```
package mypackage;

import java.util.Comparator;



// An implementation of an adaptable priority queue using an array-based heap.

public class HeapAdaptablePriorityQueue<K1,K2,V> extends HeapPriorityQueue<K1,K2,V> implements
AdaptablePriorityQueue<K1,K2,V> {


   //---------------- nested AdaptablePQEntry class ----------------

   //Extension of the PQEntry to include location information.

   protected static class AdaptablePQEntry<K1,K2,V> extends PQEntry<K1,K2,V> {

      private int index; // entry's current index within the heap

      public AdaptablePQEntry(K1 key1, K2 key2, V value, int j) {

      super(key1, key2, value); // this sets the key and value

      index = j; // this sets the new field

      }

      public int getIndex( ) { return index; }

      public void setIndex(int j) { index = j; }

   } //---------- end of nested AdaptablePQEntry class ----------


   //Creates an empty adaptable priority queue using natural ordering of keys.

   public HeapAdaptablePriorityQueue( ) { super( ); }


   // Creates an empty adaptable priority queue using the given comparator.

   public HeapAdaptablePriorityQueue(Comparator<K1> comp, Comparator<K2> comp2) { super(comp,
comp2);}




   // protected utilites
```

```
  // Validates an entry to ensure it is location-aware. */
  protected AdaptablePQEntry<K1,K2,V> validate(Entry<K1,K2,V> entry) throws
IllegalArgumentException {
    if (!(entry instanceof AdaptablePQEntry))
    throw new IllegalArgumentException("Invalid entry");
    AdaptablePQEntry<K1,K2,V> locator = (AdaptablePQEntry<K1,K2,V>) entry; // safe
    int j = locator.getIndex( );
    // if (j >= heap.size( ) || heap.get(j) != locator)
    // throw new IllegalArgumentException("Invalid entry");
    return locator;
  }

  //Exchanges the entries at indices i and j of the array list. */
  protected void swap(int i, int j) {
    super.swap(i,j); // perform the swap
    ((AdaptablePQEntry<K1,K2,V>) heap.get(i)).setIndex(i); // reset entry's index
    ((AdaptablePQEntry<K1,K2,V>) heap.get(j)).setIndex(j); // reset entry's index
  }

  //Restores the heap property by moving the entry at index j upward/downward.
  protected void bubble(int j) {
    if (j > 0 && compare(heap.get(j), heap.get(parent(j))) < 0)
    upheap(j);
    else
    downheap(j); // although it might not need to move
  }

  //Inserts a key-value pair and returns the entry created.
  public Entry<K1,K2,V> insert(K1 key1, K2 key2, V value) throws IllegalArgumentException {
```

```
    checkKey1(key1); // might throw an exception
    checkKey2(key2);
    Entry<K1,K2,V> newest = new AdaptablePQEntry<>(key1, key2, value, heap.size( ));
    heap.add(newest); // add to the end of the list
    upheap(heap.size( ) - 1); // upheap newly added entry
    return newest;
}

//Removes the given entry from the priority queue.
public void remove(Entry<K1,K2,V> entry) throws IllegalArgumentException {
    AdaptablePQEntry<K1,K2,V> locator = validate(entry);
    int j = locator.getIndex( );
    if (j == heap.size( ) - 1) // entry is at last position
        heap.remove(heap.size( ) - 1); // so just remove it
    else {
        swap(j, heap.size( ) - 1); // swap entry to last position
        heap.remove(heap.size( ) - 1); // then remove it
        bubble(j); // and fix entry displaced by the swap
    }
}

// Replaces the key of an entry.
public void replaceKey1(Entry<K1,K2,V> entry, K1 key) throws IllegalArgumentException {
    AdaptablePQEntry<K1,K2,V> locator = validate(entry);
    checkKey1(key); // might throw an exception
    locator.setKey1(key); // method inherited from PQEntry
    bubble(locator.getIndex( )); // with new key, may need to move entry
}
```

```
    // Replaces the key of an entry.
    public void replaceKey2(Entry<K1,K2,V> entry, K2 key) throws IllegalArgumentException {
        AdaptablePQEntry<K1,K2,V> locator = validate(entry);
        checkKey2(key); // might throw an exception
        locator.setKey2(key); // method inherited from PQEntry
        bubble(locator.getIndex( )); // with new key, may need to move entry
    }

    // Replaces the value of an entry.
    public void replaceValue(Entry<K1,K2,V> entry, V value) throws IllegalArgumentException {
        AdaptablePQEntry<K1,K2,V> locator = validate(entry);
        locator.setValue(value); // method inherited from PQEntry
    }

}
```