Chapter 12

02204271

Java Collection Framework

Wrapper Class

- Wrapper class คือคลาสที่ใช้งานเกี่ยวข้องกับข้อมูลที่เป็น primitive type โดย object ของ wrapper class จะไม่สามารถแก้ไขค่าที่กำหนดไว้ได้ (เป็น immutable object)

- primitive type แต่ละตัวจะมี wrapper class ที่เป็นคู่ของมัน

Wrapper Class (Cont.)

Primitive Type	Wrapper Class
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

Wrapper Class (Cont.)

- ตัวอย่างเช่น หากต้องการเก็บค่าเลขจำนวนเต็ม 10 ลงในตัวแปร number ซึ่งมีชนิด ข้อมูลพื้นฐาน (Primitive data type) เป็น int แล้ว สามารถทำได้ดังนี้

```
int i = 10 ;
```

- หากต้องการเก็บค่าเลขจำนวนเต็ม 10 ลงในรูปแบบของวัตถุ โดยนำ Wrapper class ชื่อ Integer สามารถทำได้ดังนี้

```
Integer  i = new Integer(10);

หรือ

Integer i = 10;
```

Wrapper Class (Cont.)

- method ส่วนใหญ่ใน wrapper class จะเป็น static method

- เราสามารถเรียก method เหล่านั้นผ่านทางชื่อคลาสได้เลย เช่น Integer.parseInt(), Double.valueOf()

```
Integer object = new Integer(1);

Integer anotherObject = Integer.valueOf(1);

anotherObject = Integer.parseInt(“1");
```

Autoboxing & Autounboxing

- Boxing: การแปลง primitive type เป็น object
- Unboxing: การแปลง object เป็น primitive type

- ใน J2SE 5.0 มีเพิ่มเข้ามาคือ
  - Autoboxing: การแปลง primitive type เป็น object โดยอัตโนมัติ
  - Autounboxing: การแปลง object เป็น primitive type โดยอัตโนมัติ

- ประโยชน์ของ autoboxing และ autounboxing คือสะดวกต่อการทำงานระหว่าง primitive type และ collection

Autoboxing & Autounboxing (Cont.)

```
//Type Example
//Boxing
int x = 5;
Integer y = new Integer(x);

//Unboxing
Integer x = new Integer("5");
Integer y = x.intValue();

//Autoboxing
int x = 5;
Integer y = x;

//Autounboxing
Integer x = new Integer("5");
int y = x;
```

Java Collection

- จาวาคอลเลคชั่นคือชุดของอินเตอร์เฟสและคลาสในแพคเกจ java.util

- คอลเลคชั่นคลาส คือ คลาสที่ออบเจ็กต์ของมันประกอบขึ้นมาจากสมาชิกย่อย ซึ่งสมาชิกย่อยนั้นต้องเป็นพอยต์เตอร์ไปยังออบเจ็กต์

