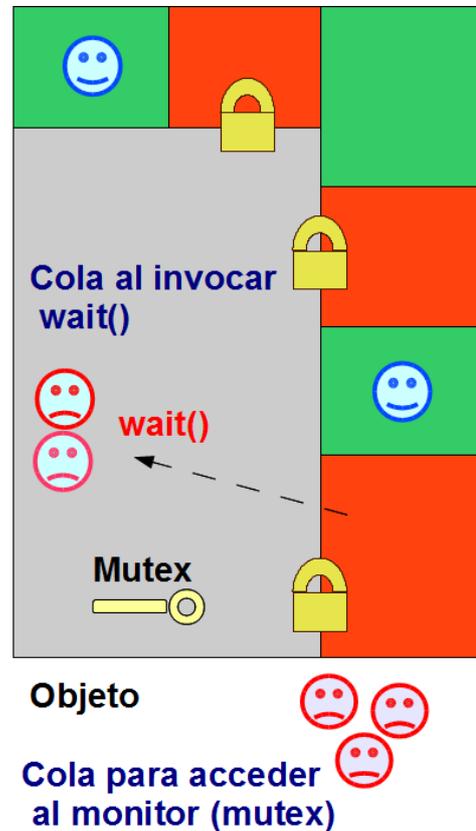


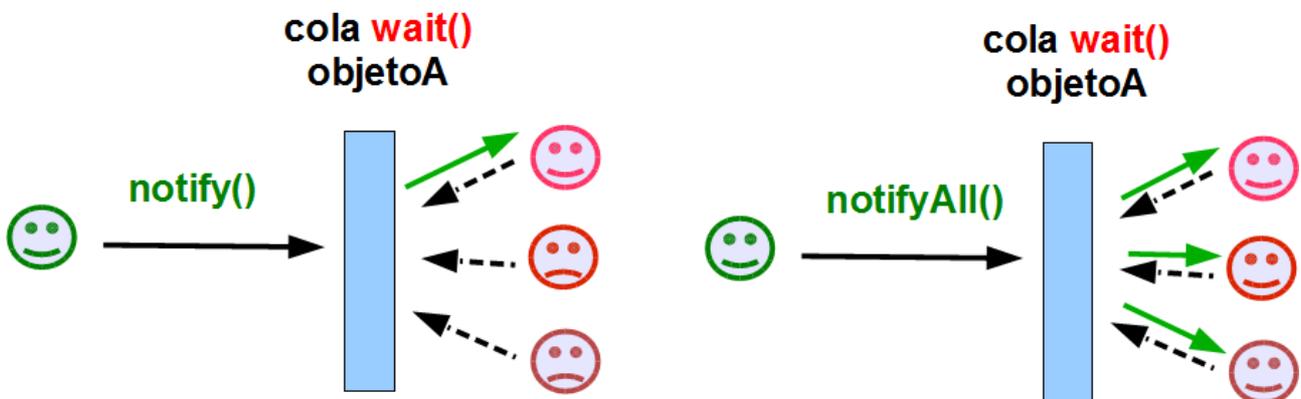
Comunicación entre hilos con wait(), notify() y notifyAll()

Estos métodos pertenecen a la clase `java.lang.Object`. Su comportamiento es el siguiente:

- **wait(). Detiene al hilo que lo invoca** hasta que le sea notificada la posibilidad de continuar.
 - El método `wait()` se debe invocar sobre un objeto compartido por los hilos a sincronizar.
 - Para poder invocar a `wait()` el hilo debe tener el mutex del objeto compartido.
 - La invocación de `wait()` detiene al hilo, pasa a “no ejecutable”, **lo pone en una cola de espera asociada al objeto (cola wait del objeto)**, y libera el mutex del objeto.
 - El método `wait()` puede provocar una `InterruptedException`.
- **notify(). Notifica** a un hilo que invocó `wait()` sobre el mismo objeto y que está en la cola de espera del objeto (cola wait), **que ya puede continuar**.
 - Un hilo sale de la cola de espera del objeto (cola wait) pasando al estado “ejecutable”, y se bloquea hasta conseguir el mutex del objeto para continuar su ejecución.
 - Si hay más de un hilo en la cola de espera (cola wait), `notify()` reactivará solo a uno de ellos. El criterio de selección del hilo a reactivar o pasar a “ejecutable” depende de la implementación de Java.
 - Una vez re-obtenido el mutex del objeto, el hilo que salió de la lista de espera (cola wait) continuará la ejecución del método en la instrucción siguiente a la llamada a `wait()`.



