CS209A Computer System Design & Application

lecture 2

2.1 Generics(since JDK 5.0)

Advantages:

1. No need for type cast(type-safety)

Terms:

```
List<E>; //generic type
E; //Formal type parameter
List<String>; //Parameterized type
String; //Actual type parameter
List;   //raw type(原生类型，在JDK5.0前使用)
// we should avoid using raw types
```

Using Generics

1. Generics classes

```
public class gen<T>{
    ...
    //we can use type T or any data structure with type T here
}
```

2. Generics interfaces, for example, Comparable<T>

```
public interface Comparable<T>
{
    int compareTo<T> (T o);
}
```

then we should override the method to use it

```
public myInt implements Comparable<myInt>, Comparable<myInt>{
    //实现了对不同类型的比较
    int data;
    public int compareTo<myInt>(myInt m){
        if(data == m.data){
            return 0;
        }else if(data > m.data){
            return 1;
        }else {
```

```
            return -1;
        }
    }
    public int compareTo<int>(int i){
        if(data == i){
            return 0;
        }else if(data > i){
            return 1;
        }else {
            return -1;
        }
    }
}
```

3. Generics methods

```
public static <E> Set<E> union(Set<E> s1, Set<E> s2){
    //the type parameter should be declared right after the static
    Set<E> result = new HashSet<>(s1); //type in <> can be omitted
    result.addAll(s2);
    return result;
}
```

Bounds for Type Varibles

1. form

```
<T extends superclass & interface1 & interface2>;
//use & to seperate classed and interfaces
//superclass should be the first one
```

2. application

```
public static<T extend Comparable> Pair<T> minmax(T[] a){
    ...
}
```

Wildcards

1. create a relationship between generic types

```
String is a subclass of Object
List<String> is not a subclass of List<Object>
```

for parameter-matching in the methods, classes, or interfaces

```
public staic void process(List<?> list){
    //we can pass List<String> to this method
}
```

2. set bounds for wildcards

```
//"extends" can be either direct or indirect
List<? extends superclass>;
//we can pass List<T>, where T extends superclass
List<? super subclass>;
//we can pass List<T>, where subclass extends T
```

Type Erasure

T is converted to Object.

no generics at all in JVM.

2.2 Abstract Data Type(ADT)

Primitive types:

1. **values** immediataly map to machine representations
2. **operations** immediataly map to machine representations

ADT:

1. A type for objects whose behavior is defined by a set of values and a set of operations
2. **Hide** how values are stored in memory and operations are implemented
3. clients only know the data options which can be accessed.

Operations of ADT

1. creater: create new objects of the types
2. producers: create new objects from old objects
3. observer: return a different type for the current ADT
4. Mutators: modify objects itself

