**//Example 4: Demonstration\_4.java**

import java.util.Scanner;

// PC (Produce Consumer) class with produce() and

// consume() methods.

public static class PC

{

// Prints a string and waits for consume()

public void produce()throws InterruptedException

{

// synchronized block ensures only one thread

// running at a time.

synchronized(this)

{

System.out.println("producer thread running");

// releases the lock on shared resource

**wait();**

// and waits till some other method invokes notify().

System.out.println("Resumed");

}

}

// Sleeps for some time and waits for a key press. After key

// is pressed, it notifies produce().

public void consume()throws InterruptedException

{

// this makes the produce thread to run first.

Thread.sleep(1000);

Scanner s = new Scanner(System.in);

// synchronized block ensures only one thread

// running at a time.

synchronized(this)

{

System.out.println("Waiting for return key.");

s.nextLine();

System.out.println("Return key pressed");

// notifies the produce thread that it

// can wake up.

**notify();**

// Sleep

Thread.sleep(2000);

}

}

**}**

**}**

public class **Demonstration\_4**

{

public static void main(String[] args)

throws InterruptedException

{

final PC pc = new PC();

// Create a thread object that calls pc.produce()

Thread t1 = new Thread(new Runnable()

{

@Override

public void run()

{

try

{

pc.produce();

}

catch(InterruptedException e)

{

e.printStackTrace();

}

}

});

// Create another thread object that calls

// pc.consume()

Thread t2 = new Thread(new Runnable()

{

@Override

public void run()

{

try

{

pc.consume();

}

catch(InterruptedException e)

{

e.printStackTrace();

}

}

});

// Start both threads

t1.start();

t2.start();

// t1 finishes before t2

t1.join();

t2.join();

}

**Output:**

producer thread running

Waiting for return key.

Return key pressed

Resumed

**//Example 5: Demonstration\_5.java**

class Notify1 extends Thread

{

public void run()

{

synchronized(this)

{

System.out.println("Starting of " + Thread.currentThread().getName());

try {

this.wait();

}

catch (InterruptedException e) {

e.printStackTrace();}

System.out.println(Thread.currentThread().getName() + "...notified");

}

}

}

class Notify2 extends Thread {

Notify1 notify1;

Notify2(Notify1 notify1)

{

this.notify1 = notify1;

}

public void run()

{

synchronized(this.notify1)

{

System.out.println("Starting of " + Thread.currentThread().getName());

try {

**this.notify1.wait();**

}

catch (InterruptedException e) {

e.printStackTrace();

}

System.out.println(Thread.currentThread().getName() + "...notified");

}

}

}

class Notify3 extends Thread

{

Notify1 notify1;

Notify3(Notify1 notify1)

{

this.notify1 = notify1;

}

public void run()

{

synchronized(this.notify1)

{

System.out.println("Starting of " + Thread.currentThread().getName());

// call the notifyAll() method

**this.notify1.notifyAll();**

System.out.println(Thread.currentThread().getName() + "...notified");

}

}

}

public class **Demonstration\_5**

{

public static void main(String[] args) throws InterruptedException

{

Notify1 notify1 = new Notify1();

Notify2 notify2 = new Notify2(notify1);

Notify3 notify3 = new Notify3(notify1);

// creating the threads

Thread t1 = new Thread(notify1, "Thread-1");

Thread t2 = new Thread(notify2, "Thread-2");

Thread t3 = new Thread(notify3, "Thread-3");

// call run() method

t1.start();

t2.start();

Thread.sleep(100);

t3.start();

}

}

**Output:**

Starting of Thread-1

Starting of Thread-2

Starting of Thread-3

Thread-3...notified

Thread-1...notified

Thread-2...notified