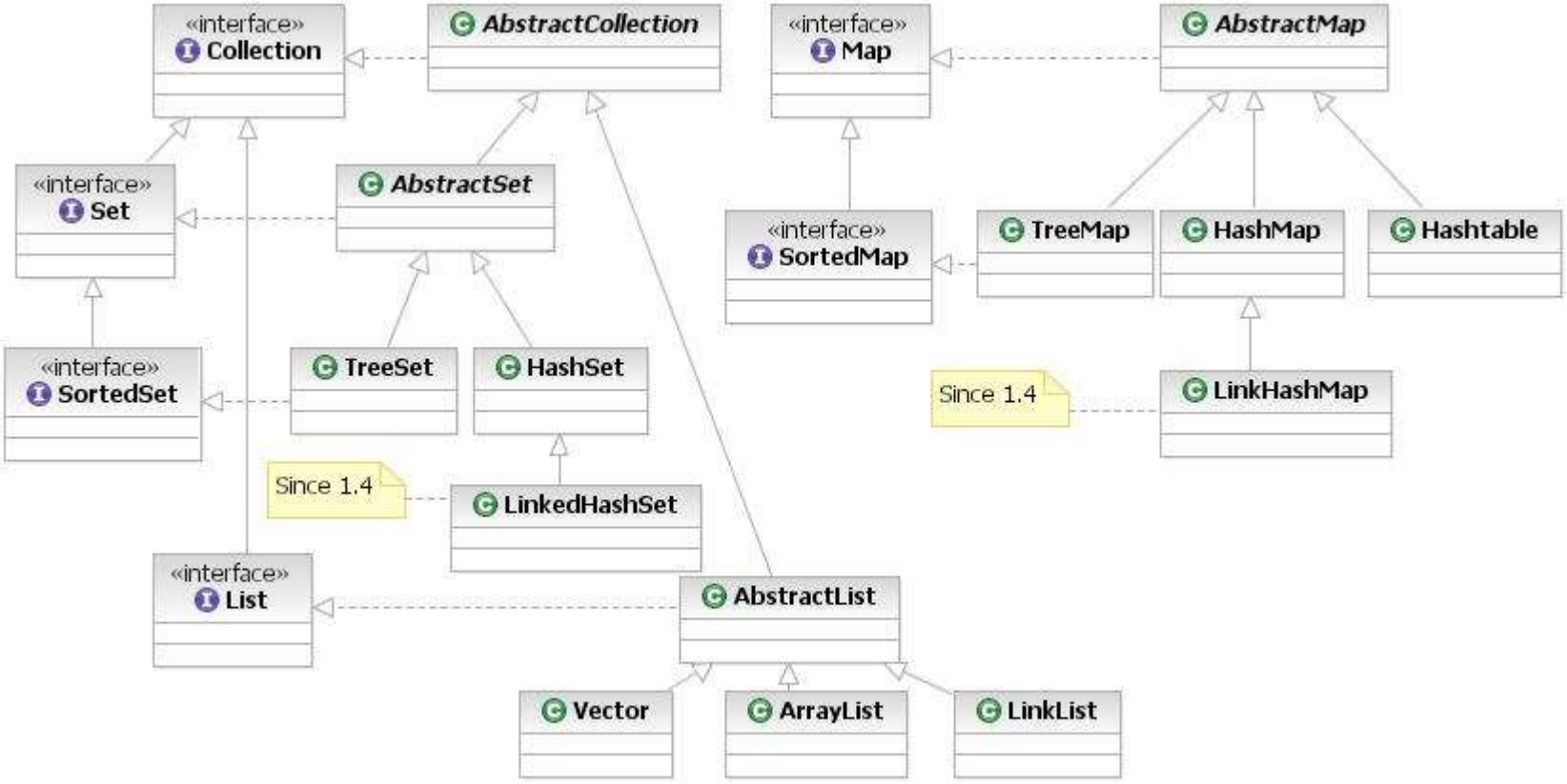
- ดังนั้นเราไม่สามารถใช้ชุดข้อมูลพื้นฐานเป็นส่วนประกอบของคอลเลคชั่นคลาสได้ (แต่อย่าลืมว่าเรามี wrapper class ที่ใช้แทนชุดข้อมูลพื้นฐานได้)

Java Collection (Cont.)

- ผู้ใช้เพียงแต่เรียกใช้เมธอดให้เหมาะสมเท่านั้นก็พอ ไม่จำเป็นต้องรู้ถึงการทำงานภายใน

- ถ้ามีอาร์เรย์ภายใน ก็เป็นหน้าที่ของผู้เขียนคอลเลคชั่นคลาสที่จะเขียนโค้ดเตรียมไว้จัดการกับการขยายอาร์เรย์เอง

Java Collection (Cont.)



Generics

- จาวา 1.5 ขึ้นไปอนุญาตให้เรากำหนดชนิดข้อมูลของสมาชิกของคลาสได้ โดยเขียนในวงเล็บที่สร้างจากเครื่องหมาย “>” และ “<” เรียกว่า Generic Types

```
ArrayList   myList = new ArrayList();
```

```
ArrayList<Double> myList = new ArrayLis<Double>();
```

Generics (Cont.)

```
public class Box {

  private Object content;

  public Object getContent() {
      return content;
  }

  public void setContent(Object content) {
      this.content = content;
  }
}
```

