INTER-THREAD COMMUNICATION

Inter-process communication (IPC) is a mechanism that allows the exchange of data between processes. By providing a user with a set of programming interfaces, IPC helps a programmer organize the activities among different processes. IPC allows one application to control another application, thereby enabling data sharing without interference.

IPC enables data communication by allowing processes to use segments, semaphores, and other methods to share memory and information. IPC facilitates efficient message transfer between processes. The idea of IPC is based on Task Control Architecture (TCA). It is a flexible technique that can send and receive variable length arrays, data structures, and lists. It has the capability of using publish/subscribe and client/server data-transfer paradigms while supporting a wide range of operating systems and languages.

Inter-thread communication or Co-operation is all about allowing synchronized threads to communicate with each other. Interthread communication is important when you develop an application where two or more threads exchange some information.

Cooperation (Inter-thread communication) is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed. It is implemented by following methods of Object class:

• wait()

• notify()

• notifyAll()

All these methods belong to object class as final so that all classes have them. They must be used within a synchronized block only.

1) wait() method

Causes current thread to release the lock and wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed. The current thread must own this object’s monitor, so it must be called from the synchronized method only otherwise it will throw exception.

2) notify() method

Wakes up a single thread that is waiting on this object's monitor. If any threads are waiting on this object, one of them is chosen to be awakened. The choice is arbitrary and occurs at the discretion of the implementation. Syntax: public final void notify()

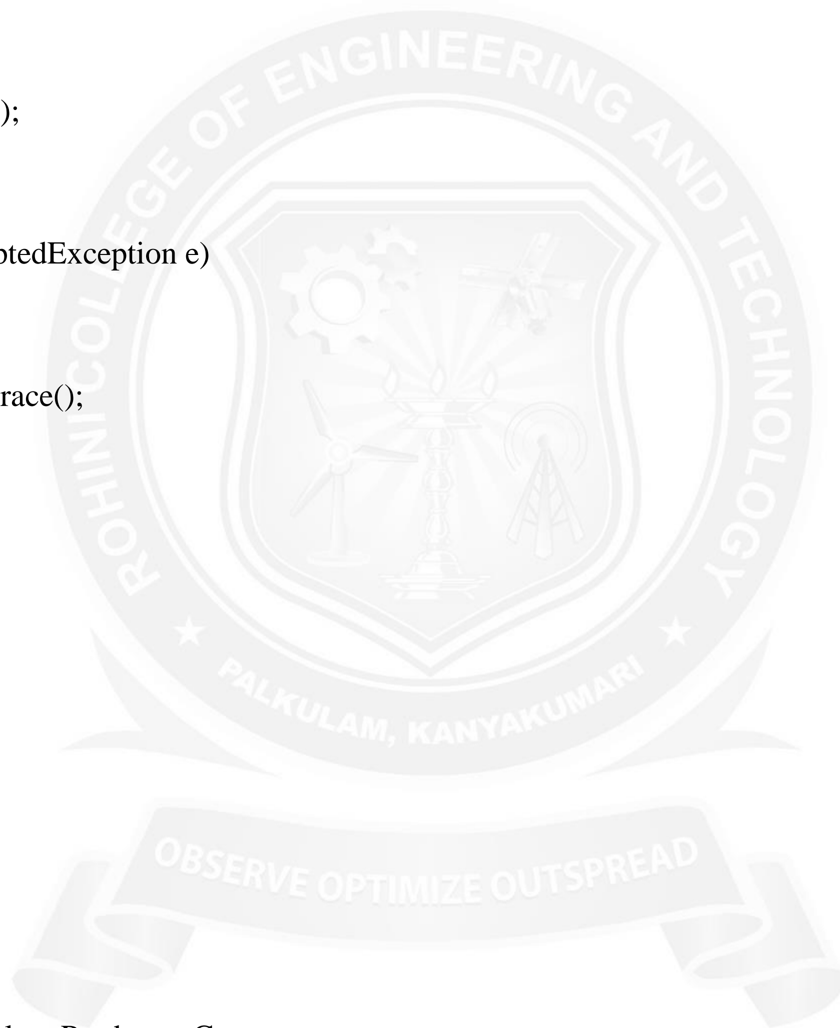
3) notifyAll() method

Wakes up all threads that are waiting on this object's monitor. Syntax:

public final void notifyAll()

