INTERTHREAD COMMUNICATION, SUSPEND AND RESUME METHOD

OUR TEAM

- Shweta Rawat - 063
- Siddhi Gupta - 064
- Sommay Shivangi - 065
- Somssi - 066

INTERTHREAD COMMUNICATION

Inter-thread communication is a mechanism in which a thread releases the lock and enter into paused state and another thread acquires the lock and continue to be executed.

There are three methods to implement inter-thread communication, but the thread should be inside synchronised area to call them.

Interthread Communication is implemented by following methods of object class

wait()

Tells the calling thread to give up the monitor and go to sleep until some other thread enters the same monitor and notify or a specified amount of time has elapsed

notify()

Wakes up the first thread that called wait() on the same object

notifyAll()

Wakes up all the threads that called wait() on the same object. The highest priority thread will run first.

EXAMPLE 1

Here interthread communication is not implemented

```
class TotalEarnings extends Thread {
        int total = 0;
        public void run() {
                for (int i = 1; i <=10; i++) {
                total += 100;
                }
        }
}

class movieBookNew {
        public static void main(String[] args) {
                TotalEarnings obj = new TotalEarnings();
                obj.start();
                System.out.println("Without using wait() and notify() methods")
                System.out.println("Total earnings are: "+obj.total+"rs");
        }
}
```

thread T_0

main thread

thread T_0

EXAMPLE 2

Here interthread communication is implemented

```
class totalEarning extends Thread {
        int total = 0;
        public void run() {
                synchronized (this) {
                        for (int i = 1; i <= 10; i++) {
                                total += 100;
                        }
                this.notify();
                }
        }
}

class movieBook {
        public static void main(String[] args) {
                totalEarning obj = new totalEarning();
                obj.start();
                synchronized (obj) {
                        try {
                                obj.wait();
                        } catch (Exception e) {
                                System.out.println(e);
                        }

                        System.out.println("Using wait() and notify() method");
                        System.out.println("Total earnings are: "+obj.total+"rs");
                }
        }
}
```

OUTPUT

Output of example 1

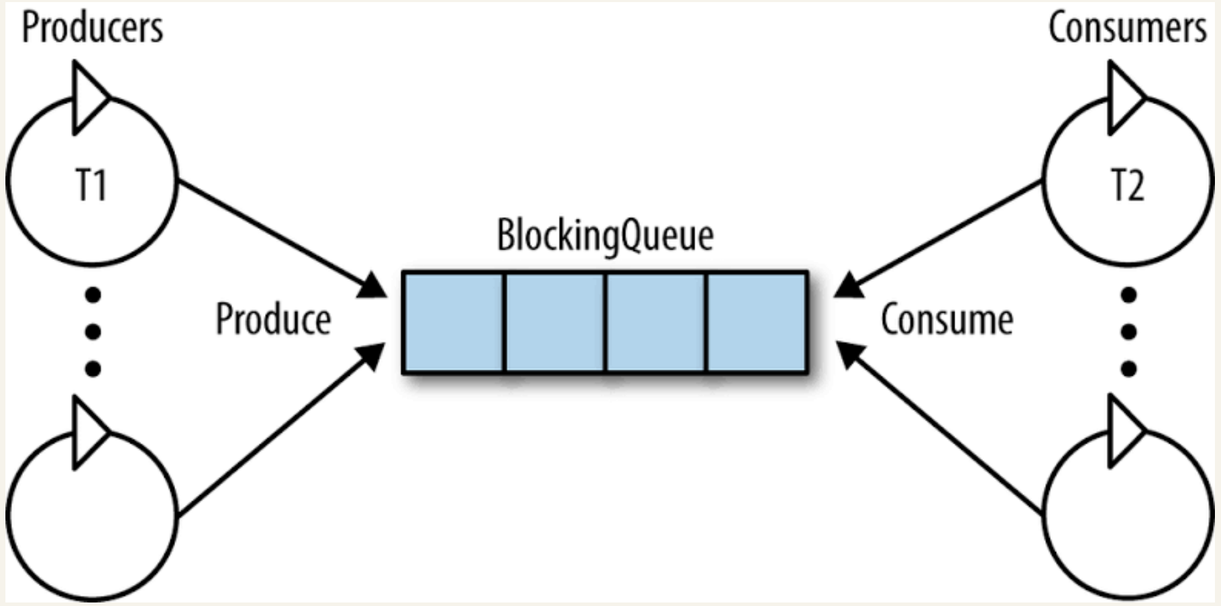
```
C:\Users\somma\OneDrive\Desktop\MCA 2 SEM\java mca 2>java movieBookNew
Without using wait() and notify() methods
Total earnings are: 0rs
```

Output of example 2

```
C:\Users\somma\OneDrive\Desktop\MCA 2 SEM\java mca 2>java movieBook
Using wait() and notify() method
Total earnings are: 1000rs
```

PRODUCER-CONSUMER PROBLEM

1)Producer thread writes to buffer (shared memory)
2)Consumer thread reads from buffer
3)If not synchronised, data may get corrupted:
   a)Producer writes before consumer reads, data overflow and lost
   b)Consumer reads before Producer writes, data doubled



```
class Producer implements Runnable {
  Q q;

  Producer(Q q) {
    this.q = q;
    new Thread(this, "Producer").start();
  }

  public void run() {
    int i = 0;

    while(true) {
      q.put(i++);
```

```
class Consumer implements Runnable {
  Q q;

  Consumer(Q q) {
    this.q = q;
    new Thread(this, "Consumer").start();
  }

  public void run() {
    while(true) {
      q.get();
```

OUTPUT

```
Put: 1
Got: 1
Got: 1
Got: 1
Got: 1
Got: 1
```

SOLUTION?