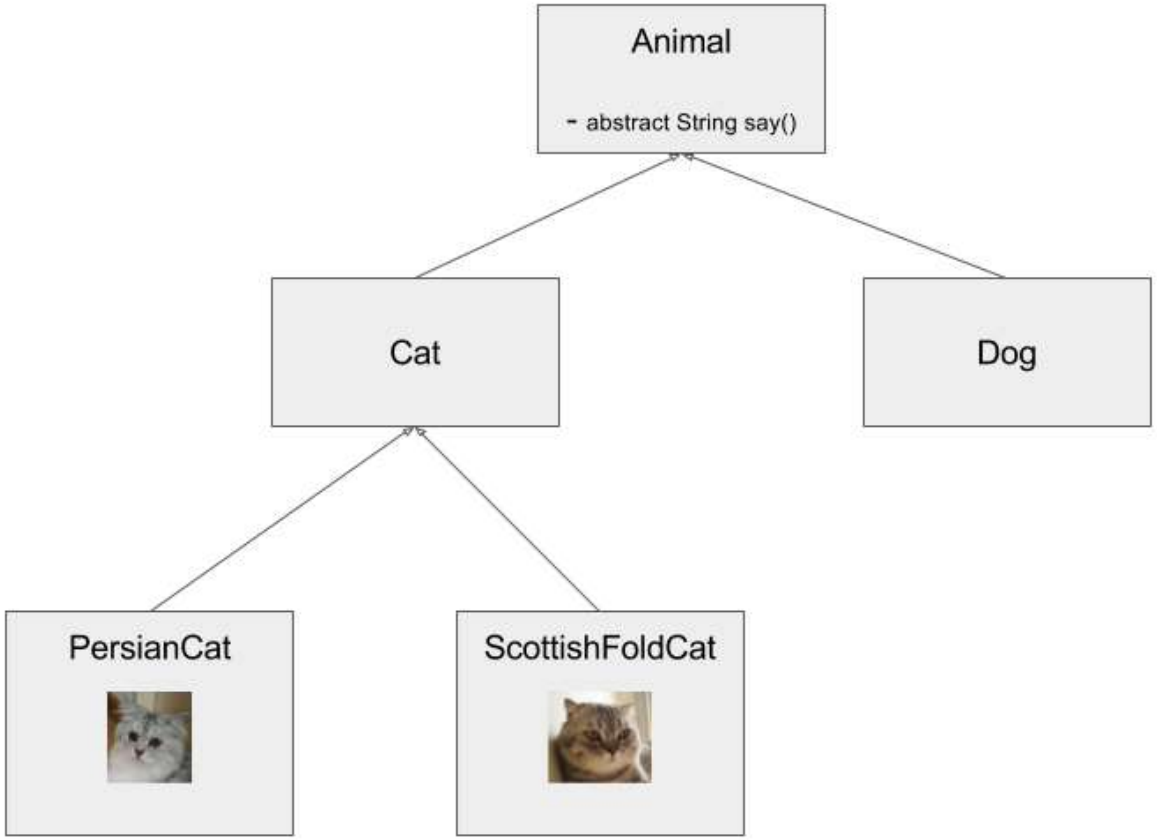
```
Box myBox = new Box();
myBox.setContent("I am a String in a little box");
String boxString = (String) myBox.getContent();
```

Generics (Cont.)

```
public class Box<T> {

    private T content;

    public T getContent() {
        return content;
    }

    public void setContent(T content) {
        this.content = content;
    }
}
```

```
Box<String> myBox = new Box<>();
myBox.setContent("I am a String in a generic box");
// no casting required
String boxString = myBox.getContent();
// compile error
//myBox.setContent(1);
// compile error
//Integer boxInteger = myBox.getContent();
```

Generics (Cont.)



Generics (Bounded Type Parameter)

- Box เก็บได้เฉพาะ Animal รวมถึง sub class ของ Animal เท่านั้นต้องทำยังไง ?

- Bounded Type Parameter ก็คือการกำหนด scope ให้กับ generic type

```
public class Box <T extends Animal>
{

}
```

Generics (Bounded Type Parameter)

```
public class AnimalBox<T extends Animal> {

    private T animal;

    public T getAnimal() {
        return animal;
    }

    public void setAnimal(T animal) {
        this.animal = animal;
    }

    public String poke() {
        return animal.say();
    }

}
```

```
// compile error
//AnimalBox<String> stringBox = new AnimalBox<>();
AnimalBox<Cat> catBox = new AnimalBox<>();
Cat myCat = new Cat();
catBox.setAnimal(myCat);
System.out.println(catBox.poke());// compile error
//catBox.setAnimal("Some String");
```

Generics (Cont.)

```
ArrayList myList = new ArrayList();

myList.add("1");
myList.add(1);


for(int i = 0 ; i < s.size() ; i++)
     System.out.println(myList.get(i));
```

```
ArrayList<Integer> myList = new ArrayList<Integer> ();

myList.add(1);
myList.add(1);


for(int i = 0 ; i < s.size() ; i++)
     System.out.println(myList.get(i));
```

Generics(Cont.)

- ซึ่งเมื่อเขียนแบบนี้ไปแล้ว จะมีแต่ ข้อมูลประเภท Integer เท่านั้นที่ลิสต์จะยอมรับ ดังนั้นในการใช้งานเมธอด get ที่เขียนไว้แล้วในลิงค์ลิสต์ (ยังไม่ต้องรู้มากเพราะมีเรื่องลิงค์ลิสต์ของจาวาต่างหากอีกที)
  - Integer val = myList.get(o);
  - เราก็ไม่ต้องมาทำการเปลี่ยนรูปแบบข้อมูลที่ได้ให้เป็น Integer เพราะว่าถูกบังคับไว้เรียบร้อยแล้ว

Advantages (Generics)

