BLOCKINGQUEUE

Es una cola que soporta operaciones en la que se debe esperar que la cola no este vacia para remover un elemento y tambien esperar que haya espacio disponible para almacenar elementos.

METODOS	THROWS EXCEPTION	VALORES ESPECiALES	BLOQUEANTES	TEMPORALES
INSERTAR	add()	offer()	put()	offer()
REMOVER	remove()	poll()	take()	poll()
EXAMINAR	element()	peek()	__	__

SYNCHRONOUSQUEUE

Es una BlockingQueue en la cual cada operacion de insercion debe esperar por una correspondiente operacion de remocion realizada por otro hilo y viceversa. Una SynchronousQueue no tiene una capacidad interna,ni siquiera la capacidad de uno. Un elemento solo esta presente cuando se intenta removerlo, no se puede insertar un elemento si no existe otro hilo intentando removerlo. Tampoco se puede iterar ya que no hay nada sobre lo cual iterar. El frente de la cola es el elemento que un primer hilo encolado esta intentando agregar, si no existe ese hilo entonces no hay ningun elemento disponible para remover. Una SynchronousQueue actua como una coleccion vacia, la cual no permite elementos nulos. Este tipo de cola es similar al rendezvous. Son utilizadas en diseños donde un objeto ejecutandose en un hilo debe sincronizarse con un objeto ejecutandose en otro hilo para manejar informacion, eventos, o tareas. Esta clase permite una politica para ordenar los (hilos) productores y consumidores en espera. Por defecto, este orden no esta garantizado. Sin embargo una cola con true en su constructor garantiza que los hilos accedan en orden FIFO. Esta clase y su iterador implementan todos los metodos opcionales de las interfaces Collection e Iterator.

Constructores:

SynchronousQueue()::

Crea una SynchronousQueue sin una politica de acceso.

SynchronousQueue(boolean fair)::

Crea una SynchronousQueue con un politica de acceso. Si fair es true entonces los hilos en espera acceden en orden FIFO. De lo contrario, no tienen un orden de acceso determinado.

DELAYQUEUE

Es una cola bloqueante de objetos Delayed en la cual un elemento solo puede sacado de la cola si su retardo ha expirado. El frente de la cola es el elemento Delayed, el cual expiro hace mas tiempo. Si ningun retardo ha expirado, no existe un frente en la cola. La expiraciones ocurren cuando un elemento Delayed llama a un metodo getDelay() y este retorna un valor menor o igual a 0. Aunque un elemento que no expiro no pueda ser sacado de la cola, estos son tratados como elementos normales. Por ejemplo, el metodo size() retorna la cantidad de elementos expirados y no expirados. La SynchronousQueue no permite elementos nulos. La misma y su iterador implementan todos los metodos opcionales de las interfaces Collection e Iterator.

La interfaz Delayed se utiliza para crear objetos que actuan de acuerdo a un retardo determinado. Las implementaciones de esta interfaz deben definir los metodos:

getDelay(TimeUnit retardo)::

Metodo que retorna cuanto tiempo falta para que el retardo se complete.

compareTo()::

Metodo que hereda de la interfaz Comparable, el cual especifica como se deben ordenar las instancias Delayed de acuerdo a lo que devuelva el metodo getDelay().

Constructores:

DelayQueue()::

Crea una DelayQueue vacia.

DelayQueue(Collection c)::

Crea una DelayQueue que contiene elementos de una coleccion de instancias Delayed.

CONCURRENTHASHMAP

Es una tabla hash que soporta la concurrencia en los accesos o actualizaciones. Esta clase tiene la misma especificacion funcional que una Hashtable, e incluye versiones de los metodos de Hashtable. Todas las operaciones de la tabla son Thread-Safe. Esta clase es completamente interoperable con Hashtable en programas que dependen de la seguridad de los hilos pero no de los detalles de

sincronizacion. Las operaciones de acceso generalmente no bloquean la tabla, por lo tanto se pueden solapar con las operaciones de actualizacion, las cuales solo bloquean parte de la tabla. Los accesos reflejan los resultados de la actualizacion mas reciente. Para operaciones de agregacion, los accesos concurrentes mostraran la insercion o remocion de solo algunas entradas. De manera similar, los Iterators y Enumerations retornan elementos que reflejan el estado de la tabla hash en cierto punto o desde la creacion del Iterator o Enumeration. El ConcurrentHashMap no dispara la excepcion, ConcurrentModificationException.
La concurrencia permitida para las operaciones de actualizacion esta dada por el constructor(opcional), concurrencyLevel. La tabla es particionada internamente para permitir el numero indicado de actualizaciones concurrentes. Esta clase y sus vistas e iteradores implementan todos los metodos opcionales de las interfaces Map e Iterator. Como Hashtable pero diferente a HashMap, esta clase no permite que null sea utilizado como clave o valor en la tabla.

Constructores:

ConcurrentHashMap()::
Crea un mapa con una capacidad inicial(16), un factor de carga(0.75) y un nivel de concurrencia(16).

ConcurrentHashMap(int capacidadInicial)::
Crea un mapa con la capacidad inicial especificada, factor de carga(0.75) y nivel de concurrencia(16).

ConcurrentHashMap(int capacidadInicial, float factorDeCarga)::
Crea un mapa con la capacidad inicial y factor de carga especificados y con un nivel de concurrencia(16).

ConcurrentHashMap(int capacidadInicial, float factorDeCarga, int nivelDeConcurrencia)::
Crea un mapa con los valores de capacidad inicial, factor de carga y nivel de concurrencia especificados.

ConcurrentHashMap(Map m)::
Crea un mapa con el mismo mapeo que el mapa dado.
ConcurrentHashMap implementa la interfaz ConcurrentMap, la cual representa un mapeo entre una clave y un valor, la cual es capaz de manejar sus accesos concurrentemente. Ademas,contiene metodos para proveer funcionalidades atomicas com:

putIfAbsent(K clave, V valor)::
Si no existe un valor asociado con esa clave, entonces asocia ese valor con dicha clave.

remove(Object clave)::
Remueve la clave y su valor correspondiente.

replace(K clave, V valor)::
Reemplaza el valor asociado con esa clave.

BIBLIOGRAFIA

http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/BlockingQueue.html
http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/SynchronousQueue.html
http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/DelayQueue.html
http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Delayed.html
http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ConcurrentHashMap.html
http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ConcurrentMap.html