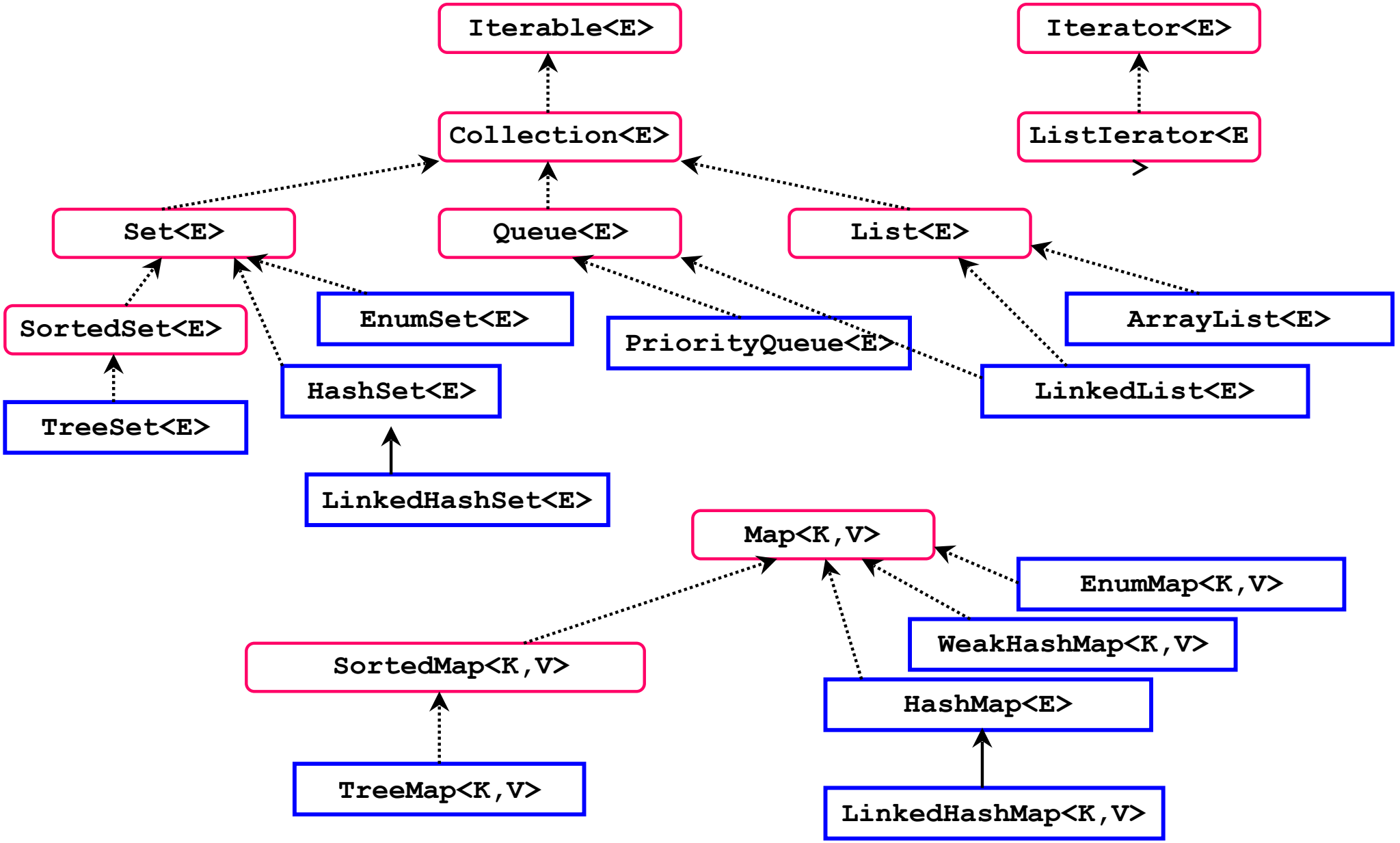
IS311

Java Collections Framework (ต่อ)