- เราสามารถเห็นได้ทันทีว่าคอลเลคชั่นของเราเก็บอะไรอยู่ คอมไพเลอร์สามารถเช็คชนิดสมาชิกที่เราพยายามเอาใส่คอลเลคชั่นได้ทันที

- ถ้าไม่มีการกำหนดไทป์แบบนี้ก็จะคอมไพล์ผ่านแต่ไปเกิดข้อผิดพลาดตอนรัน (exception)

Collection Interface

```
public Interface Collection extends Iterable<E>{
  boolean add(E o);
  boolean addAll(Collection<? extends E> c);
  void clear();
  boolean contains(Object o);
  boolean containsAll(Collection<?> c);
  boolean equals(Object o);
  int hashCode();
  boolean isEmpty();
  Iterator<E> iterator();
  boolean remove(Object o);
  boolean removeAll(Collection<?> c);
  boolean retainAll(Collection<?> c);
  int size();
  Object[]toArray();
  <T> T[]toArray(T[] a);
}
```

Collection Interface Mathod

- boolean add(E o)
  - พยายามทำให้มี o อยู่ในคอลเลคชั่นนี้ รีเทิร์น true ถ้าการเรียกเมธอดนี้ทำให้ภายในคอลเลคชั่นเปลี่ยนไป รีเทิร์น false ถ้าคอลเลคชั่นนี้มีก็อปปี้ไม่ได้ และ มี o อยู่แล้ว
- boolean addAll(Collection<? extends E> c)
  - เติมสมาชิกจากคอลเลคชั่น c ทั้งหมดลงในคอลเลคชั่นของเรา แต่เมธอดนี้จะถือว่าใช้ไม่ได้ถ้า c ถูกเปลี่ยนแปลงระหว่างที่เมธอดนี้ทำงานอยู่ นั่นก็คือ c จะเป็นคอลเลคชั่นที่เราใช้เรียกเมธอดนี้เองไม่ได้
  - เมธอดนี้รีเทิร์น true ถ้าการเรียกเมธอดนี้ทำให้ภายในคอลเลคชั่นเปลี่ยนไป
  - พารามิเตอร์ <? extends E> หมายถึงอะไรก็ได้ที่เป็นสับคลาสของ E โดย E คือพารามิเตอร์ของคอลเลคชั่นที่เรียกใช้เมธอดนี้ อย่าลืมว่าคลาสใดๆก็เป็นสับคลาสของตัวมันเอง
- void clear()
  - เอาสมาชิกของคอลเลคชั่นนี้ออกไปให้หมด

Collection Interface Mathods

- void **clear**()
  - เอาสมาชิกของคอลเลคชั่นนี้ออกไปให้หมด
- boolean **contains**(Object o)
  - รีเทิร์น true ถ้าคอลเลคชั่นนี้มี o อยู่ หรือพูดได้อีกอย่างว่า รีเทิร์น true ก็ต่อเมื่อคอลเลคชั่นนี้มีสมาชิก e ซึ่ง (o==null ? e==null : o.equals(e))
- boolean **containsAll**(Collection<?> c)
  - รีเทิร์น true ถ้าคอลเลคชั่นนี้มี สมาชิกจากคอลเลคชั่น c อยู่ทั้งหมด

Collection Interface Mathods

- boolean **equals**(Object o)
  - เปรียบเทียบออบเจ็กต์ o กับคอลเลคชั่นนี้ว่าเท่ากันหรือไม่ โดยพื้นฐานแล้ว นี่คือเมธอดที่เอามาจากคลาส Object แต่ถ้าจะเขียนเมธอดนี้เอง เพื่อเปลี่ยนให้เป็นการเปรียบเทียบค่าที่อยู่ในคอลเลคชั่นก็ได้
  - โครงสร้างข้อมูล List กับ Set ของจาวานั้นได้เขียนเมธอดนี้ขึ้นเองโดยกำหนดให้ List ต้องเท่ากับ List และ Set ต้องเท่ากับ Set เท่านั้น ฉะนั้นถ้าเราเขียนเมธอดนี้ให้คลาสของเราเองซึ่งไม่ใช่ List หรือ Set แล้วละก็ เมธอดของเราจะรีเทิร์น false เมื่อนำไปใช้กับ List หรือ set
  - นอกจากนี้ยังไม่สามารถสร้างคลาสซึ่ง implement List กับ Set ในเวลาเดียวกันได้

