Lab8

Queue: Java API

The Java ConcurrentSkipListSet class

The ConcurrentSkipListSet<E> class is one of the collection classes ensuring safe concurrent access provided by the java.util.concurrent package. The elements in the collection are ordered either based on their natural order or on a provided Comparator. The class offers the main methods usually offered by a list:

- boolean add(E e);
- boolean remove(Object o);
- boolean contains(Object o);

Due to the order maintained among the list elements, also a series of methods returning elements based on this order are provided (*first() ; last() ; ceiling (E e) ; floor (E e) ;* etc). More information about the class can be found in the Java API documentation.

The Java ConcurrentLinkedQueue class

The ConcurrentLinkedQueue<E> class is one of the queue classes provided by the same java.util.concurrent package that offers safe concurrent access. The class provides the usual queue specific methods:

- boolean add(E e); (adds an element to the queue tail)
- E poll(); (removes an element from the queue head)
- E peek(); (gets an element from the queue head without removing it)

The class offers also access inside the queue, not only on the head and tail, through two methods: *contains(Object o)* and *remove(Object o)*. More information about the class can be found in the Java API documentation.

ArrayBlockingQueue

It is a class that offers the FIFO functionality of a queue in a concurrent safe manner with a fixed capacity for the number of elements it is able to hold, being backed by an array. Besides the methods that simply return a specific boolean value in case of failing on adding or removing a queue element, the class (as also the rest of the blocking ones) offers a set of methods that wait (block) in the current thread until the operation is possible. The main ones are *put(E e)* which waits until space for an element is available in the queue, and *take()* which waits until there is at least one element in the queue to be returned. More information about the class can be found here.

LinkedBlockingQueue

It resembles the functionality of the ArrayBlockingQueue with the main difference that is being backed by a linked list instead of an array and that it is optionally bounded. By default the capacity is given by Integer.MAX_VALUE, but this can also be set to be fixed at a lower number through the class constructor. More information about the class can be found here.

DelayQueue

This class is an unbounded queue with a specific characteristic for the elements it holds. They are supposed to extend the *Delayed* interface. Through this a time delay is associated with each element, the class being able to provide the feature of restricting the removal of the elements until this delay expires. The first element - head of the queue - that can be removed by the *take()* method (which blocks in case no element expired) is the element for which the delay expires first. More information about the class can be found here.

PriorityBlockingQueue Used inside a router for instance

It provides a priority ordering of the elements inserted (which should be comparable) based on the natural order by the default (e.g. smaller than for integers) or on a Comparator object specified through the class constructor. This way, the head of the queue is considered to be the "smallest" element with respect to the comparison criteria. The queue is unbounded, waiting (blocking) occuring only when it is empty on the call of the *take()* method. More information about the class can be found here.

SynchronousQueue

It is not exactly a queue in the true sense of a collection, being more like a rendezvous channel without any capacity. An attempt to insert an element by a thread will succeed only when another thread is trying to remove an element. As long as there isn't a simultaneous call for *put(E e)* at the same time with an existent *take()* call, the *take()* call will block (and viceversa). More information about the class can be found here.