```
//These are all inside the HashSet class:

   private transient HashMap<E,Object> map = new HashMap<E,Object>();

   public Iterator<E> iterator() {
      return map.keySet().iterator();
   }

//These are all inside the TreeMap class:

   /**
    * Base class for TreeMap Iterators
    */
   abstract class PrivateEntryIterator<T> implements Iterator<T> {
      Entry<K,V> next;
      Entry<K,V> lastReturned;
      int expectedModCount;

      PrivateEntryIterator(Entry<K,V> first) {
         expectedModCount = modCount;
         lastReturned = null;
         next = first;
      }

      public final boolean hasNext() {
         return next != null;
      }

      final Entry<K,V> nextEntry() {
         Entry<K,V> e = next;
         if (e == null)
            throw new NoSuchElementException();
         if (modCount != expectedModCount)
            throw new ConcurrentModificationException();
         next = successor(e);
         lastReturned = e;
         return e;
      }

      final Entry<K,V> prevEntry() {
         Entry<K,V> e = next;
         if (e == null)
            throw new NoSuchElementException();
         if (modCount != expectedModCount)
            throw new ConcurrentModificationException();
         next = predecessor(e);
         lastReturned = e;
         return e;
      }
```

```
  public void remove() {
      if (lastReturned == null)
         throw new IllegalStateException();
      if (modCount != expectedModCount)
         throw new ConcurrentModificationException();
      // deleted entries are replaced by their successors
      if (lastReturned.left != null && lastReturned.right != null)
         next = lastReturned;
      deleteEntry(lastReturned);
      expectedModCount = modCount;
      lastReturned = null;
   }
}

final class EntryIterator extends PrivateEntryIterator<Map.Entry<K,V>> {
   EntryIterator(Entry<K,V> first) {
      super(first);
   }
   public Map.Entry<K,V> next() {
      return nextEntry();
   }
}

final class ValueIterator extends PrivateEntryIterator<V> {
   ValueIterator(Entry<K,V> first) {
      super(first);
   }
   public V next() {
      return nextEntry().value;
   }
}

final class KeyIterator extends PrivateEntryIterator<K> {
   KeyIterator(Entry<K,V> first) {
      super(first);
   }
   public K next() {
      return nextEntry().key;
   }
}

final class DescendingKeyIterator extends PrivateEntryIterator<K> {
   DescendingKeyIterator(Entry<K,V> first) {
      super(first);
   }
   public K next() {
      return prevEntry().key;
   }
}
```

```java
/**
 * Node in the Tree.  Doubles as a means to pass key-value pairs back to
 * user (see Map.Entry).
 */

static final class Entry<K,V> implements Map.Entry<K,V> {
   K key;
   V value;
   Entry<K,V> left = null;
   Entry<K,V> right = null;
   Entry<K,V> parent;
   boolean color = BLACK;

   /**
    * Make a new cell with given key, value, and parent, and with
    * <tt>null</tt> child links, and BLACK color.
    */
   Entry(K key, V value, Entry<K,V> parent) {
      this.key = key;
      this.value = value;
      this.parent = parent;
   }

   /**
    * Returns the key.
    *
    * @return the key
    */
   public K getKey() {
      return key;
   }

   /**
    * Returns the value associated with the key.
    *
    * @return the value associated with the key
    */
   public V getValue() {
      return value;
   }
```

```
    /**
     * Replaces the value currently associated with the key with the given
     * value.
     *
     * @return the value associated with the key before this method was
     *         called
     */
    public V setValue(V value) {
       V oldValue = this.value;
       this.value = value;
       return oldValue;
    }

    public boolean equals(Object o) {
       if (!(o instanceof Map.Entry))
          return false;
       Map.Entry<?,?> e = (Map.Entry<?,?>)o;

       return valEquals(key,e.getKey()) && valEquals(value,e.getValue());
    }

    public int hashCode() {
       int keyHash = (key==null ? 0 : key.hashCode());
       int valueHash = (value==null ? 0 : value.hashCode());
       return keyHash ^ valueHash;
    }

    public String toString() {
       return key + "=" + value;
    }
 }
```