Collection Interface Mathod

- int hashCode()
  - รีเทิร์นแฮชโค้ดของคอลเลคชั่นนี้ ถ้าเราโอเวอร์ไรด์ equals() แล้วเราต้องโอเวอร์ไรด์ hashcode() ด้วย เพราะ c1.equals(c2) ต้องหมายถึง c1.hashCode()==c2.hashCode()
- boolean isEmpty()
  - รีเทิร์น true ถ้าคอลเลคชั่นนี้ไม่มีสมาชิก
- Iterator<E> iterator()
  - รีเทิร์นอิเทอเรเตอร์ที่จะอนุญาตให้เราดูสมาชิกในคอลเลคชั่นนี้ได้ การเรียงของสมาชิกนั้นไม่ได้มีการกำหนดไว้ในชั้นนี้

Collection Interface Mathod

- boolean **remove**(Object o)
  - เอาสมาชิก e ซึ่ง (o==null ? e==null : o.equals(e)) ออกจากคอลเลคชั่น ถ้ามีอยู่ในคอลเลคชั่น
  - รีเทิร์น true ถ้ามีสมาชิกถูกเอาออกไปจริงๆ
- boolean **removeAll**(Collection<?> c)
  - เอาสมาชิกที่เหมือนกับสมาชิกใน c (โดยเทียบด้วย equals()) ทิ้ง
  - รีเทิร์น true ถ้ามีสมาชิกถูกเอาออกไปจริงๆ
  - c ห้ามเป็น null ไม่งั้นจะ throw exception
- boolean **retainAll**(Collection<?> c)
  - เอาสมาชิกที่เหมือนกับสมาชิกใน c (โดยเทียบด้วย equals()) เหลือไว้ นอกนั้นลบทิ้งหมด
  - รีเทิร์น true ถ้ามีสมาชิกถูกเอาออกไปจริงๆ
  - c ห้ามเป็น null ไม่งั้นจะ throw exception

Collection Interface Mathod

- int size()
  - รีเทิร์นจำนวนสมาชิกในคอลเลคชั่นนี้ ถ้าจำนวนสมาชิกมากกว่า Integer.MAX_VALUE ให้รีเทิร์น Integer.MAX_VALUE
- Object[] toArray()
  - รีเทิร์นอาร์เรย์ซึ่งใส่สมาชิกของคอลเลคชั่นนี้ไว้ทั้งหมด
  - ถ้ามีการกำหนดลำดับของสมาชิกด้วยอิเทอเรเตอร์ไว้แล้ว ลำดับสมาชิกในอาร์เรย์ก็ต้องเป็นไปตามนั้นด้วย

Collection Interface Mathod

- <T> T[] toArray(T[] a)
  - รีเทิร์นอาร์เรย์ซึ่งใส่สมาชิกของคอลเลคชั่นนี้ไว้ทั้งหมด
  - รันไทม์ไทป์ของอาร์เรย์ที่รีเทิร์นให้เป็นชนิดเดียวกับอาร์เรย์ a
  - ถ้าคอลเลคชั่นของเราใส่ a ได้ ก็จะเอาใส่ a แล้วรีเทิร์น a เลย (สมาชิกในด้านหลังของ a ที่มีที่เหลือก็จะถูกเซ็ตเป็น null)
  - มิฉะนั้นต้องสร้างอาร์เรย์ใหม่ซึ่งขนาดใหญ่พอที่จะเก็บคอลเลคชั่นของเราได้
  - ถ้ามีการกำหนดลำดับของสมาชิกด้วยอิเทอเรเตอร์ไว้แล้ว ลำดับสมาชิกในอาร์เรย์ก็ต้องเป็นไปตามนั้นด้วย

Iterator

- Iterator เป็น interface ที่ใช้สำหรับเข้าถึง object ต่าง ๆ ภายใน collection โดยมีมาตั้งแต่ J2SE 1.2

- แต่ต่อมาใน J2SE 5.0 ได้มี for-each loop เข้ามาให้ใช้แทนซึ่งใช้สะดวกกว่า

Iterator (Cont.)

- public Interface Iterator<E>{...}
- boolean hasNext()
  - รีเทิร์น true ถ้ายังมีสมาชิกให้ดูได้อีก (นั่นคือ รีเทิร์น true เมื่อ next() จะรีเทิร์นสมาชิกนั่นเอง
- E next()
  - รีเทิร์นสมาชิกตัวต่อไปตามลำดับที่กำหนดไว้
- void remove()
  - เอาสมาชิกตัวที่พึ่งถูกรีเทิร์นด้วย next() ออกไป
  - เรียกเมธอดนี้ได้หนึ่งครั้งต่อการเรียกใช้ next() หนึ่งครั้ง
  - ถ้าตัวคอลเลคชั่นถูกเปลี่ยนระหว่างที่กำลังลูปด้วยวิธีอื่นที่ไม่ใช่เมธอดนี้ เราจะถือว่าเมธอดนี้ใช้ไม่ได้

