Week 09 - Generics

Topics covered in this week

- Generic classes
- Generic methods
- super vs. extends
- bounded and unbounded wildcards
- local variable type inference (Java 10+)

Reading material

- Fundamentals
- Generic types
- Generic methods
- Type parameters
- Type arguments
- Local variable type inference

Homework

Difficulty	Problem	Notes

EASY	Transform the following code to use generics:
	`interface MyCollection {`
	`  public boolean containsAll(Collection c);`
	`  public boolean addAll(Collection c);`
	`}`
	Write an implementation of this interface and use it in at least 2 examples.	

EASY	Write a *generic* function to sort any collections using *bubble sort*. Test it using unit tests.	

HARD

Implement an efficient generic priority queue class, similar to the PriorityQueue generic class defined by the Java API. Your priority queue will must support *O(log n)* insertion of an arbitrary value into the priority queue, *O(log n)* removal of the largest element from the priority queue, and constant time lookup of the largest element (without removing it from the queue). Note that the above complexity measures assume that a comparison can be performed in constant time.

You are free to choose the internal representation of the priority queue and you cannot use the built-in Java PriorityQueue class and you must be sure that your code has the right asymptotic complexity. Of course, you must write all the code yourself.

Implementaion details:

- *Generic Parameter:* The priority queue class must be a generic class parameterized by a type variable E, where objects of type E must implement the java.lang.Comparable interface (which is itself generic).
- *Interface:* Your priority queue class should itself implement the java.lang.Comparable interface, so that two priority queues (with the same element type) could be compared. You can choose any basis for comparing two priority queues that you like, such as using the value of the largest element in each priority queue, using the size of each priority queue, etc. (Yes, this is a silly requirement, but it's good practice for using generics).
- *Constructors:* The priority queue class must implement two constructors: One that takes no parameters and creates an empty priority queue with a default maximum size (of your choosing, but at least 10,000); and one that takes an integer MaxSize parameter and creates an empty priority queue with the specified maximum size.

Methods

void insert(E e) - Inserts the element e into the priority queue (in O(log n) time). If the priority queue has already reached its maximum size, then an exception may be raised. The priority queue does not have to be resized to increase its capacity - but you may do so if you like.

E remove() - Removes the largest element from the priority queue (in *O(log n)* time).

void clear() - Removes all elements from the priority queue (in constant time).

E head() - Returns the head (i.e. the largest element) of the priority queue (in constant time), but does not remove it.

boolean isEmpty() - Returns true if the priority queue is empty, false otherwise.

Sorting

This should be a generic method parameterized by type E, where E ranges over the classes that implement the java.lang.Comparable interface (for comparing with other E's). The method should take a List<E> as a parameter and return a List<E> containing the same elements but in sorted order. It should perform the sort by inserting every element of the input list into a priority queue (of your priority queue class) and then removing every element from the priority queue and inserting it in the appropriate place of the output list. You may choose to use any class implementing List<E> (e.g. ArrayList<E>) that you want for the return value.

Test it using unit tests.