Collections

- Collections are holders that let you store and organize objects in useful ways for efficient access.
- In the package `java.util`, there are interfaces and classes that provide a generic collection framework.
- The Collections interfaces: `Collection<E>`, `Set<E>`, `SortedSet<E>`, `List<E>`, `Queue<E>`, `Map<K,V>`, `SortedMap<K,V>`, `Iterator<E>`, `ListIterator<E>`, `Iterable<E>`
- Some useful implementations of the interfaces: `HashSet<E>`, `TreeSet<E>`, `ArrayList<E>`, `LinkedList<E>`, `HashMap<K,V>`, `TreeMap<K,V>`, `WeakHashMap<K,V>`
- Exceptions:
  - `UnsupportedOperationException`
  - `ClassCastException`
  - `IllegalArgumentException`
  - `NoSuchElementException`
  - `NullPointerException`

Type Trees for Collections



The Collections Framework

- The Java collection framework is a set of generic types that are used to create collection classes that support various ways to store and manage objects of any kind in memory.
- A generic type for collection of objects: To get static checking by the compiler for whatever types of objects to want to manage.

Generic Types

Generic Class/Interface Type	Description
The Iterator<T> **interface type**	**Declares methods for iterating through elements of a collection, one at a time.**
The Vector<T> **type**	**Supports an array-like structure for storing any type of object. The number of objects to be stored increases automatically as necessary.**
The Stack<T> **type**	**Supports the storage of any type of object in a pushdown stack.**
The LinkedList<T> **type**	**Supports the storage of any type of object in a doubly-linked list, which is a list that you can iterate though forwards or backwards.**
The HashMap<K,V> **type**	**Supports the storage of an object of type V in a hash table, sometimes called a map. The object is stored using an associated key object of type K. To retrieve an object you just supply its associated key.**

Collections of Objects

- Three Main Types of Collections
  - Sets
  - Sequences
  - Maps
- Sets
  - The simple kinds of collection
  - The objects are not ordered in any particular way.
  - The objects are simply added to the set without any control over where they go.

Collections of Objects

- Sequences
  - The objects are stored in a linear fashion, not necessarily in any particular order, but in an arbitrary fixed sequence with a beginning and an end.
  - Collections generally have the capability to expand to accommodate as many elements as necessary.
  - The various types of sequence collections
    - Array or Vector
    - LinkedList
    - Stack
    - Queue

Collections of Objects

- Maps
  - Each entry in the collection involves a pair of objects.
  - A map is also referred to sometimes as a **dictionary**.
  - Each object that is stored in a map has an associated **key** object, and the object and its key are stored together as a "name-value" pair.

Comparable and Comparator

- The interface `java.lang.Comparable<T>` can be implemented by any class whose objects can be sorted.
  - `public int compareTo (T other)`: return a value that is less than, equal to, or greater than zero as this object is less than, equal to, or greater than the other object.

- If a given class does not implement Comparable or if its natural ordering is wrong for some purpose, `java.util.Comparator` object can be used
  - `public int compare(T o1, T o2)`
  - `boolean equals(Object obj)`

The `Collection` Interface

- **The Collection Interface**
  - The basis of much of the collection system is the Collection interface.

- **Methods:**
  - *`public int size()`*
  - *`public boolean isEmpty()`*
  - *`public boolean contains(Object elem)`*
  - *`public Iterator<E> iterator()`*
  - *`public Object[] toArray()`*
  - *`public <T> T[] toArray(T[] dest)`*
  - *`public boolean add(E elem)`*
  - *`public boolean remove(Object elem)`*
  - *`public boolean containsAll(Collection<?> coll)`*
  - *`public boolean addAll(Collection<? extends E> coll)`*
  - *`public boolean removeAll(Collection<?> coll)`*
  - *`public boolean retainAll(Collection<?> coll)`*
  - *`public void clear()`*