Iterator (Cont.)

```
ArrayList al = new ArrayList();

      // add elements to the array list
      al.add("C");
      al.add("A");
      al.add("E");
      al.add("B");
      al.add("D");
      al.add("F");
      Iterator itr = al.iterator();

      while(itr.hasNext()) {
         Object element = itr.next();
         System.out.print(element + " ");
      }
```

List Interface

- ลิสต์ของจาวาคือที่เก็บของเรียงกัน
- โดยสามารถเข้าถึงของแต่ละชิ้นโดยการใช้ดัชนี (index) ได้
- ดัชนีนั้นมีค่าเริ่มจากศูนย์
- ภายในลิสต์อนุญาตให้มีของซ้ำกันได้
- สิ่งที่เป็นลิสต์มักมี null เป็นสมาชิกได้

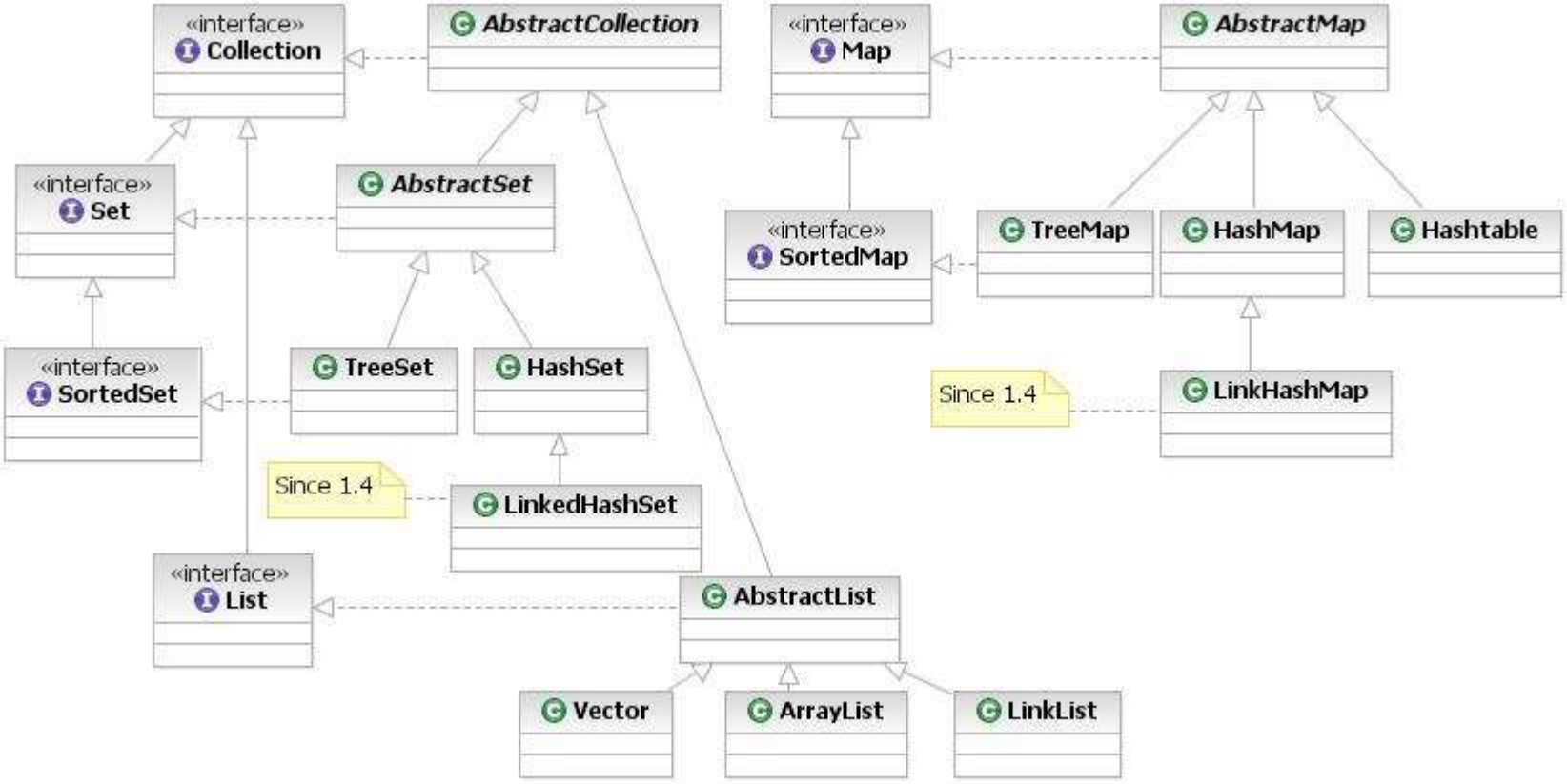
List Interface (Cont.)

