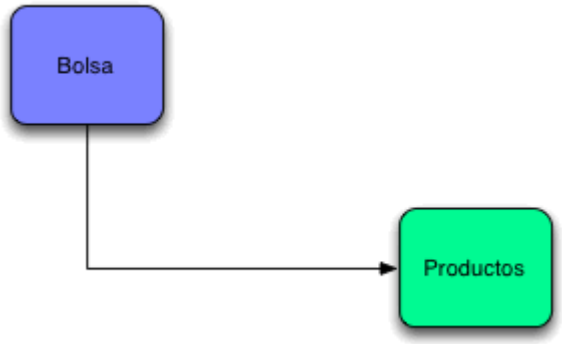
Todos conocemos la clase Object y los métodos principales que posee como equals y hashcode. Sin embargo a mucha gente le cuesta entender para que sirven java wait notify,dos métodos que pertenecen a la clase Object y estan relacionados con programación concurrente, vamos a abordarlo. Para ello vamos a crearnos dos clases sencillas (Bolsa y Producto) donde una Bolsa contiene varios Productos.




Vamos a ver su código:

```
package com.arquitecturajava;

import java.util.ArrayList;

public class Bolsa {

private ArrayList<Producto> listaProductos = new
ArrayList<Producto>();

public void addProducto(Producto producto) {

if (!estaLlena())
```

```
 listaProductos.add(producto);
 }

public ArrayList<Producto> getListaProductos() {
 return listaProductos;

}

public int getSize() {
 return listaProductos.size();
 }
 public boolean estaLlena() {

return listaProductos.size() >= 5;
 }
}



package com.arquitecturajava;

public class Producto {

 private String nombre;

 public String getNombre() {
 return nombre;
 }

 public void setNombre(String nombre) {
 this.nombre = nombre;
 }
```

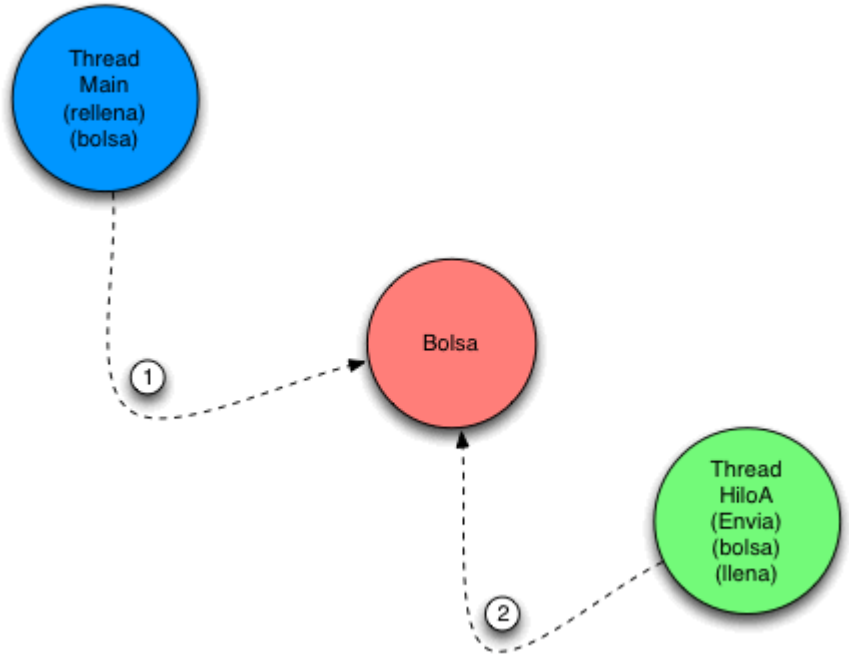
```
}
```

Lo único que tiene de peculiar la bolsa que hemos creado es que no permite que se le añadan más de 5 elementos. Ahora vamos a construir dos Threads un primer Thread que va poco a poco llenando la bolsa (thread main)  y otro Thread que cuando la bolsa este llena la envía. Vamos a ver el Thread que envía la bolsa:

```
package com.arquitecturajava;

public class HiloEnvio extends Thread {

private Bolsa bolsa;

public HiloEnvio(Bolsa bolsa) {
 super();
 this.bolsa = bolsa;
 }

@Override
 public void run() {

if (bolsa.estaLlena() != true) {

try {
 synchronized (bolsa) {
 bolsa.wait();
 }

 } catch (InterruptedException e) {
```

```
 // TODO Auto-generated catch block
 e.printStackTrace();
 }

 System.out.println("Enviando la bolsa con "+
bolsa.getSize()+"elementos");
 }

}

public Bolsa getBolsa() {
 return bolsa;
 }

public void setBolsa(Bolsa bolsa) {
 this.bolsa = bolsa;
 }

}
```

Este Hilo se encarga de sacarnos por pantalla el mensaje de “Enviando la bolsa con 5 elementos” . Para ello recibe la Bolsa como parámetro y chequea que esta llena. Usa un bloque de código sincronizado (syncronized)  . Este bloque de código coordina el trabajo entre nuestros dos Threads y permite que un Thread llene la Bolsa y cuando este llena el otro Thread la envíe.

