```java
package lifecycle;

public class MyThread extends Thread {
    @Override
    public void run() {
        System.out.println("Child thread is running");
        try{
            Thread.sleep(2000);
        } catch (InterruptedException e){
            e.printStackTrace();
        }
        System.out.println("Child thread has finished");
    }
}
```

```java
package lifecycle;

public class ThreadJoinExample {
    public static void main(String[] args) {
        MyThread thread = new MyThread();
        thread.start();

        System.out.println("Main thread is waiting for child
thread to complete its execution");
        try {
            thread.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("Main thread resumes after child
thread's execution");
    }
}
```

Flow:

1. The main thread starts the child thread(thread.start()).

2. The main thread waits at thread.join() until the child thread finishes its execution

3. The child thread prints “Child thread is running”, simulates work by sleeping 2 seconds, and then prints “Child thread has finished”.

4. After the child thread complete execution, the main thread resumes and prints “Main thread resumes after child thread's execution”.

wait() : makes a thread wait until another thread notifies it

notify(): Wakes up one thread that is waiting on the same object’s monitor

notifyAll() : Wakes up all threads waiting on the same object’s monitor

```java
package notifyMethod;

public class SharedResource {
    private boolean available = false;   // Shared resource
state : false means no data produced yet

    // Producer method,synchronized to allow only one thread
to execute at a time
    public synchronized void produce(){
        System.out.println("Producing data.....");
        available=true; // Marking the resource as available
        notify(); // Notify one waiting thread(consumer
thread)
    }

    //Consumer method,synchronized to allow only one thread to
execute at a time
    public synchronized void consume(){
        while(!available){ // check if resource is not
available
            try{
                System.out.println("Waiting for data to be
produced......");
                wait(); // Wait until the producer notifies
            } catch (InterruptedException e) {
               Thread.currentThread().interrupt();
            }
        }
        System.out.println("Consuming data...");
        available = false; // Marking the resource as consumed
    }
}
```

```java
package notifyMethod;

public class NotifyExample {
    public static void main(String[] args) {
        SharedResource resource = new SharedResource();

        // Consumer thread: Calls consume method to wait for
data to be produced
        Thread consumer = new Thread(() ->
resource.consume());

        //Producer thread: Calls produce method to create data
and notify the consumer
        Thread producer = new Thread(() ->
resource.produce());
```

```
        consumer.start();    // Start the consumer thread
        producer.start();    // Start the producer thread
    }
}
```

```java
package notifyAllExample;

public class SharedResource {
    private boolean available = false;


    public synchronized void produce(){
        System.out.println("Producing data.....");
        available = true;
        notifyAll(); // Notify all waiting consumer
threads(consumer)
    }

    public synchronized void consume(int threadId) {
        while (!available) {
            try {
                System.out.println("Thread " + threadId + " is
waiting for data to be produced...");
                wait(); // Wait until notified
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt(); // Handle
thread interruption
            }
        }
        System.out.println("Thread " + threadId + " is
consuming data...");
        available = false; // Mark the resource as consumed
    }

}
```

```java
package notifyAllExample;


public class NotifyAllExample {
    public static void main(String[] args) {
        SharedResource resource = new SharedResource();

        Thread consumer1 = new Thread(() ->
resource.consume(1));
        Thread consumer2 = new Thread(() ->
resource.consume(2));
        Thread producer = new Thread(()-> resource.produce());

        consumer1.start();
        consumer2.start();
        producer.start();
    }
}
```

Explanation:

- Multiple consumer threads (consumer1 and consumer2) wait for the resource to be available.
- The producer thread produces data and calls notifyAll() to wake up all waiting threads.
- Each consumer thread competes to acquire the lock and consume the resource.