Clases *Contenedoras*

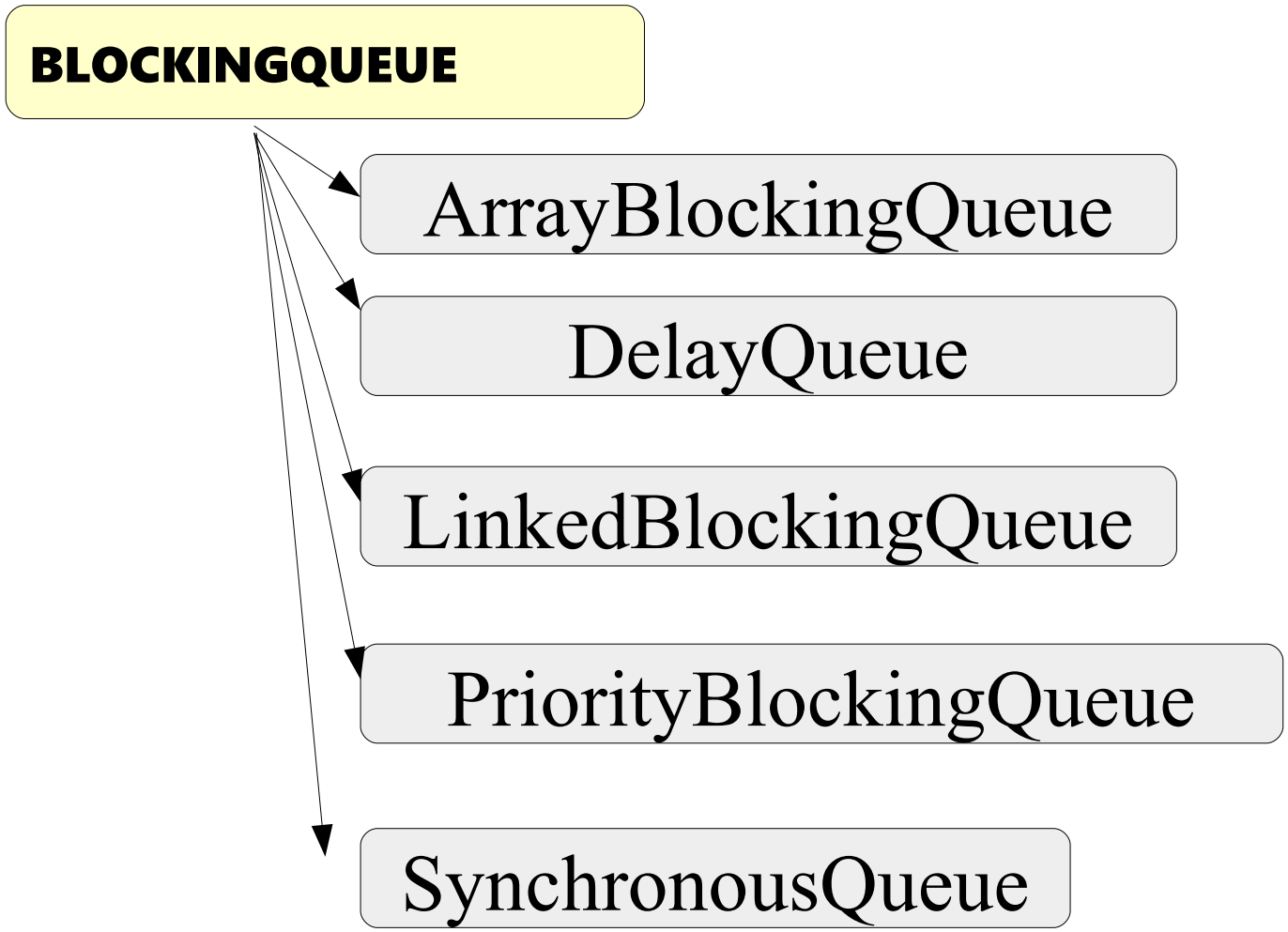
- Versiones sincronizadas de las principales clases contenedoras
- Disponibles en java.util.concurrent
- Clases:
  - ConcurrentHashMap, CopyOnWriteArrayList
  - ReadWriteLock
- Colas Sincronizadas – Interfaz BlockingQueue
    - *Clases:*
      - *LinkedBlockingQueue, ArrayBlockingQueue, SynchronousQueue, PriorityBlockingQueue,y DelayQueue*

Interfaz *BlockingQueue*

- BlockingQueue establece que es una Cola tipo FIFO.
- Los elementos insertados en un pedido en particular son recuperados en ese mismo pedido,
- Si la cola esá vacía bloqueará el hilo de llamada hasta que el elemento esté listo para ser recuperado.
- Cualquier intento de insertar un elemento dentro de una cola que esté llena bloqueará el hilo hasta que haya espacio disponible