SYNCHRONISATION....

PRODUCER

If it knows that CONSUMER has not read last data, waits for notify command from consumer

CONSUMER

If it knows that PRODUCER has not updated data, waits for notify command from producer

```
synchronized int get() {
  if(!valueSet)
    try {
      wait();

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

    System.out.println("Got: " + n);
    valueSet = false;
    notify();
    return n;
}

synchronized void put(int n) {
  if(valueSet)
    try {
      wait();
    } catch(InterruptedException e) {
      System.out.println("InterruptedException caught");
    }

    this.n = n;
    valueSet = true;
    System.out.println("Put: " + n);
    notify();
  }
}
```

```
Put: 1
Got: 1
Put: 2
Got: 2
Put: 3
Got: 3
Put: 4
Got: 4
Put: 5
Got: 5
```

SUSPEND() METHOD

The suspend() method of the thread class puts a thread from the running state to a waiting state. This method is employed if you would like to prevent the thread execution and begin it again when a particular event occurs.

MORE ABOUT SUSPEND()

1. Syntax: public final void suspend()
   Return: Does not return any value.

2. Example: A separate thread can display the time of day. If the user doesn't want a clock, its thread can be suspended.

3. The suspend( ) method of the Thread class is deprecated in Java 2.

USE OF SUSPEND() IN EARLIER VERSIONS

IMPLEMENTATION

```
class MyThread implements Runnable {
  public void run() {
    try {
      Thread.sleep(500);
      System.out.println(Thread.currentThread().getName());
    } catch (Exception e) {

    }
  }
}

public class Main {
  public static void main(String[] args) {
    Thread t1 = new Thread(new MyThread());
    Thread t2 = new Thread(new MyThread());
    Thread t3 = new Thread(new MyThread());

    t1.start();
    t2.start();
    t2.suspend();
    t3.start();
    t2.resume();
  }
}
```

OUTPUT

```
Thread-0
Thread-1
Thread-2
```

WHY DEPRECATED?

- **Reason**

  This was done because suspend( ) can sometimes cause serious system failures.

- **Example**

  Assume that a thread has obtained locks on critical data structures. If that thread is suspended at that point, those locks are not relinquished. Other threads that may be waiting for those resources can be deadlocked.

SOLUTION IN JAVA2

A thread must be designed so that the run( ) method periodically checks to determine whether that thread should suspend, resume, or stop its execution. Typically, this is accomplished by establishing a flag variable that indicates the execution state of the thread.

USE OF SUSPEND() IN JAVA2

Adding a Boolean variable suspendFlag

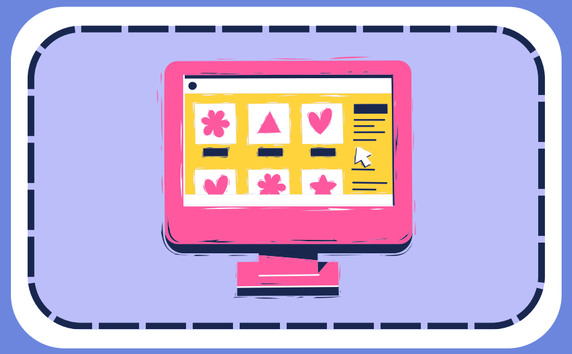
```
class NewThread implements Runnable {
    String name; // name of thread
    Thread t;
    boolean suspendFlag;
}
```

Creating functions for suspend and resume

```
void mysuspend() {
    suspendFlag = true;
}
synchronized void myresume() {
    suspendFlag = false;
    notify();
}
```

