider == false) {
                try {
                        Thread.sleep(10);
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
}

//consume the data produced by the producer
System.out.println(producer.sb);

In the above code when the interaction happens b/w 2 threads, always the consumer
thread is ready for
consumption,but the consumer thread will get the data only when the dataProvider
value is set to true.
This increases the waiting time of a thread and makes the cpu time idle,through
which communication b/w
2 threads wont be efficient.
      To reduce the effieciency problem we use the methods given by Object class
a.wait()
b.notify()
c.notifyAll()

Note:
 wait(),notify(),notifyAll() methods are present in Object class not in Thread
class.
 If any thread has to call wait(),notify(),notifyAll() then that thread should be
the owner of the thread.
      We say the thread as owner iff the thread has the lock of the object.
 If the thread calls notify(),notifyAll() and wait() and the thread is not a owner
then it would result in
      RE: "IllegalMonitorStateException".

wait() -> Whichever thread is excpecting the updated result from the object that
thread should call wait method.
               Whenever wait() is called automatically that thread will release the
lock of the Object to the other
               thread to use that lock.

notify()     -> Which ever thread wants to update the Object, that thread should
call notify() to the Other thread(one).
notifyAll() ->Which ever thread wants to update the Object, that thread should call
notifyAll() to the Other
                     waiting threads(many)
```

Note: only wait(),notify(),notifyAll() have the mechanism to release the automatically, where as sleep(),join() these methods
can't release the lock.

FAQ:
Why wait(),notify(),notifyAll() methods are part of Object class, why not Thread class?
ans. These methods will be used by the thread on different types of Objects like StringBuffer,Student,Customer,Account,.....
For every object the parent class is Object,so these methods comes from object class.
=>methods like join,yield,sleep will be applied only on Threads, so only these methods
are part of Thread class not Object class.

What is the difference b/w notify() and notifyAll()?
notify() -> It will give notification only to one thread which needs the lock of that object
notifyAll() -> It will give notification to mulitple threads which needs the lock of that object.

Example of wait() and notify()

```
class Demo extends Thread
{
	//data is updated
	int total = 0;

	public void run(){

		//producer thread
		synchronized(this){
			System.out.println("Child thread starts the calculation");//step-2

			//sum of first 100 numbers
			for (int i =1;i<=100 ; i++)
			{
				total = total + i;
			}
			System.out.println("Child thread is giving the notifcation call");//step-3
			this.notify();

		}
	}
}
class Test
{
	public static void main(String[] args)  throws Exception
	{
		Demo d = new Demo();
		d.start();

		//consumer thread
		synchronized(d){
			System.out.println("Main Thread is calling  wait() method....");//step-1
			d.wait();
```

```
                System.out.println("Main Thread got the notification
call");//step-4
                System.out.println(d.total);//5050
            }
      }
}
```

Output

```
Main Thread is calling  wait() method....
Child thread starts the calculation
Child thread is giving the notifcation call
Main Thread got the notification call
5050
```

Note:

Demo class had total variable

Main thread[5]

- => needs Demo class total variable with proper value(5050)
- => lock is applied on Demo object and call wait()
- => wait() releases the lock of Demo object and main thread enters into sleeping state.

User Defined Thread[5]

- -> should update total variable in Demo class and it should send the notification
- -> notify() is used so the code should be in synchronize region
- -> it needs the lock of Demo object, now the calculation is started to update.
- -> it will give the notification

Producer consumer problem

---

refer : InterThreadCommunicationApp,
InterThreadCommunication.png

Writing the code in lambda Expression style

---

refer: LambdaExpressionApp