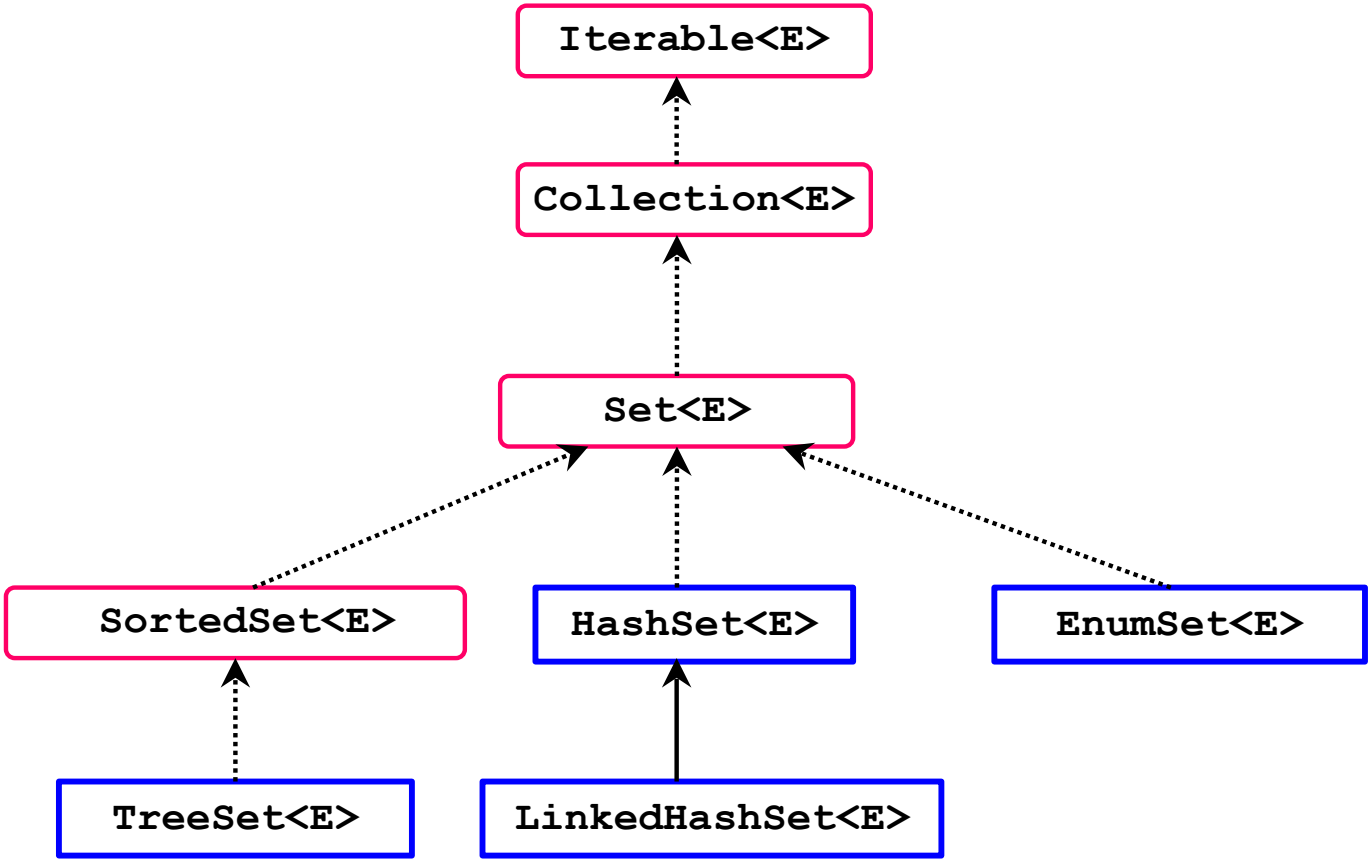
Collection Classes

- **Classes in Sets:**
  - `HashSet<T>`
  - `LinkedHashSet<T>`
  - `TreeSet<T>`
  - `EnumSet<T extends Enum<T>>`
- **Classes in Lists:**
  - *To define a collection whose elements have a defined order- each element exists in a particular position the collection.*
  - `Vector<T>`
  - `Stack<T>`
  - `LinkedList<T>`
  - `ArrayList<T>`
- **Class in Queues:**
  - *FIFO ordering*
  - `PriorityQueue<T>`
- **Classes in Maps:**
  - *Does not extend* `Collection` *because it has a contract that is different in important ways: do not add an element to a Map(add a key/value pair), and a Map allows looking up.*
  - `Hashtable<K,V>`
  - `HashMap<K,V>`
  - `LinkedHashMap<K,V>`
  - `WeakHashMap<K,V>`
  - `IdentityHashMap<K,V>`
  - `TreeMap<K,V>` *: keeping its keys sorted in the same way as* `TreeSet`

Sets

- The simple kinds of collection
- The objects are not ordered in any particular way.
- The objects are simply *added* to the set without any control over where they go.
- Contains no methods other than those inherited from `Collection`
- `Iterator`
  - The elements are traversed in no particular order

Tree of Sets


Iterable<E>
Collection<E>
Set<E>
SortedSet<E>
HashSet<E>
EnumSet<E>
TreeSet<E>
LinkedHashSet<E>

interface **Set<E>**

- `boolean add(E e)`
- `boolean addAll(Collection<? extends E> c)`
- `void clear( )`
- `boolean contains(Object o)`
- `boolean containsAll(Collection<?> c)`
- `boolean isEmpty( )`
- `Iterator iterator( )`
- `boolean remove(Object o)`
- `boolean retainAll(Collection <?> c)`
- `int size( )`
- `Object [] toArray()`
- `<T> T[] toArray(T[] a)`

interface `SortedSet<E>`