Making sure the run function keeps checking the status of the thread

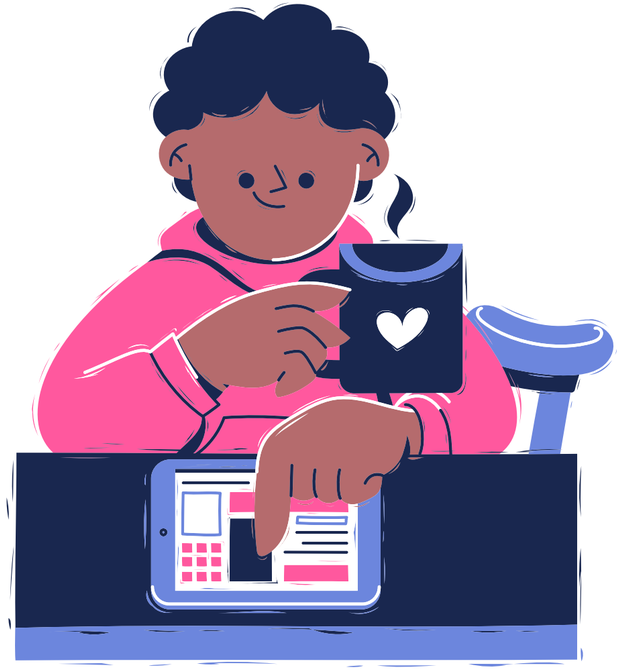
```
public void run() {
    try {
      for(int i = 15; i > 0; i--) {
        System.out.println(name + ": " + i);
        Thread.sleep(200);
        synchronized(this) {
          while(suspendFlag) {
            wait();
          }
        }
      }
    } catch (InterruptedException e) {
      System.out.println(name + " interrupted.");
    }
    System.out.println(name + " exiting.");
  }
```



resume() method in Java

-Shweta Rawat 63

Presentation Outline



- Introduction
- Why was resume() used for?
- Pitfalls of resume()
- Modern approaches

Think About...

What does it mean when you say resume something?

Why was resume() method used for?

Syntax:

public final void resume()

```
class NewThread implements Runnable {
    String name; // name of thread
    Thread t;

    NewThread(String threadname) {
        name = threadname;
        t = new Thread(this, name);
        System.out.println("New thread: " + t);
        t.start(); // Start the thread
    }
```

```
// This is the entry point for thread.
public void run() {
    try {
        for (int i = 15; i > 0; i--) {
            System.out.println(name + ": " +
                );
            Thread.sleep(200);
        }
    } catch (InterruptedException e) {
        System.out.println(name + "
            interrupted.");
    }
    System.out.println(name + " exiting.");
}
```

```
public static void main(String args[]) {
    NewThread ob1 = new NewThread("One");
    NewThread ob2 = new NewThread("Two");
    try {
        Thread.sleep(1000);
        ob1.t.suspend();
        System.out.println("Suspending thread One");
        Thread.sleep(1000);
        ob1.t.resume();
        System.out.println("Resuming thread One");
        ob2.t.suspend();
        System.out.println("Suspending thread Two");
        Thread.sleep(1000);
        ob2.t.resume();
        System.out.println("Resuming thread Two");
    } catch (InterruptedException e) {
        System.out.println("Main thread Interrupted");
    }
```

```
        // wait for threads to finish
        try {
            System.out.println("Waiting for threads to
                finish.");
            ob1.t.join();
            ob2.t.join();
        } catch (InterruptedException e) {
            System.out.println("Main thread Interrupted");
        }
        System.out.println("Main thread exiting.");
    }
}
```

```
Note: /tmp/9a9W6G64B0/NewThread.java uses or overrides a
    deprecated API.
Note: Recompile with -Xlint:deprecation for details.
java -cp /tmp/9a9W6G64B0/NewThread
New thread: Thread[One,5,main]
New thread: Thread[Two,5,main]
Two: 15
One: 15
Two: 14
One: 14
Two: 13
One: 13
Two: 12
One: 12
Two: 11
One: 11
Suspending thread One
```

More on this later

```
Two: 10
Two: 9
Two: 8
Two: 7
Two: 6
Resuming thread One
Suspending thread Two
One: 10
One: 9
One: 8
One: 7
One: 6
```

```
Resuming thread Two
Waiting for threads to finish.
Two: 5
One: 5
Two: 4
One: 4
Two: 3
One: 3
Two: 2
One: 2
Two: 1
One: 1
One exiting.
Two exiting.
Main thread exiting.

=== Code Execution Successful ===
```

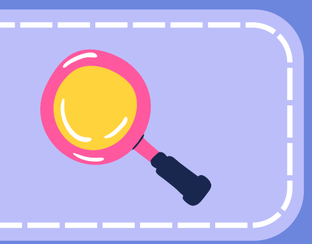
The resume() method in Java used in conjunction with the suspend() method.

Purpose:

The resume() method is employed to resume a thread that was previously suspended using the suspend() method.

Functionality:

When a thread is suspended, it enters a waiting state. The resume() method allows the suspended thread to start running again.



resume( ) was deprecated

BUT WHY?





suspend( ) can sometimes cause serious system failures.
Assume that a thread has obtained locks on critical data structures. If that thread is suspended at that point, those locks are not relinquished. Other threads that may be waiting for those resources can be deadlocked.





The resume( ) method is also deprecated. It does not cause problems, but cannot be used without the suspend( ) method as its counterpart.





Lack of Predictability: The behavior of suspend() and resume() is unpredictable. Resuming a thread might happen at an arbitrary point in its execution, leading to unexpected results.





Modern methods in use:

Wait and Notify Mechanism

- Use the wait() and notify() methods for thread synchronization.
- These methods allow threads to wait for specific conditions and notify other threads when those conditions are met.
- By using wait() and notify(), you can achieve better coordination between threads without directly manipulating their execution state

Thank you!