	Throws excepción	Special especial	Blocks	TimesOut Value
Insert	add(o)	offer(o)	put(o)	offer(o, timeout, timeunit)
Remove	remove(o)	poll()	take()	poll(timeout, timeunit)
Examine	element()	peek()		

Implementaciones de la Interfaz



ArrayBlockingQueue

```
public class Productor implements Runnable{
   protected BlockingQueue cola = null;

   public Producer(BlockingQueue cola1) {
      this.cola = cola1;
   }
   public void run() {
      try {
         cola.put("1");
         Thread.sleep(1000);
         cola.put("2");
         Thread.sleep(1000);
         cola.put("3");
      } catch (InterruptedException e)
        e.printStackTrace();}
   }
}
```

```
public class Consumidor implements
Runnable{
   protected BlockingQueue cola = null;
   public Consumidor (BlockingQueue col1{
      this.cola = col1;  }
   public void run() {
      try {
         System.out.println(cola.take());
         System.out.println(cola.take());
         System.out.println(cola.take());
      } catch (InterruptedException e) {
         e.printStackTrace(); } }
}
```

```
public class BlockingQueueEj {
   public static void main(String[] args) throws Exception
{

   BlockingQueue cola=new ArrayBlockingQueue(1024);
      Producto productor = new Productor(cola);
      Consumidor consumidor = new Consumidor(cola);
      new Thread(productor).start();
      new Thread(consumidor).start();
      Thread.sleep(4000); }
}
```

DelayQueue

- Implementa la interfaz BlockingQueue.
- DelayQueue bloquea los elementos internamente hasta que pase un cierto tiempo.
- Los elementos deben implementar la interfaz java.util.concurrent.Delayed. Así es como se ve la interfaz:
- La instancia de TimeUnit pasada al método getDelay () indica en qué unidad de tiempo debe devolverse el retraso.

PriorityBlockingQueue

- Es una cola concurrente ilimitada.
- Utiliza las mismas reglas de ordenación que la clase java.util.PriorityQueue.
- No puede insertar nulo en esta cola.
- Todos los elementos insertados en PriorityBlockingQueue deben implementar la interfaz java.lang.Comparable.
- Los elementos se ordenan de acuerdo con la prioridad que decida en su implementación comparable.
- No impone ningún comportamiento específico para los elementos que tienen la misma prioridad (compare () == 0).

SynchronousQueue

- Cola que solo puede contener un único elemento internamente.
- Un hilo que inserta un elemento en la cola se bloquea hasta que otro hilo tome ese elemento de la cola.
- Del mismo modo, si un subproceso intenta tomar un elemento y no hay ningún elemento presente, ese subproceso se bloquea hasta que un subproceso inserte un elemento en la cola.
- Llamar a esta clase cola es un poco exagerado. Es más un punto de rendesvouz.

PriorityBlockingQueue

- Es una cola concurrente ilimitada.
- Utiliza las mismas reglas de ordenación que la clase java.util.PriorityQueue.
- No puede insertar nulo en esta cola.
- Todos los elementos insertados en PriorityBlockingQueue deben implementar la interfaz java.lang.Comparable.
- Los elementos se ordenan de acuerdo con la prioridad que decida en su implementación comparable.
- No impone ningún comportamiento específico para los elementos que tienen la misma prioridad (compare () == 0).

Intercambiador

- Clase ***Exchanger<V>*** en Java
- Type Parameters:
  - V - The type of objects that may be exchanged

- Punto de sincronización: donde los hilos pueden intercambiar elementos de a pares.
- Cada subproceso:
  - presenta algún objeto al ingresar al método de intercambio,
  - se empareja con un subproceso asociado y
  - recibe el objeto de su asociado al regresar.
- Forma bidireccional de un SynchronousQueue.

import java.util.concurrent.Exchanger;