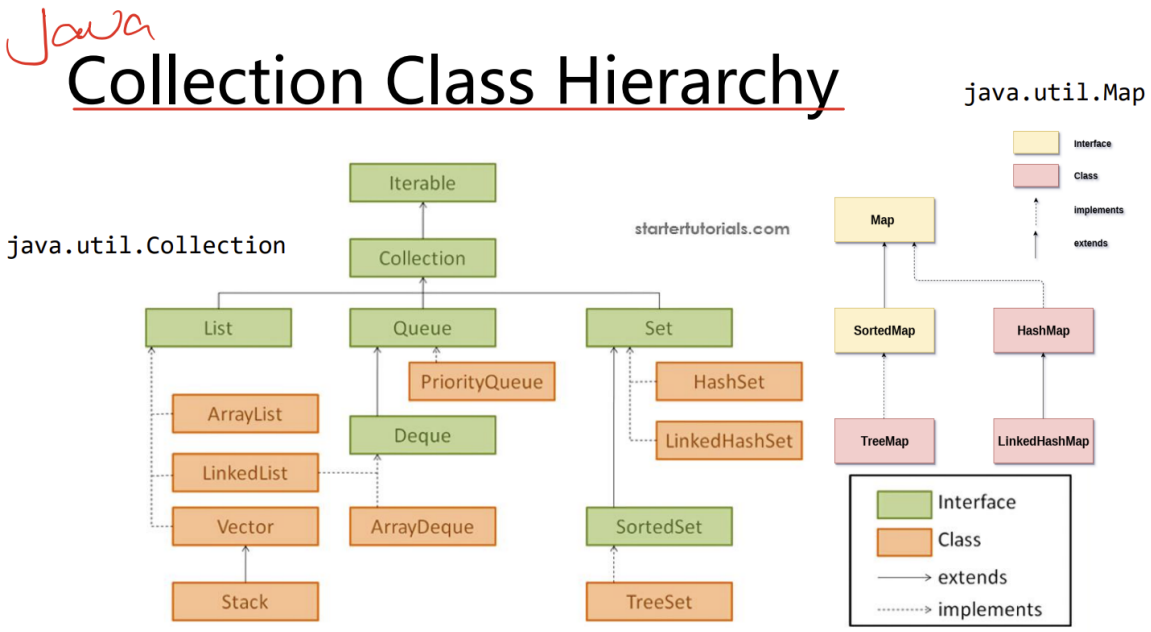
2.3.1 Collections

1. a group of objects
2. mainly used for data storage, data retrieval(检索), and data manipulation

2.3.2 Java Collections Framework

Interfaces

1. Collections Class Hierarchy



2. **Iterable<T>** interface:

```
//impletmenting this interface allows an object to be the target of the "foreach" statements
public interface Iterable<T> //可迭代的，即一定会有一个迭代器
{
    Iterator<T> iterator();
}
```

```
public interface Iterator<E>
{
    boolean hasNext();
    E next();
    void remove(); //删除目前迭代器迭代到的元素
    //注意不能连续使用两次remove，因为迭代器所迭代到的对象已经被删除
    //解决方法1：remove之后要调用iterator.next()
    //解决方法2：使用foreach,每迭代一次会自动调用迭代器方法
}
```

3. **Collection** Interface

```
public interface Collection<E>{
    int size;
    boolean isEmpty();
    boolean contains(Object element);
    //增加，删除元素不强制重写
    boolean add(E element);//optional
    boolean remove(Object element);//optional

    //从一个假头开始
    Iterator<E> iterator(); //返回该对象的迭代器
```

```
11
12      Object[] toArray(); //返回一个包含集合中所有元素的 Object 类型数组
13      /*
14      1.方法的参数 a 是用于指定数组类型和大小的数组。
15      2.如果 a 数组的长度大于等于集合的大小，那么集合中的元素将被存储在 a 数组中并返回，
16      3.否则将返回一个新的类型为 T 的数组，其中包含集合中的元素。
17      */
18      E[] toArray(E a[]);
19
20      //Bulk Operations
21      boolean containsAll(Collection<?> c); //判断是否包含
22      boolean addAll(Collection<? extends E> c); //增加一系列的而元素
23      boolean removeAll(Collection<?> c);//删除当前集合与另一个集合c中相同的元素。
24
25      /*保留集合中与另一个集合 c 相同的元素，
26      而删除集合中不在另一个集合 c 中的元素。*/
27      boolean retainAll(Collection<?> c);
28
29      void clear();//清空当前集合
30  }
```

4. **Set** interface

    1. Add no methods to Collection
    2. Add stipulation(规定)：no dulplicated elements
    3. redefine some methods of collections<E> or objects
    4. Set idioms

```
1  //for 2 sets s1, s2
2  s1.equals(s2); //比较每个元素地址/hashcode是否相等
3  s1.hashcode(); //返回每个元素的hashcode之和,这也是元素的比较核心
4  s1.containsAll(s2); //判断s2是否含于s1
5  s1.addAll(s2); //取并集并存到s1中
6  s1.retainAll(s2); //取交集并赋值给s1
7  s1.removeAll(s2); //s2 - s1
```

5. **List** interface

    1. List have **"order"**
    2. Add methods

```
1  int indexOf(Object o); //返回列表中o相同的第一个对象
2  int lastIndexOf(Object o); //返回列表中o相同的最后一个对象
3  List<E> subList(int from, int to);// 返回[from, to)的子列表
```