// Java program to demonstrate inter-thread communication (wait(), join() and notify()) in Java

```
import java.util.Scanner;
public class Thread_Example
{
public static void main(String[] args) throws InterruptedException
{
final Producer_Consumer pc = new Producer_Consumer ();
Thread t1 = new Thread(new Runnable()
{
public void run()
{
try
{
pc.producer();
}
catch(InterruptedException e)
{
e.printStackTrace();
}
}
```



```
});
Thread t2 = new Thread(new Runnable()
{
public void run()
{
try
{
pc.consumer();
}
catch(InterruptedException e)
{
e.printStackTrace();
}
}
});
t1.start();
t2.start();
t1.join();
t2.join();
}
public static class Producer_Consumer
{
public void producer()throws InterruptedException
{
synchronized(this)
{
```

```
System.out.println("producer thread running");
wait();
System.out.println("Resumed");
}
}
public void consumer()throws InterruptedException
{
Thread.sleep(1000);
Scanner ip = new Scanner(System.in);
synchronized(this)
{
System.out.println("Waiting for return key.");
ip.nextLine();
System.out.println("Return key pressed");
notify();
Thread.sleep(1000);
}
}
}
}
```

The following statements explain how the above producer-Consumer program works.

• The use of synchronized block ensures that only one thread at a time runs. Also since there is a sleep method just at the beginning of consumer loop, the produce thread gets a kickstart.

• When the wait is called in producer method, it does two things.

1. It releases the lock it holds on PC object.

2. It makes the produce thread to go on a waiting state until all other threads have terminated, that is it can again acquire a lock on PC object and some other

method wakes it up by invoking notify or notifyAll on the same object.

• Therefore we see that as soon as wait is called, the control transfers to consume thread and it prints -"Waiting for return key".

• After we press the return key, consume method invokes notify(). It also does 2 things- Firstly, unlike wait(), it does not releases the lock on shared resource therefore for getting the desired result, it is advised to use notify only at the end of your method.

Secondly, it notifies the waiting threads that now they can wake up but only after the current method terminates.

• As you might have observed that even after notifying, the control does not immediately passes over to the produce thread. The reason for it being that we have called Thread. sleep() after notify(). As we already know that the consume thread is holding a lock on PC object, another thread cannot access it until it has released the lock. Hence only after the consume thread finishes its sleep time and thereafter terminates by itself, the produce thread cannot take back the control.

• After a 2 second pause, the program terminates to its completion.

• The following program is one more example for interthread communication

```
class InterThread_Example
{
public static void main(String arg[])
{
final Client c = new Client();
new Thread()
{
public void run()
{
c.withdraw(15000);
}
}.start();
```

```
new Thread()
{
public void run()
{
c.deposit(10000);
}
}.start();
new Thread()
{
public void run()
{
c.deposit(10000);
}
}.start();
}
}
class Client
{
int amount = 10000;
synchronized void withdraw(int amount)
{
System.out.println("Available Balance " + this. amount);
System.out.println("withdrawal amount." + amount);
if (this.amount < amount)
{
System.out.println("Insufficient Balance waiting for deposit.");
```

```
try
{
wait();
} catch (Exception e)
{
System.out.println("Interruption Occured");
}
}
this.amount -= amount;
System.out.println("Detected amount: " + amount);
System.out.println("Balance amount : " + this.amount);
}
synchronized void deposit(int amount)
{
System.out.println("Going to deposit " + amount);
this.amount += amount;
System.out.println("Available Balance " + this.amount);
System.out.println("Transaction completed.\n");
notify();
}
}
```