- ลิสต์อินเตอร์เฟสมีอิเทอเรเตอร์ของมันเองเรียกว่า ลิสต์อิเทอเรเตอร์ (ListIterator) ซึ่งมีเมธอดสำหรับการใส่ของและเอาของออกจากลิสต์ รวมทั้งสามารถอนุญาตการลูปได้สองทิศทางอีกด้วย

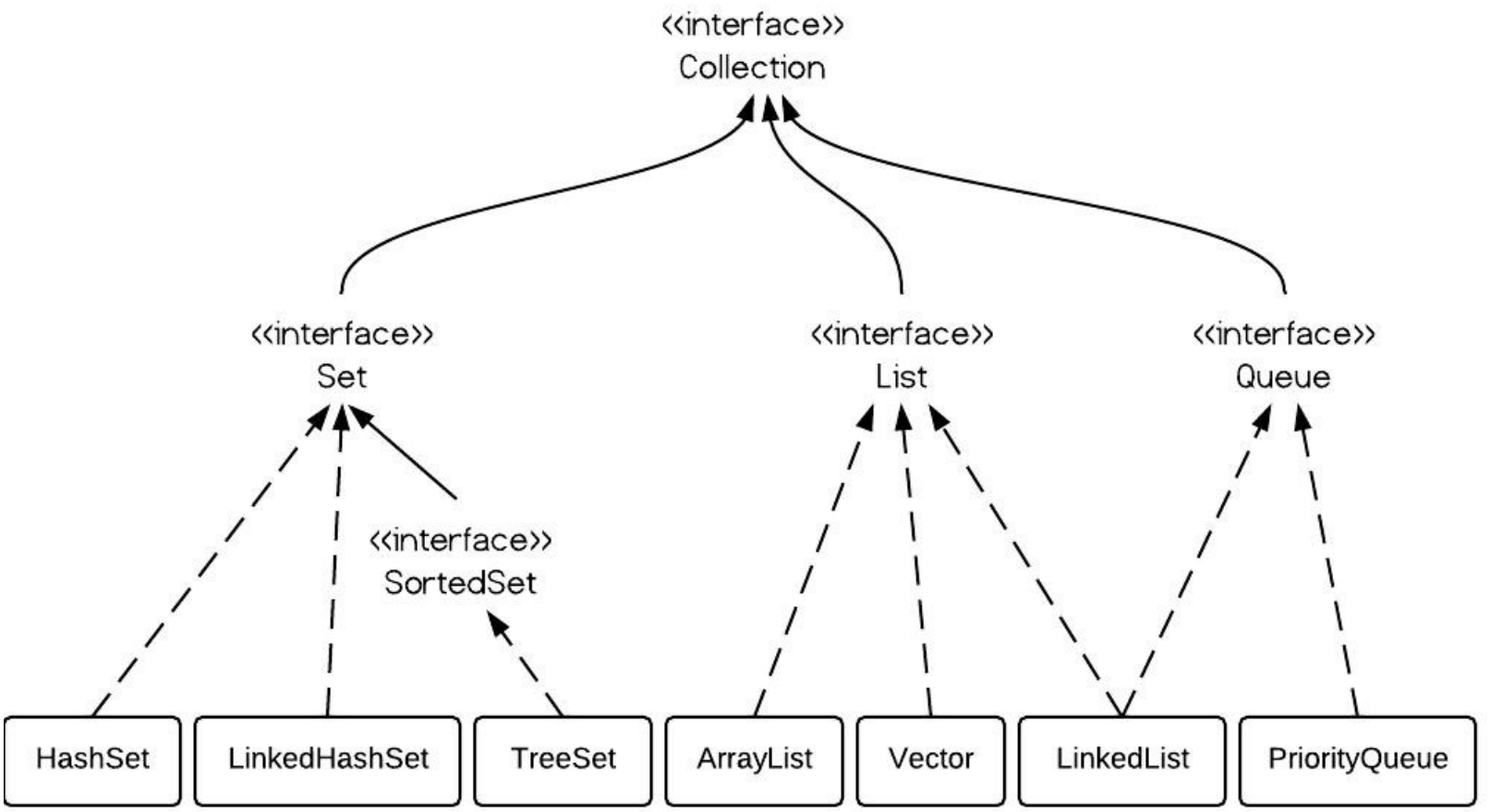
- ลิสต์เป็นอินเตอร์เฟส ดังนั้นโครงสร้างภายในอาจสร้างขึ้นมาจากอะไรก็ได้

- จาวามีคลาสแอ็บสแตรกลิสต์ (AbstractList) เป็นสับคลาสของลิสต์ซึ่งแอ็บสแตรกลิสต์ อิมพลีเม้นท์โค้ดของลิสต์บางส่วน นอกนั้นจะเป็นหน้าที่ของสับคลาสของแอ็บสแตรกลิสต์อีกทีหนึ่ง ซึ่งจริงๆแล้วมีสองคลาสคือ อาร์เรย์ลิสต์ (ArrayList) กับลิงค์ลิสต์(LinkedList)

List Interface (Cont.)