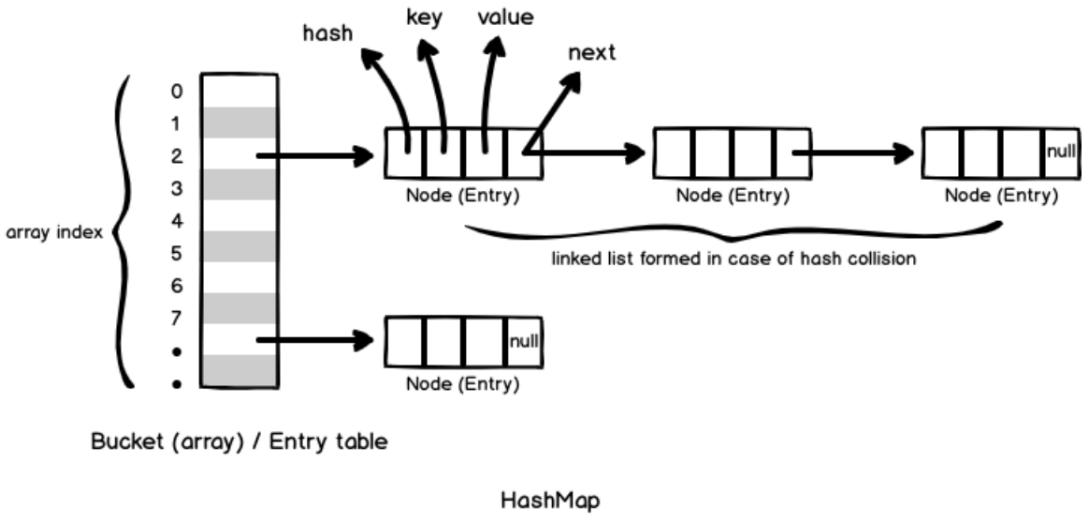
6. **Map** interface

```
1  public interface Map<K, V>{
2      int size();
3      boolean isEmpty();
```

```
    boolean containsKey(Object key); //判断是否包含一个键
    boolean containsValue(Object value); //判断是否包含值
    V get(Object key); //通过键取值
    V put(K key, V value); //存放键值对, optional
    V remove(Object key); //删除键值对并返回值，optional
    void putAll(Map<? extends K, ? extends V> t);//optional
    void clear();

    //from a collection view

    public Set<K> keySet();
    public Collection<V> values();

    //返回所有键值对组成的集合
    public Set<Map.Entry<K, V>> entrySet();
}
```

Implementations

1. **List** Implementation
    1. ArrayList interally uses an array to store the elements.
       remark: ArrayList 底层工作原理
        1. 当创建一个 `ArrayList` 对象时，会创建一个数组来存储元素，这个数组的**默认容量为 10。**当向一个已满的 `ArrayList` 中添加新的元素时，`ArrayList` 会创建一个新的数组，并将原来的元素**复制**到新数组中，然后将**新元素添加到新数组**中。
        2. 在扩容时，一般情况下增长因子的值为 **1.5。这个过程比较耗费时间和内存，因此，在创建 `ArrayList` 对象时，可以通过指定初始容量来减少扩容的次数，从而提高效率。**
        3. **如果需要频繁地进行插入、删除等操作，可以考虑使用链表实现的 `LinkedList` 类。**
2. **Map** Implementation
    1. HashMap struture



HashMap

    2. Map Implementation

```
step1: map.put(key, value);
step2: 计算key.hashcode();
step3: 通过hashcode,计算出哈希桶的下标;
step4: 判断是否发生哈希碰撞
    没有发生哈希碰撞:
        在哈希桶的对应位置链接，成为第一个节点
    发生哈希碰撞：判断key.equals(existing_key)
        相等：替换当前节点
        不相等：链接当前节点，成为下一个节点
```

```
//看看有什么不同
//FB的哈希值为2236，LD的哈希值为2236，Ea的哈希值为2236
map.put("FB", 1);
map.put("LD", 2);
map.put("Ea", 3);
System.out.println(map);
map.put("FB", 4);
System.out.println(map);
```

3. hashCode() and equals()
    1. `hashCode()` convert interal address to an int and return
    2. `equals(Objected)` compared hashcode by default.
    3. `==` always compared hashcode