Notify versus NotifyAll

We give an example to show the difference between using notify() and notifyAll(). Put these four classes into Eclipse or DrJava and play with them yourself. The example is based on one found in stackoverflow.com/questions/37026/java-notify-vs-notifyall-all-over-again.

Class Dropbox to the right is a conventional bounded buffer simplified by having the buffer size be 1 and “complified” by requiring a consumer to ask for either an even or an odd integer —similar to a bakery customer asking for white bread or dark bread. In addition, methods take and put print out info about what they are doing. Take the time to study this class.

Directly below are Runnable classes Producer and Consumer, both of which use a Dropbox object for a 1-element buffer.

Producer repeats two steps forever: (1) sleep for a random amount of time and (2) put a random integer into the buffer.

Consumer repeats two steps forever: (1) take an integer from the buffer and (2) sleep for a random amount of time. But class Consumer is special. An object of Consumer takes only even integers or only odd integers, depending on how it was constructed.

Please study both classes Producer and Consumer. On the next page, we have a class that puts this all together with a method main that creates producer and consumer threads and starts them running.

```
/** An instance is a bounded buffer of size 1.
  * Restriction: a consumer must ask for
  * either an even integer or an odd integer. */
class Dropbox {
  private boolean full= false;  // = "buffer is full"
  private int num;      // buffer value (if buffer is full)

  /** Wait for buffer to hold an even num (if e is true) or
    * an odd num (if e is false); then take and return it. */
  public synchronized int take(boolean e) {
    String s= e ? "Even" : "Odd";
    while (!full ||  e == (num % 2 != 0)) {
      try {
        System.out.println("wait for: " + s);
        wait();
      } catch (InterruptedException ex) { }
    }
    System.out.println(s + " took " + num);
    full= false;  notifyAll();  return num;
  }

  /** Wait for buffer to be empty, then put n into it. */
  public synchronized void put(int n) {
    while (full) {
      try {wait();}
       catch (InterruptedException e) {}
    }
    System.out.println("Producer put in " + n);
    num= n;  full= true;  notifyAll();
   }
}
```

```
import java.util.Random;

/** An instance alternately sleeps and puts
 *  and a random integer into into a Dropbox. */
public class Producer implements Runnable {
  private Dropbox dropbox; // The 1-item buffer

  /** Constructor: an instance using Dropbox d. */
  public Producer(Dropbox d) { dropbox= d; }

  /** Forever: sleep for a random time in 0..99 and
    * put a random integer in 0..9 into the buffer. */
  public void run() {
    Random random= new Random();
    while (true) {
      int n= random.nextInt(10);
      try {
        Thread.sleep(random.nextInt(100));
        dropbox.put(n);
      } catch (InterruptedException e) { }
     }}}
```

```
import java.util.Random;

/** An instance alternately consumes either even or odd
  * integers (but not both) from a Dropbox and sleeps */
public class Consumer implements Runnable {
  private Dropbox dropbox; // The 1-item buffer
  private boolean even;  // Take even values iff even

  /** Constructor: An instance taking values
    * from d ---even values iff e is true. */
  public Consumer(boolean e, Dropbox d) {
    even= e;  dropbox= d;
  }

  /** Forever: Get value of right kind from buffer
    * and sleep for a random amount of time. */
  public void run() {
    Random random= new Random();
    while (true) {
      dropbox.take(even);
      try {
        Thread.sleep(random.nextInt(100));
      } catch (InterruptedException e) { }
     }}}
```

Notify versus NotifyAll

To the right is short application. Read the specification of method main.

```
/** Example of diff between notify() and notifyAll(). */
public class PCDropBox {
   /** Create and start a Consumer thread that gets even
     * integers, a Consumer thread that gets odd integers,
     * and a Producer of integers, all using the same
     * Dropbox. */
   public static void main(String[] args) {
      Dropbox db= new Dropbox();
      (new Thread(new Consumer(true, db))).start();
      (new Thread(new Consumer(false, db))).start();
      (new Thread(new Producer(db))).start();
   }
}
```

Below is the start of one execution of this application, showing only the beginning of the output, which goes on forever. It works well, always responding to consumers and the producer in a reasonable way.

```
wait for: Odd
wait for: Even
Producer put in 7
wait for: Even
Odd took 7
wait for: Even
Producer put in 1
Odd took 1
wait for: Even
Producer put in 7
wait for: Even
Odd took 7
wait for: Even
Producer put in 0
Even took 0
```

Switch from notifyAll() to notify()

Now change the last statement notifyAll() in method put of class Dropbox to notify() and run the application again. The very first time we did this we got the output shown to the right. It hung! No more progress can be made. Here's what happened:

```
wait for: Odd
wait for: Even
Producer put in 8
wait for: Odd
```

(1) The Odd and Even Consumers called method take and were put on the wait list because the Dropbox was empty.

(2) The Producer put in the even number 8 and called notify(). Randomly, the Odd Consumer was notified; the Odd Consumer got the lock and was put on the wait list because the Dropbox number is even. Both consumers are now on the wait list and cannot do anything until notified. The Producer attempts to put another value into the Dropbox, but the Dropbox is full, so it also ends up on the wait list. All three threads are on the wait list, forever.

Try running the application as often as you like. The output may be longer, but at some point all three threads will be on the wait list and wait forever..

Method notifyAll() is much less efficient that notify(), because *all* threads on the wait list are placed in the list of threads wanting the lock. But at least it works! *Always* use notifyAll(), except when you know enough to understand that notify() will work —this is not always easy. It *should* work for a bounded-buffer problem if there is only one kind of consumer and one kind of producer, so the wrong consumer or producer won't be notified.