36. Collections

Java

Summer 2008

Instructor: Dr. Masoud Yaghini

Outline

- Introduction
- Arrays Class
- Interface Collection and Class Collections
- ArrayList Class
- Generics
- LinkedList Class
- Collections Algorithms
- Stack Class
- Class PriorityQueue and Interface Queue
- Sets Class
- Maps Class
- References

Introduction

Introduction

- Java collections framework
  - Contain prepackaged data structures, interfaces, algorithms for manipulating those data structures
  - With collections, programmers use existing data structures, without concern for how they are implemented.
  - This is a example of code reuse.
  - Programmers can code faster and can expect excellent performance, maximizing execution speed and minimizing memory consumption.

Introduction

- Collection
  - Data structure (object) that can hold references to other objects
- Collections framework
  - Interfaces declare operations for various collection types
  - Provide high-performance, high-quality implementations of common data structures
  - Enable software reuse

Some collection framework interfaces

- Collection
  - The root interface in the collections hierarchy from which interfaces Set, Queue and List are derived.
- Set
  - A collection that does not contain duplicates.
- List
  - An ordered collection that can contain duplicate elements.
- Map
  - Associates keys to values and cannot contain duplicate keys.
- Queue
  - Typically a first-in, first-out collection that models a waiting line; other orders can be specified.

Arrays Class

Class Arrays

- Class Arrays
  - Provides static methods for manipulating arrays
  - Provides “high-level” methods
    - Method binarySearch for searching sorted arrays
    - Method equals for comparing arrays
    - Method fill for placing values into arrays
    - Method sort for sorting arrays

Class Arrays

- The program:
  - UsingArrays.java demonstrates methods fill, sort, binarySearch and equals.
  - Method main creates a UsingArrays object and invokes its methods.

Class Arrays

- The program output:

```
doubleArray: 0.2 3.4 7.9 8.4 9.3
intArray: 1 2 3 4 5 6
filledIntArray: 7 7 7 7 7 7 7 7 7 7
intArrayCopy: 1 2 3 4 5 6

intArray == intArrayCopy
intArray != filledIntArray
Found 5 at element 4 in intArray
8763 not found in intArray
```

Common Programming Error

- Passing an unsorted array to binarySearch is a logic error—the value returned is undefined.

Interface Collection and Class Collections

Interface Collection

- Interface Collection
  - Root interface in the collection hierarchy
  - Interfaces Set, Queue, List extend interface Collection
    - List – ordered collection can contain duplicate elements
    - Set – collection does not contain duplicates
    - Queue – collection represents a waiting line
  - Contains bulk operations
    - Adding, clearing, comparing and retaining objects
  - Provide method to return an Iterator object
    - Walk through collection and remove elements from collection

Iterator

- It is common in object-oriented programming to declare an iterator class that can traverse all the objects in a collection, such as an array or an ArrayList.
- For example, a program can print an ArrayList of objects by creating an iterator object and using it to obtain the next list element each time the iterator is called.
- Iterators often are used in polymorphic programming to traverse a collection that contains references to objects from various levels of a hierarchy.

Class Collections

- Class Collections
  - Provides static methods that manipulate collections
    - Implement algorithms for searching, sorting and so on
  - Collections can be manipulated polymorphically

ArrayList Class

Lists

- List
  - A list is an ordered Collection that can contain duplicate elements
  - Sometimes called a **sequence**
  - List indices are zero based (i.e., the first element's index is zero)
  - Classes
    - ArrayList: is resizable-array
    - LinkedList : is resizable-array

Lists

- ArrayListTest.java
  - Demonstrate Collection interface capabilities
  - Place two String arrays in ArrayLists
  - Use Iterator to remove elements in ArrayList

- The program output:

```
ArrayList:
MAGENTA RED WHITE BLUE CYAN

ArrayList after calling removeColors:
MAGENTA CYAN
```

Common Programming Error

- If a collection is modified by one of its methods after an iterator is created for that collection, the iterator immediately becomes invalid—any operations performed with the iterator after this point throw ConcurrentModificationExceptions.
- For this reason, iterators are said to be “fail fast.”

Generics

Overloaded methods

- Overloaded methods
  - Perform similar operations on different types of data
  - For example an overloaded methods
    - Integer array
    - Double array
    - Character array

Generics

- It would be nice if we could write a single sort method that could sort the elements in an Integer array, a String array or an array of any type that supports ordering (i.e., its elements can be compared).
- It would also be nice if we could write a single Stack class that could be used as a Stack of integers, a Stack of floating-point numbers, a Stack of Strings or a Stack of any other type.