List Interface



ArrayList

- ArrayList ถูกใช้เป็นอาร์เรย์ที่ปรับขนาดได้
- เมื่อมีการเพิ่ม ArrayList ลงไปขนาดของมันจะเพิ่มขึ้นแบบไดนามิก
- องค์ประกอบของมันสามารถเข้าถึงได้โดยตรงโดยใช้เมธอด get and set เนื่องจาก ArrayList เป็นอาร์เรย์เป็นหลัก

LinkedList & Vector

- ถูกกำหนดใช้เป็นข้อมูลที่มีการเชื่อมโยงสองทาง (double linked list)
- ประสิทธิภาพในการเพิ่มและลบจะดีกว่า Arraylist
- แต่จะแย่กว่าในวิธีการกำหนดและรับค่า
- Vector เหมือนกับ ArrayList แต่ข้อมูลจะถูกจัดการแบบ synchronized

listIterator - Forward

```
…
ListIterator litr = al.listIterator();

    while(litr.hasNext()) {
       Object element = litr.next();
       litr.set(element + "+");
    }
itr = al.iterator();

    while(itr.hasNext()) {
       Object element = itr.next();
       System.out.print(element + " ");
    }
```

listIterator - Backward

```
…

ListIterator litr = al.listIterator();

while(litr.hasPrevious()) {
        Object element = litr.previous();
        System.out.print(element + " ");
    }
```

Set Interface

- เหมือน Collection แต่ห้ามมีสมาชิกซ้ำ

- สร้างเซตขึ้นจาก Collection
  - Collection<Double> d = new HashSet(c);

- TreeSet นั้นภายในสร้างขึ้นมาด้วยโครงสร้างต้นไม้
  - ลำดับในการไล่ดูสมาชิกด้วยอิเทอเรเตอร์จะเรียงเป็นรูปแบบที่แน่นอน
- HashSet นั้นสร้างขึ้นมาด้วยโครงสร้างข้อมูลตารางแฮช
  - ซึ่งจะมีการเรียงของสมาชิกภายในค่อนข้างสุ่ม

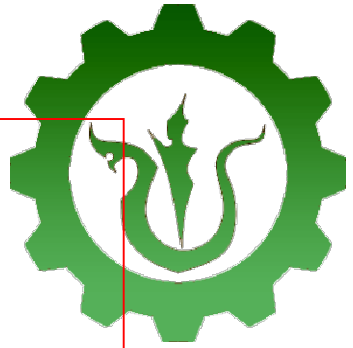
Set Interface

```
Set<String> hash_Set = new HashSet<String>();
hash_Set.add("Geeks");
hash_Set.add("For");
hash_Set.add("Geeks");
hash_Set.add("Example");
hash_Set.add("Set");
System.out.print("Set output without the duplicates");

System.out.println(hash_Set);

   // Set deonstration using TreeSet
System.out.print("Sorted Set after passing into TreeSet");
Set<String> tree_Set = new TreeSet<String>(hash_Set);
System.out.println(tree_Set);
```

Set output without the duplicates[Geeks, Example, For, Set]

Sorted Set after passing into TreeSet[Example, For, Geeks, Set]

Set Interface



```
public static void main(String args[])
    {
        Set<Integer> a = new HashSet<Integer>();
        a.addAll(Arrays.asList(new Integer[] {1, 3, 2, 4, 8, 9, 0}));
        Set<Integer> b = new HashSet<Integer>();
        b.addAll(Arrays.asList(new Integer[] {1, 3, 7, 5, 4, 0, 7, 5}));

        // To find union
        Set<Integer> union = new HashSet<Integer>(a);
        union.addAll(b);
        System.out.print("Union of the two Set");
        System.out.println(union);

        // To find intersection
        Set<Integer> intersection = new HashSet<Integer>(a);
        intersection.retainAll(b);
        System.out.print("Intersection of the two Set");
        System.out.println(intersection);

        // To find the symmetric difference
        Set<Integer> difference = new HashSet<Integer>(a);
        difference.removeAll(b);
        System.out.print("Difference of the two Set");
        System.out.println(difference);
    }
```

```
Union of the two Set[0, 1, 2, 3, 4, 5, 7, 8, 9]
Intersection of the two Set[0, 1, 3, 4]
Difference of the two Set[2, 8, 9]
```