`SortedSet` — a Set that maintains its elements in ascending order. Several additional operations are provided to take advantage of the ordering. Sorted sets are used for naturally ordered sets, such as word lists and membership rolls.

- `Iterator`
  - The elements are traversed according to the natural ordering (ascending)

interface `SortedSet<E>`

```
public interface SortedSet<E> extends Set<E> {
    // Range-view
    SortedSet<E> subSet(E fromElement, E toElement);
    SortedSet<E> headSet(E toElement);
    SortedSet<E> tailSet(E fromElement);

    // Endpoints
    E first();
    E last();

    // Comparator access
    Comparator<? super E> comparator();
}
```

Set implementations

- `HashSet` implements `Set`
  - Hash tables as internal data structure (faster)
- `LinkedHashSet` extends `HashSet`
  - Elements are traversed by iterator according to the insertion order
- `TreeSet` implements `SortedSet`
  - *red-black tree structure (*R-B trees) as internal data structure (computationally expensive)

HashSet

- Implement the interface **Set**.
- Implemented using a hash table.
- อ็อบเจ็กท์ที่จะนำมาบรรจุในเซตชนิดนี้ได้ต้องมาจากคลาสที่มีการอิมพลีเม้นท์เมท็อด `hashCode()`
- No ordering of elements.
- **add**, **remove**, and **contains** methods constant time complexity *O(c)*.

`LinkedHashSet`

- extend **`HashSet`** with linked list implementation
- support ordering of elements (ตามลำดับที่เพิ่มเข้าไป)
- **`add`**, **`remove`**, and **`contains`** methods linear time complexity *O(n)*, where *n* is the number of elements in the set.

`TreeSet`

- Implement the interface **`Set`**.
- Implemented using tree structure.
- Guarantees ordering of elements.
- **`add`**, **`remove`**, and **`contains`** methods logarithmic time complexity *O(log (n))*, where *n* is the number of elements in the set.