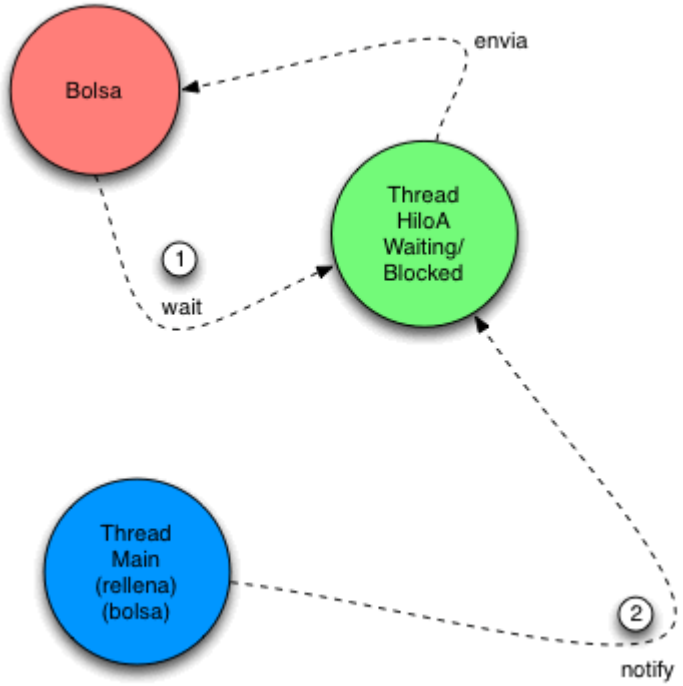



Lamentablemente ambos Threads deben de estar sincronizados ya que hasta que la bolsa no este llena no la podemos enviar. Para sincronizar el trabajo de los Threads usamos la pareja wait notify. De tal forma que nuestro primer HiloEnvio se pondrá a esperar(wait) hasta que la Bolsa este llena antes de enviarla. Para rellenar la bolsa usamos el Thread del programa principal que cada segundo añadirá un nuevo elemento a la lista.

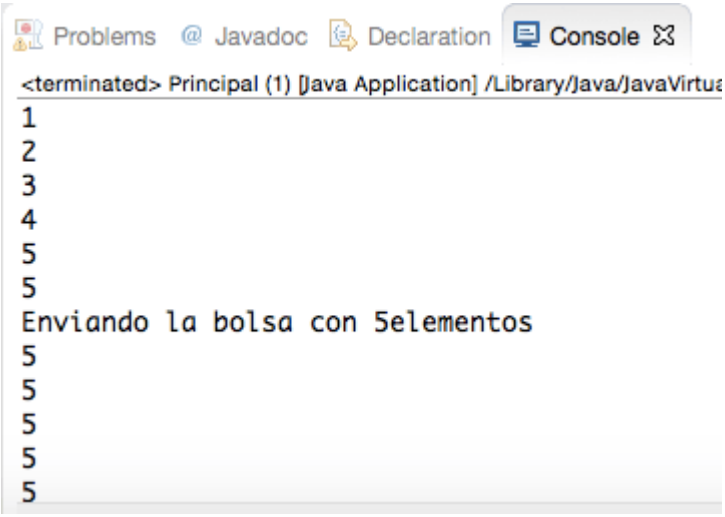
```
package com.arquitecturajava;

public class Principal {

 public static void main(String[] args) {

 Bolsa bolsa= new Bolsa();
 HiloEnvio hilo= new HiloEnvio(bolsa);
```

```
 hilo.start();

 for(int i=0;i<=10;i++) {

 Producto p= new Producto();
 try {

 synchronized (bolsa) {

 Thread.sleep(1000);
 if (bolsa.estaLlena()) {
 bolsa.notify();
 }
 }

 } catch (InterruptedException e) {
 e.printStackTrace();
 }

 bolsa.addProducto(p);
 System.out.println(bolsa.getSize());

 }

 }

}
```

Como vemos el método main va añadiendo productos a la Bolsa y cuando esta llena notifica (notify) al otro Thread de que ya  puede procesarlo saliendo del estado wait en el que se encuentra.




El uso de wait() y notify() es muy habitual en programación concurrente y son parte de la clase Object ya que lo que bloqueamos y sincronizamos es el acceso a cada uno de los objetos. Si ejecutamos el programa nos saldrá por consola el siguiente resultado:




En cuanto tenemos 5 elementos y la bolsa llena el Thread principal no puede añadir más pero ha notificado al otro Thread que en cuando pueda realice su trabajo que es enviar la bolsa.