- **notifyAll().** Notifica a todos los hilos puestos en espera para el mismo objeto (cola wait) que ya pueden continuar.
- **Este modelo de notificación es indirecto**



El hilo que invoca `notify()` no tiene ninguna referencia del hilo que está en espera por haber invocado `wait()` sobre el mismo objeto. Cuando un hilo invoca a `notify()`, otro hilo (no se sabe cual) de los que están en espera es reactivado (pasa a “ejecutable”).

Con `notifyAll()` se reactivan, volverán “ejecutables” todos los hilos que estaban bloqueados en la cola de espera del objeto. Sin embargo el mutex, solo podrán tomarlo de uno en uno.

Ejemplo Pintor_Vendedor (productor-consumidor)

Veamos el funcionamiento de estos métodos en el siguiente ejemplo simplificado, que ilustra el problema clásico del **productor-consumidor** (un hilo produce la información que otro hilo consume) y que modela situaciones en las que en una aplicación **se reparte el trabajo entre los hilos**. Por ejemplo, un programa en el que al pulsar una tecla o con el ratón sobre un botón, unos hilos (productores) se encargan de la transferencia de un bloque de datos desde el disco, o de un paquete desde la red, estos datos se almacenan en una zona de memoria, búfer, a la que acceden otros hilos (consumidores) para recuperar esos datos y procesarlos. Tanto los hilos productores como los consumidores realizan su trabajo simultáneamente, por lo que habrá que proteger el acceso a la zona de memoria compartida y coordinar el trabajo de ambos hilos.

En nuestro **ejemplo simplificado** simulamos un hilo que pinta 30 cuadros, y otro hilo debe venderlos de uno en uno. Cuando se pinta un cuadro éste se deposita en un almacén, y hasta que no es retirado del almacén no se podrá depositar otro. Por tanto en el almacén de cuadros sólo puede haber un cuadro almacenado. El pintor pinta cuadros, pero no deposita ningún cuadro en el almacén hasta que éste está vacío. Observa que el recurso compartido por los hilos es el almacén, el pintor lo usa para poner cuadros y el vendedor para sacarlos.

<pre>public class Hilo_Pintor extends Thread{ private AlmacenCuadros almacen; //constructor public Hilo_Pintor(AlmacenCuadros a) { almacen = a; } public void run() { for (int i = 1; i < 30; i++) { almacen.guardar(); } //pinta y guarda en almacén 30 cuadros } }</pre>	<pre>public class Hilo_Vendedor extends Thread { private AlmacenCuadros almacen; //constructor public Hilo_Vendedor(AlmacenCuadros a) { almacen = a; } public void run() { for (int i = 1; i < 30; i++) { almacen.sacar(); } //vende (saca) del almacén 30 cuadros } }</pre>
---	---

```
public class AlmacenCuadros {
    //recurso compartido
    private int cuadros=0;
    //contador de cuadros en almacen
    public synchronized void guardar(){
        try{
            while(cuadros >0)
                //mientras haya cuadros hay que esperar
                this.wait();
            //el hilo invoca wait() y se pone en la cola de espera wait del objeto
        }catch(InterruptedException e){}
        cuadros++;
        //incrementa en 1 simulando que guarda un cuadro
        System.out.println("Pintado cuadro: "+ cuadros);
        this.notify();
        //notifica que se ha producido el evento 'pintado y guardado un cuadro'
    }
    public synchronized void sacar(){
        try{
            while(cuadros ==0)
                //mientras no haya cuadros hay que esperar
                this.wait();
            //el hilo invoca wait() y se pone en la cola de espera wait del objeto
        }catch(InterruptedException e){}
        cuadros--;
        //decrementa 1 simulando que vende un cuadro
        System.out.println("Vendido cuadro: "+ cuadros);
        this.notify();
        //notifica que se ha producido el evento 'vendido y sacado un cuadro'
    }
}
```

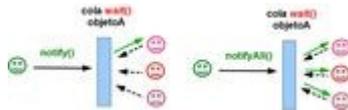

CREDENCIALES

Imagen



Datos de licencia

Autor: Isabel M. Cruz Granados
Licencia: Uso educativo-nc.
Procedencia: Elaboración propia.



Autor: Isabel M. Cruz Granados
Licencia: Uso educativo-nc.
Procedencia: Elaboración propia.

```
Salida - ProductorConsumidorImágenes (ms)  
Finido cuadro: 1  
Vendido cuadro: 0  
Finido cuadro: 1  
Vendido cuadro: 0
```

Autoría: Isabel M. Cruz Granados
Licencia: Uso educativo-no comercial.
Procedencia: Captura de pantalla de la Salida de NetBeans, propiedad Sun Microsystems, bajo licencia GNU GPL v2.