Application Program Development

Segment : Thread Synchronization

Presenter Name

Outcomes

- Understanding of Thread Communication
- Understanding of Why Communication is needed.
- Implementation of Thread Communication.
  - Wait.
  - Notify.
  - NotifyAll.
  - Interupt
- Thread Communication using Producer and Consumer.

Java Monitors

- Thread Communication was achieved prior to Java 5 by programming object’s built-in monitors.
- Locks and conditions are more powerful and flexible than the built-in monitor.
- A *monitor* is an object with mutual exclusion and synchronization capabilities.
- Only one thread can execute a method at a time in the monitor.
- A thread enters the monitor by acquiring a lock on the monitor and exits by releasing the lock.
- *Any object can be a monitor*.
- An object becomes a monitor once a thread locks it.
- Locking is implemented using the synchronized keyword on a method or a block.
- A thread must acquire a lock before executing a synchronized method or block.
- A thread can wait in a monitor if the condition is not right for it to continue executing in the monitor.

Guarded Blocks

- Threads have to coordinate their actions (they must work together).
- The *guarded block* is the most common coordination idiom for threads coordination.
- The guarded block uses three methods from **Object**class:
- **wait()**
  - Causes the current thread to wait until another thread invokes the **notify()**method or the **notifyAll()**method for this object.
- **notify()**
  - Wakes up a single thread that is waiting on this object's monito
- **notifyAll()**
  - Wakes up all threads that are waiting on this object's monitor.

wait(), notify(), notifyAll()

- **wait()**
  - Makes a thread to wait until some conditions are satisfied. Places the invoking thread on the monitor's waiting list.

- **notify()/notifyAll()**
  - Tells waiting thread/s that something has occurred that might satisfy that condition.
  - Reactivates one/all threads in monitor's waiting list.

Understanding the process of inter-thread communication

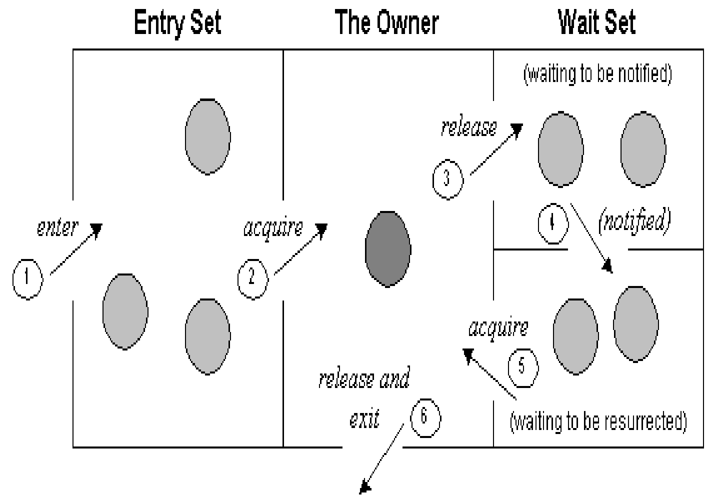
1. Threads enter to acquire lock.
2. Lock is acquired by on thread.
3. Now thread goes to waiting state if you call wait() method on the object.

- Otherwise it releases the lock and exits.