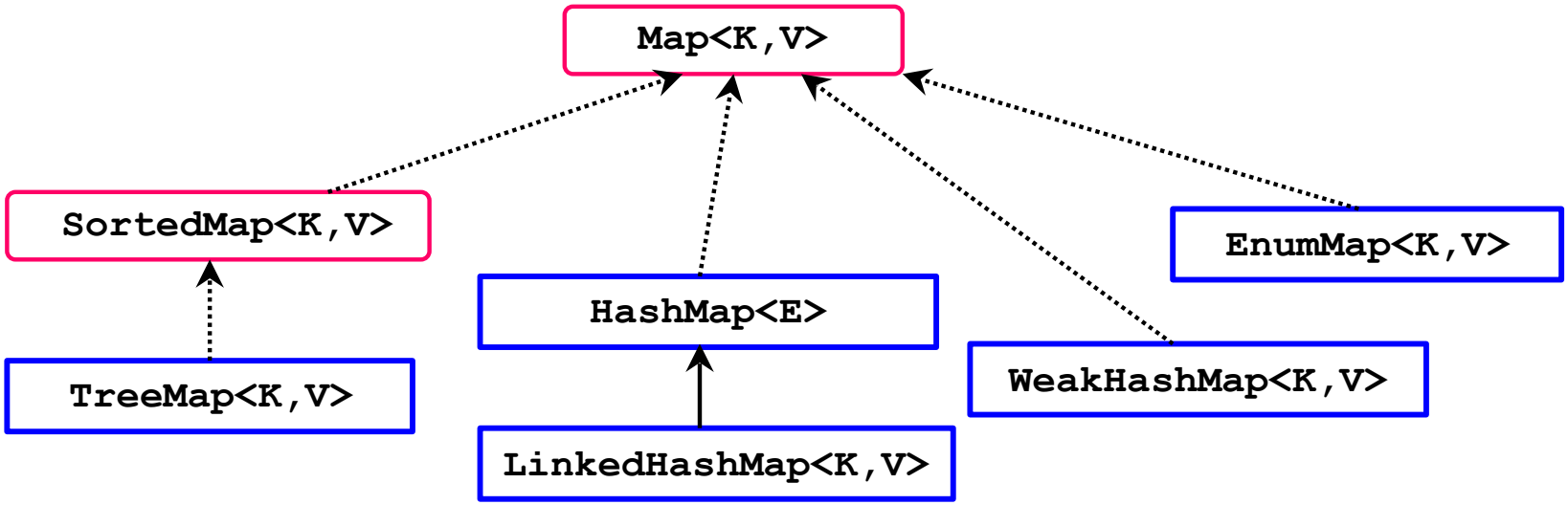
Maps

- Map
  (key, value) binding
  No duplicate keys
- Examples
  identity code (String), person (Person)
  student ID (Integer), student (Student)

Maps

- Each entry in the collection involves a pair of objects.
- A map is also referred to sometimes as a **dictionary**.
- Each object that is stored in a map has an associated **key** object, and the object and its key are stored together as a "name-value" pair.
- Maps do not have an iterator

Tree of Maps


Map<K,V>
SortedMap<K,V>
HashMap<E>
EnumMap<K,V>
TreeMap<K,V>
WeakHashMap<K,V>
LinkedHashMap<K,V>

Classes in Maps

Does not extend `Collection` *because it has a contract that is different in important ways: do not add an element to a Map(add a key/value pair), and a Map allows looking up.*

- `Hashtable<K,V>`
- `HashMap<K,V>`
- `LinkedHashMap<K,V>`
- `WeakHashMap<K,V>`
- `IdentityHashMap<K,V>`
- `TreeMap<K,V>` *: keeping its keys sorted in the same way as* `TreeSet`

interface `Map<K,V>`

`Map` — an object that maps keys to values. A Map cannot contain duplicate keys; each key can map to at most one value.

