Advanced Programming - JAVA
Lecture 5
JAVA Collections

Mahmoud El-Gayyar
elgayyar@ci.suez.edu.eg

Outline

- *Arrays*
  - Working with arrays
  - Java API support for arrays
- *Collections*
  - Types of collection (ArrayList, HashMap)
  - Working with Collections

Outline

- ***Arrays***
  - Working with arrays
  - Java API support for arrays
- *Collections*
  - Types of collection (ArrayList, HashMap)
  - Working with Collections

Java Arrays – The Basics

- *Declaring an array*

```
int[] myArray;
int[] myArray = new int[5];
String[] stringArray = new String[10];
String[] strings = new String[] {“one”, “two”};
```

- *Checking an arrays length*

```
int arrayLength = myArray.length;
```

- *Looping over an array*

```
for(int i=0; i<strings.length; i++){
   String s = myArray[i];
}
for(String s:strings){…}
```

Java Arrays – Bounds Checking

- *Bounds checking*
  - Java does this automatically. Impossible to go beyond the end of an array (unlike C/C++)
  - Automaticallygenerates an `ArrayIndexOutOfBoundsException (Runtime)`

Java Arrays – Copying

- *Don't copy arrays "by hand" by looping over the array*
- *The System class has an `arrayCopy` method to do this efficiently*

```
int array1[] = new int[10];
int array2[] = new int[10];
//assume we add items to array1

//copy array1 into array2
System.arrayCopy(array1, 0, array2, 0, 10);
//copy last 5 elements in array1 into first 5 of array2
System.arrayCopy(array1, 5, array2, 0, 5);
```

Java Arrays – Sorting

- *Again no need to do this "by hand".*
- *The java.util.Arrays class has methods to sort different kinds of arrays*

```
int myArray[] = new int[] {5, 4, 3, 2, 1};
java.util.Arrays.sort(myArray);
//myArray now holds 1, 2, 3, 4, 5
```

- *Sorting arrays of objects is involves some extra work?!!*
  - Comparable objects

Java Arrays

- *Advantages*
  - Very efficient, quick to access and add to
  - Type-safe, can only add items that match the declared type of the array
- *Disadvantages*
  - Fixed size, some overhead in copying/resizing
  - Can't tell how many items in the array, just how large it was declared to be

Outline

- *Arrays*
  - Working with arrays
  - Java API support for arrays
- ***Collections***
  - Types of collection (ArrayList, HashMap)
  - Working with Collections