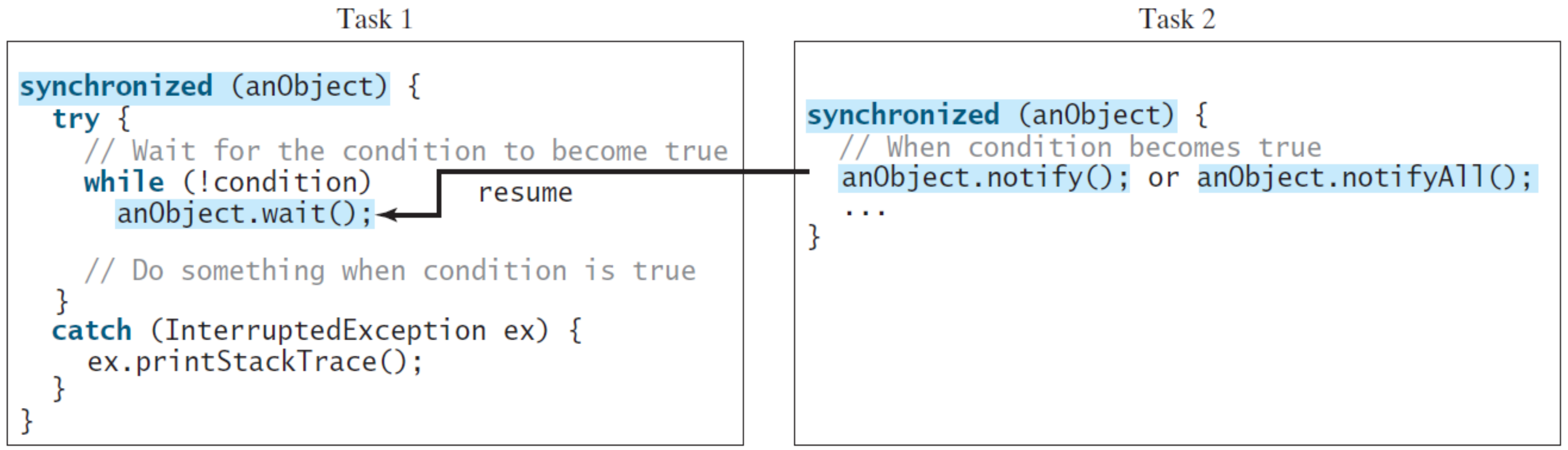
4. If you call notify() or notifyAll() method, thread moves to the notified state

- (runnable state).

5. Now thread is available to acquire lock.
6. After completion of the task, thread releases the lock and exits the monitor state of the object.



Example: Using Monitor



```
synchronized (anObject) {
  try {
    // Wait for the condition to become true
    while (!condition)
      anObject.wait();

    // Do something when condition is true
  }
  catch (InterruptedException ex) {
    ex.printStackTrace();
  }
}
```

```
synchronized (anObject) {
  // When condition becomes true
  anObject.notify(); or anObject.notifyAll();
  ...
}
```

- The wait(), notify(), and notifyAll() methods must be called in a synchronized method or a synchronized block on the receiving object of these methods. Otherwise, an IllegalMonitorStateException will occur.
- When wait() is invoked, it pauses the thread and simultaneously releases the lock on the object. When the thread is restarted after being notified, the lock is automatically reacquired.
- The wait(), notify(), and notifyAll() methods on an object are analogous to the await(), signal(), and signalAll() methods on a condition.

```java
public class ThreadCommunication {
   int amount = 10000;

  synchronized void withdraw(int amount) {
     System.out.println("Going to withdraw");
  if(this.amount < amount) {
     System.out.println("Less Balance, waiting for deposit");
  try {
    wait();
  }catch(Exception e) {System.out.print(e);}
 }
     this.amount -= amount;
     System.out.println("withdrawl is completed...");
}

 synchronized void deposit(int amount) {
   System.out.println("going to deposit...");
   this.amount += amount;
   System.out.println("depoist completed...");
   notify();
 }
}
```

```
public class TestCommunication{

public static void main(String args[]){

     final ThreadCommunication c = new
ThreadCommunication();

     new Thread(){
          public void run(){c.withdraw(15000);}
     }.start();

     new Thread(){
          public void run(){c.deposit(10000);}
     }.start();
     }
}
```

```
Output:
going to withdraw...
Less balance; waiting for deposit...
going to deposit...
deposit completed...
withdraw completed
```

wait()Method Idiom

- When **wait** is invoked, the thread releases the lock and suspends execution

```
 public synchronized void guardedExamResult() {

     // This guard only loops once for each special event,
     // which may not be the event we're waiting for.

    while (!examResult) {  try {
            wait();
        } catch (InterruptedException e) {}
     }

    System.out.println("Exam Result have been received!");
 }
```

Important note:

Always invoke **wait** inside a loop that tests for the condition being waited for.

notifyAll()Method Idiom

- When **`notifyAll`** is invoked, it informs all threads waiting on a lock that something important has happened

```
public synchronized notifyExamResult() {
    examResult = true;
    notifyAll();
}
```

Important note:

There is a second notification method, **notify**, which wakes up a single thread. The **notify** method doesn't allow you to specify the thread that is woken up.