interface Map<K,V>

```
public interface Map<K,V> {
    // Basic operations
    V put(K key, V value);
    V get(Object key);
    V remove(Object key);
    boolean containsKey(Object key);
    boolean containsValue(Object value);
    int size();
    boolean isEmpty();
    // Bulk operations
    void putAll(Map<? extends K, ? extends V> m);
    void clear();
    // Collection Views
    public Set<K> keySet();
    public Collection<V> values();
    public Set<Map.Entry<K,V>> entrySet();
    // Interface for entrySet elements
    public interface Entry {
        K getKey();
        V getValue();
        V setValue(V value);
    }
}
```

interface SortedMap<K,V>

```
public interface SortedMap<K, V> extends Map<K, V>{

    SortedMap<K, V> subMap(K fromKey, K toKey);
    SortedMap<K, V> headMap(K toKey);
    SortedMap<K, V> tailMap(K fromKey);
    K firstKey();
    K lastKey();

    Comparator<? super K> comparator();
}
```

`HashMap` and `TreeMap` Classes

- The **HashMap** and **HashTree** classes implement the **Map** interface.
- **HashMap**
  - The implementation is based on a hash table.
  - No ordering on (key, value) pairs.
- **TreeMap**
  - The implementation is based on R-B trees *structure*
  - (key, value) pairs are ordered on the key.

`Comparable` and `Comparator`

- The interface `java.lang.Comparable<T>` can be implemented by any class whose objects can be sorted.
  - `public int compareTo (T other)`: return a value that is less than, equal to, or greater than zero as this object is less than, equal to, or greater than the *`other`* object.

- If a given class does not implement `Comparable` or if its natural ordering is wrong for some purpose, `java.util.Comparator` object can be used
  - `public int compare(T o1, T o2)`
  - `boolean equals(Object obj)`

Enhanced for loop

If a class **`extends Iterable<E>`** you can use Java's enhanced for loop of this general form

`for (`*E refVar* **`:`** *collection<E>* **`) {`**
  refVar refers to each element in *collection<E>*
`}`

example

```
ArrayList<String> list = new
          ArrayList<String>();
list.add("First"); list.add("Second");
for (String s : list)
  System.out.println(s.toLowerCase());
```

Algorithms

Java has *polymorphic* algorithms to provide functionality for different types of collections

- Sorting (e.g. `sort`)
- Shuffling (e.g. `shuffle`)
- Routine Data Manipulation (e.g. `reverse`, `addAll`)
- Searching (e.g. `binarySearch`)
- Composition (e.g. `frequency`)
- Finding Extreme Values (e.g. `max`)

Algorithms

All of the algorithms, provided by the `Collections` class, take the form of static methods

- Most of the algorithms operate on `List` objects, but a couple of them (`max` and `min`) operate on arbitrary `Collection` objects

`Collections` static methods

Static library with many useful algorithms

- **Searching…**

```
int pos = Collections.binarySearch(list, key);
```

- **Counting…**

```
int f = Collections.frequency(myColl, item);
```

- **Sorting...**

```
Collections.sort(list);
Collections.sort(list, comparator);
```

Max, Min, Shuffling, reversing, performing set operations and much more…

Sorting

- The sort operation uses a slightly optimized merge sort algorithm
  - Fast: This algorithm is guaranteed to run in n log(n) time, and runs substantially faster on nearly sorted lists.
  - Stable: That is to say, it doesn't reorder equal elements.

ตัวอย่างการใช้เมท็อด sort ของ Collections

```
import java.util.*;

public class SortDemo {
    public static void main( String args[] ) {
      List <String> l = new  ArrayList <String> ();

      for ( int i = 0; i < args.length; i++ )
          l.add( args[ i ] );

      Collections.sort( l );

      System.out.println( l );
    }
}
```

ส่งข้อมูลให้โปรแกรมทาง command line ตอนสั่งรันโปรแกรม เช่น

java SortDemo **One Two Three Four Five**

Arrays

- It is too bad that arrays are not collections
  - You loose all of the power provided by the collection framework
- The class `Arrays` contains
  - various methods for manipulating arrays (such as sorting and searching)
  - It also contains methods that allows arrays to be viewed as lists.

