```
package mypackage;

import java.util.Comparator;

//An abstract base class to assist implementations of the PriorityQueue interface.
public abstract class AbstractPriorityQueue<K1,K2,V>implements PriorityQueue<K1,K2,V> {

   //---------------- nested PQEntry class ----------------
   protected static class PQEntry<K1,K2,V> implements Entry<K1,K2,V> {
      private K1 k1; // key1
      private K2 k2; // key2
      private V v; // value
      public PQEntry(K1 key1, K2 key2, V value) {
      this.k1 = key1;
      this.k2 = key2;
      this.v = value;
      }

      // methods of the Entry interface
      public K1 getKey1( ) { return this.k1; }
      public K2 getKey2( ) {return this.k2;}
      public V getValue( ) { return v; }

      // utilities not exposed as part of the Entry interface
      public void setKey1(K1 key) { this.k1 = key; }
      public void setKey2(K2 key) { this.k2 = key;}
      public void setValue(V value) { v = value; }
   } //----------- end of nested PQEntry class -----------
```

```
  // instance variable for an AbstractPriorityQueue
  //The comparator defining the ordering of keys in the priority queue.
  private Comparator<K1> comp;
  private Comparator<K2> comp2;

  // Creates an empty priority queue using the given comparator to order keys.
  protected AbstractPriorityQueue(Comparator<K1> c1, Comparator<K2> c2) {
    this.comp = c1;
    this.comp2 = c2;
   }

  //Creates an empty priority queue based on the natural ordering of its keys.
  protected AbstractPriorityQueue( ) { this(new DefaultComparator<K1>( ), new
DefaultComparator<K2>()); }

  //Method for comparing two entries according to key ∗/
  //TODO compare the entire entry
  protected int compare(Entry<K1,K2,V> a, Entry<K1,K2,V> b) {
    if(a.getKey1() == "" || b.getKey1() == ""){
      return -1*comp.compare(a.getKey1( ), b.getKey1( ));
    }

    if(comp.compare(a.getKey1(), b.getKey1()) == 0){
      return comp2.compare(a.getKey2(), b.getKey2());
    }

    return comp.compare(a.getKey1( ), b.getKey1( ));
    }
```

```
  //Determines whether a key is valid.
  protected boolean checkKey1(K1 key) throws IllegalArgumentException {
    try {
      return (comp.compare(key,key) == 0); // see if key can be compared to itself
    }
    catch (ClassCastException e) {
      throw new IllegalArgumentException("Incompatible key");
    }
  }

  //Determines whether a key is valid.
  protected boolean checkKey2(K2 key) throws IllegalArgumentException {
    try {
      return (comp2.compare(key,key) == 0); // see if key can be compared to itself
    }
    catch (ClassCastException e) {
      throw new IllegalArgumentException("Incompatible key");
    }
  }
  //Tests whether the priority queue is empty.
  public boolean isEmpty( ) { return size( ) == 0; }
}
```