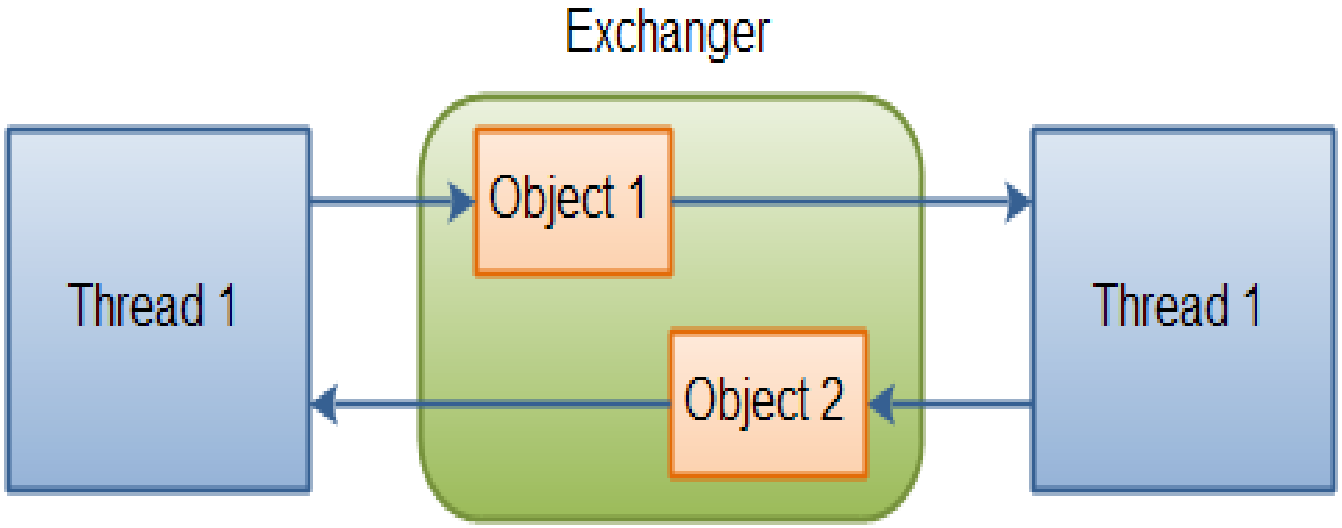
Intercambiador (exchanger)

- Actúa como un canal síncrono (buffer sin espacio), pero solo soporta un método, tipo *rendezvous*, que combina los efectos de *agregar* y *sacar* del buffer
- La operación, llamada "*exchange*" toma un argumento que representa el objeto ofrecido por un hilo a otro, y retorna el objeto ofrecido por el otro hilo

Dos agentes A y B se sincronizan en 1 punto. A le pasa un dato a B; B le pasa un dato a A.

Exchanger

La clase *java.util.concurrent.Exchanger* representa una especie de punto de encuentro donde dos hilos pueden intercambiar objetos.



El intercambio de objetos se realiza a través de uno de los dos métodos de *exchange ()*

Exchanger

```
class Vaciando implements Runnable {
    public void run() {
      DataBuf buf = llenoB;
      try {
        while (buf != null) {
          takeFromBuffer(buf);
          if (buf.isEmpty())
           buf = exchanger.exchange(buf);
        }
      }
  catch (InterruptedException ex) { ...}
    }
  }
```

```
class Llenando implements Runnable {
    public void run() {
      DataBuf buf = vacioB;
      try {
        while (buf != null) {
            addToBuffer(buf);
            if (buf.isFull())
               buf = exchanger.exchange(buf);
            }
       } catch (InterruptedException ex) {}
}
```

```
class VacioLleno
 Exchanger<DataBuf> exchanger =
                        new Exchanger<>();
   DataBuf vacioB= ... un type
   DataBuf llenoB = ...

void start() {
    new Thread(new Llenando()).start();
    new Thread(new Vaciando()).start();
  }
 }
```

Exchanger

```
public class ExchangerEj implements Runnable{
    Exchanger exch = null;
    Object    object    = null;

    public ExchangerEj(Exchanger exch, Object object) {
        this.exch = exch;
        this.object = object;
    }
    public void run() {
        try {
            Object previo = this.object;
            this.object = this.exch.exchange(this.object);
            System.out.println(Thread.currentThread().getName() + " exchanged " +
previo +" a " + t
        } catch (Inte
    }
```

```
....Exchanger exchanger = new Exchanger();
ExchangerEj exchangerRunnable1 =  new ExchangerEj(exchanger, "A");
ExchangerEj exchangerRunnable2 =    new ExchangerEj(exchanger, "B");

new Thread(exchangerRunnable1).start();
new Thread(exchangerRunnable2).start();
 }
```

Exchanger

```
public class ExchangerEj implements Runnable{
    Exchanger exch = null;
    Object    object    = null;

    public ExchangerEj(Exchanger exch, Object object) {
        this.exch = exch;
        this.object = object;
    }
    public void run() {
        try {
            Object previo = this.object;
            this.object = this.exch.exchange(this.object);
            System.out.println(Thread.currentThread().getName() + " exchanged " +
previo +" a " + t
        } catch (Inte
    }
```

```
....Exchanger exchanger = new Exchanger();
ExchangerEj exchangerRunnable1 =  new ExchangerEj(exchanger, "A");
ExchangerEj exchangerRunnable2 =    new ExchangerEj(exchanger, "B");

new Thread(exchangerRunnable1).start();
new Thread(exchangerRunnable2).start();
 }
```