ตัวอย่างการใช้เมท็อด sort ของคลาส Arrays

```
import java.util.*;

public class ArraysSortDemo {
    public static void main( String args[] )
{
     Arrays.sort( args );

     List l = Arrays.asList( args );

     System.out.println( l );
    }
}
```

ส่งข้อมูลให้โปรแกรมทาง command line ตอนสั่งรันโปรแกรม เช่น

java ArraysSortDemo **One Two Three Four Five**

Other Algorithms

- Other algorithms provided by the `Collections` class include
  - Shuffling
  - Data manipulation
    - `reverse()`
    - `fill()`
    - `copy()`
  - Searching
  - Finding extreme values
    - `max()`
    - `min()`

What About User Objects?

- The Collections framework will work with any Java class
- You need to be sure you have defined
  - `equals()`
  - `hashCode()`
  - `compareTo()`
- Don’t use mutable objects for keys in a Map

`hashCode()`

- `hashCode()` returns distinct integers for distinct objects.
  - If two objects are equal according to the `equals()` method, then the `hashCode()` method on each of the two objects must produce the same integer result.
  - When `hashCode()` is invoked on the same object more than once, it must return the same integer, provided no information used in equals comparisons has been modified.
  - It is *not* required that if two objects are unequal according to `equals()` that `hashCode()` must return distinct integer values.

Interface `Comparable`

- This ordering is referred to as the class's *natural ordering*, and the class's `compareTo()` method is referred to as its *natural comparison method*.
- A class's natural ordering is said to be *consistent with equals* if and only if `(e1.compareTo((Object)e2)==0)` has the same boolean value as: `e1.equals((Object)e2)` for every `e1` and `e2` of class C.

ตัวอย่างคลาส Name ที่ใช้กับ collection ได้

```
import java.util.*;

public class Name implements Comparable {

    private String first;
    private String last;

    public Name( String firstName, String lastName ) {
      first = firstName;
      last = lastName;
    }

    public String getFirst() {
      return first;
    }

    public String getLast() {
      return last;
    }
```

คลาส Name (ต่อ)

```
public boolean equals( Object o ) {
  boolean retval = false;

  if (o !=null && o instanceof Name ) {
      Name n = ( Name )o;
      retval = n.getFirst().equals( first ) &&
               n.getLast().equals( last );
  }

  return retval;
}

public int hashCode() {
  return first.hashCode() + last.hashCode();
}

public String toString() {
  return first + " " + last;
}
```

คลาส Name (ต่อ)

```
    public int compareTo( Object o ) throws
                                        ClassCastException {
      int retval;

      Name n = ( Name ) o;

      retval = last.compareTo( n.getLast() );

      if ( retval == 0 )
          retval = first.compareTo( n.getFirst() );

      return retval;
    }

} //Name
```

ตัว การเปรียบเทียบอ็อบเจกต์นี้ ให้ความสำคัญกับนามสกุล มากกว่า ชื่อ จึงเปรียบเทียบ นามสกุลก่อน ถ้านามสกุลเดียวกันจึงจะไปเปรียบเทียบชื่อ ถ้าต้องการให้ความสำคัญกับชื่อ มากกว่า นามสกุล จะแก้ไขตัวอย่างนี้อย่างไร?

ตัวอย่างการนำคลาส Name ไปใช้กับ collection

```
class SortNameDemo {  // run this class to test Name class
    public static void main( String args[] ) {
       List <Name> l = new  ArrayList <Name> ();

    l.add( new Name("Sombat", "Maimee"));
    l.add( new Name("Somsri", "Deejing"));
    l.add( new Name("Amorn", "Nonnan"));
       l.add( new Name("Pichai", "Maimee"));

       Collections.sort( l );

       System.out.println( l );
    }
}
```

ผลลัพธ์จากการรัน จะเรียงตามนามสกุลก่อนแล้วจึงเรียงชื่อ

[Somsri Deejing, Pichai Maimee, Sombat Maimee, Amorn Nonnan]