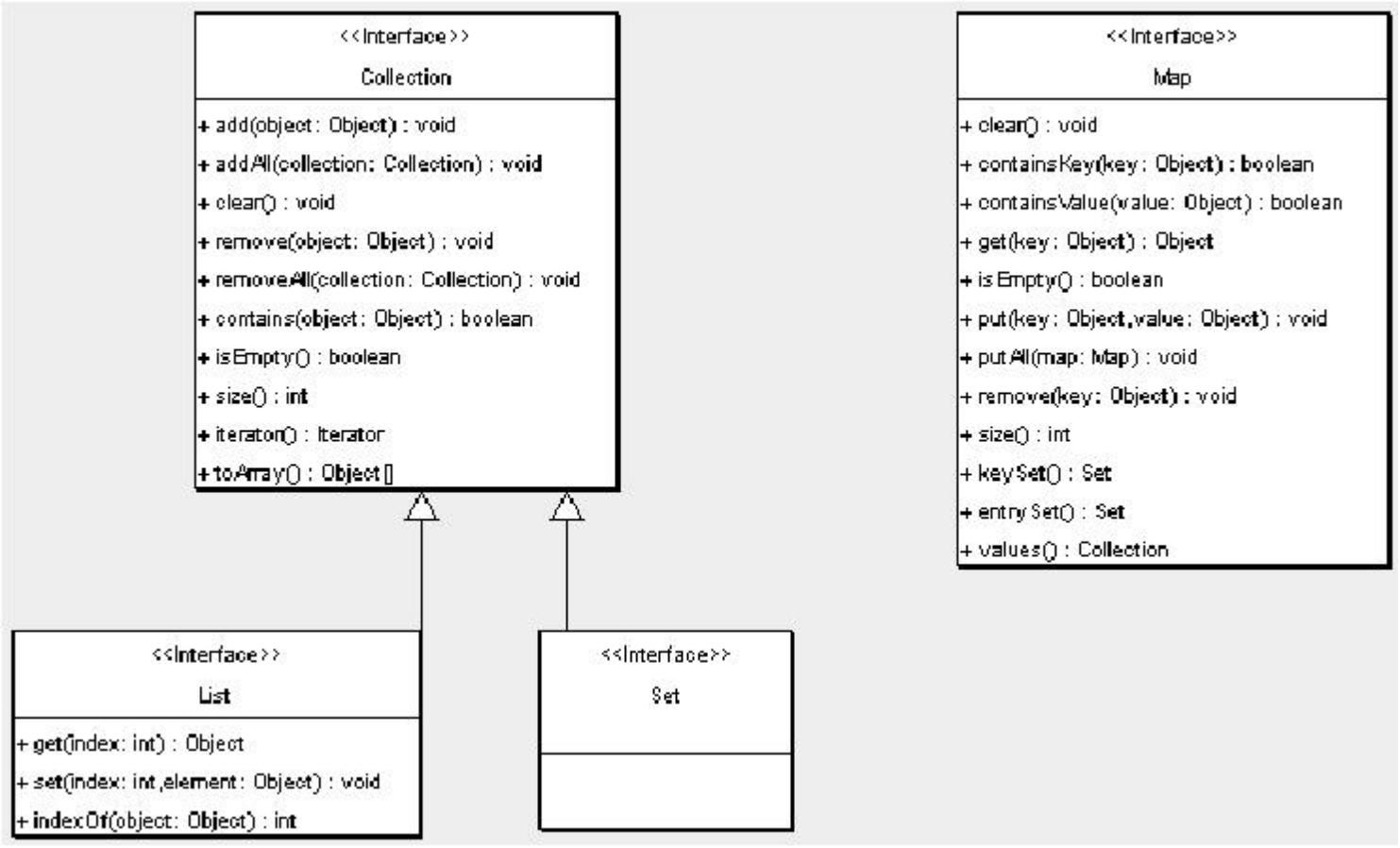
Java Collections

- *What are they?*
  - A number of pre-packaged implementationsof common ‘container’ classes, such as LinkedLists, Sets, etc.
  - Part of the `java.util` package.
- *Advantages*
  - Very flexible, can hold any kind of object
- *Disadvantages*
  - Not as efficient as arrays (for some uses)
  - Not type-safe. Store references to `Object` → ***Use Generics***

Java Collections

- *Two Types of Containers*
- *Collections*
  - Group of objects, which may restricted or manipulated in some way
  - E.g. an ordered to make a List or LinkedList
  - E.g. a Set, an unordered group which can only contain one of each item
- *Maps*
  - Associative array, Dictionary, Lookup Table, Hash
  - A group of name-value pairs

Java Collections



Java Collections

- *Several implementations associated with each of the basic interfaces*
- *Each has its own advantages/disadvantages*
- *Maps*
  - HashMap, SortedMap
- *Lists*
  - ArrayList, LinkedList
- *Sets*
  - HashSet, SortedSet

Java Collections – The Basics

- *HashMap and ArrayList are most commonly encountered*
- *Usual object creation syntax*
- *Generally hold references to the interface and not the specific collection*
  - Can then process them generically

```
List myList = new ArrayList();
List otherList = new ArrayList(5);
Map database = new HashMap();
```

Java Collections – Adding Items

- *For Collections, use* `add()`

```
List myList = new ArrayList();
myList.add("A String");
myList.add("Other String");
```

- *For Maps, use* `put()`

```
Map myMap = new HashMap();
myMap.put("google", "http://www.google.com");
mpMap.put("yahoo", "http://www.yahoo.com");
```

Java Collections – Copying

- *Very easy, just use addAll()*

```
List myList = new ArrayList();
//assume we add items to the list

List otherList = new ArrayList();
otherList.addAll(myList);
```

Collections – Getting Individual Items

- *Use `get()`*
- *Note that we have to cast the object to its original type.*
- *Collections…*

```
String s = (String)myList.get(1); //get  first element
String s2 = (String)myList.get(10); //get tenth  element
```

- *Maps…*

```
String s =  (String)myMap.get("google");
String s2 = (String)mpMap.get("yahoo");
```

Collections – Getting all items

- *For Lists, we could use a `for` loop, and loop through the list to `get()` each item*
- *But this doesn't work for Maps.*
- *To allow generic handling of collections, Java defines an object called an Iterator*
  - An object whose function is to walk through a Collection of objects and provide access to each object in sequence

Collections – Getting all items

- *Get an iterator using the `iterator()` method*
- *Iterator objects have three methods:*
  - `next()` – gets the next item in the collection
  - `hasNext()` – tests whether it has reached the end
  - `remove()` – removes the item just returned

Collections – Getting all items

- *Simple example:*

```
List myList = new ArrayList();
//we add items

Iterator iterator = myList.iterator();
while (iterator.hasNext())
{
  String s = (String)iterator.next();
  //do something with it
}
```

Example 1 – Array List

```
static public void main(String[] args) {
    ArrayList argsList = new ArrayList();
    for(String str : args) {
        argsList.add(str);
    }
    if(argsList.contains("Koko")) {
       System.out.println("We have Koko");
    }
    String first = (String)argsList.get(0);
    System.out.println("First: " + first);
}
```

Example 2 – ArrayList with Generics

```
static public void main(String[] args) {
    ArrayList<String> argsList =
              new ArrayList<String>();
    for(String str : args) {
        argsList.add(str); // argsList.add(7) would fail
    }
    if(argsList.contains("Koko")) {
       System.out.println("We have Koko");
    }
    String first = argsList.get(0); // no casting!
    System.out.println("First: " + first);
}
```

Example – HashMap

```
public static void main(String args[]) {
    // Create a hash map
    HashMap <String, Double> hm = new HashMap<String, Double>();
    // Put elements to the map
    hm.put("Ali", new Double(3434.34));
    hm.put("Mohamed", new Double(123.22));
    hm.put("Nasser", new Double(1378.00));

    Set set = hm.entrySet();          // Get the entries
    Iterator i = set.iterator();      // Get an iterator
    // Display elements
    while(i.hasNext()) {
       Entry me = (Entry)i.next();
       System.out.print(me.getKey() + ": " +me.getValue());
    }
    // Deposit 1000 into Ali's account
    double balance = hm.get("Ali").doubleValue();
    hm.put("Ali", new Double(balance + 1000));
    System.out.println("Ali's new balance: " +
                       hm.get("Ali"));
}
```

Collections – Other Functions

- *The `java.util.Collections` class has many useful methods for working with collections*
  - min, max, sort, reverse, search, shuffle
- *Virtually all require your objects to implement an extra interface, called `Comparable`*

Summary

- *Arrays*
  - Working with arrays
  - Java API support for arrays
- *Collections*
  - Types of collection (ArrayList, HashMap)
  - Working with Collections