Generics

- It would be even nicer if we could detect type mismatches at compile timeknown as compile-time type safety.
- For example, if a Stack stores only integers, attempting to push a String on to that Stack should issue a compile-time error.
- **Generics** provides the means to create the general models mentioned above.

Generics

- Generics
  - Provide compile-time type safety
    - Catch invalid types at compile time
  - Generic methods
    - A single method declaration
  - Generic classes
    - A single class declaration

Generics

- Note that ArrayList is a generic class, so we are able to specify a type argument (String in this case) to indicate the type of the elements in each list.

- ArrayListTest2.java
  - Demonstrates how to specify a type argument for generic ArrayList class

LinkedList Class

LinkedList

- LinkedLists can be used to create stacks, queues, trees and deques (double-ended queues, pronounced “decks”).
- The collections framework provides implementations of some of these data structures.

LinkedList

- ListTest.java
  - The program creates two LinkedLists that contain Strings.
  - The elements of one List are added to the other.
  - Then all the Strings are converted to uppercase, and
  - a range of elements is deleted

LinkedList

- The program output:

```
list1:
black yellow green blue violet silver gold white brown blue gray silver

list1:
BLACK YELLOW GREEN BLUE VIOLET SILVER GOLD WHITE
    BROWN BLUE GRAY SILVER

Deleting elements 4 to 6...
list1:
BLACK YELLOW GREEN BLUE WHITE BROWN BLUE GRAY SILVER

Reversed List:
SILVER GRAY BLUE BROWN WHITE BLUE GREEN YELLOW BLACK
```

LinkedList

- `static` method `asList` of class `Arrays`
  - View an array as a `List` collection
  - Allow programmer to manipulate the array as if it were a list
  - Any modification made through the `List` view change the array
  - Any modification made to the array change the `List` view

LinkedList

- UsingToArray.java
  - The program calls method asList to create a List view of an array, which is then used for creating a LinkedList object,
  - adds a series of strings to a LinkedList and
  - calls method toArray to obtain an array containing references to the strings.
  - Notice that the instantiation of LinkedList indicates that LinkedList is a generic class that accepts one type argumentString, in this example.

LinkedList

- The program output:

```
colors:
cyan
black
blue
yellow
green
red
pink
```

Common Programming Error

- If the number of elements in the array is smaller than the number of elements in the list on which toArray is called, a new array is allocated to store the list's elements
- If the number of elements in the array is greater than the number of elements in the list, the elements of the array (starting at index zero) are overwritten with the list's elements. Array elements that are not overwritten retain their values.

Collections Algorithms

Collections Algorithms

- Collections framework provides set of algorithms, implemented as static methods
- Algorithms operate on List :
  - sort
    - Sorts the elements of a List.
  - binarySearch
    - Locates an object in a List.
  - reverse
    - Reverses the elements of a List.
  - shuffle
    - Randomly orders a List's elements.
  - fill
    - Sets every List element to refer to a specified object.
  - copy
    - Copies references from one List into another.

Collections Algorithms

- Algorithms operate on any Collections:
  - min
    - Returns the smallest element in a Collection.
  - max
    - Returns the largest element in a Collection.
  - addAll
    - Appends all elements in an array to a collection.
  - frequency
    - Calculates how many elements in the collection are equal to the specified element.
  - disjoint
    - Determines whether two collections have no elements in common.

Algorithm sort

- sort
  - Sorts List elements
    - Order is determined by natural order of elements' type
    - List elements must implement the Comparable interface
    - Or, pass a Comparator to method sort
- Sorting in ascending order
  - Collections method sort
- Sorting in descending order
  - Collections static method reverseOrder
- Sorting with a Comparator
  - Create a custom Comparator class

Algorithm sort

- Sort1.java
  - uses algorithm sort to order the elements of a List in ascending order.
  - Recall that List is a generic type and accepts one type argument that specifies the list element type

Algorithm sort

- The program output:

  Unsorted array elements:

  [Hearts, Diamonds, Clubs, Spades]

  Sorted array elements:

  [Clubs, Diamonds, Hearts, Spades]

Sorting in Descending Order

- Sort2.java
  - sorts the same list of strings in descending order.
  - The example introduces the Comparator interface, which is used for sorting a Collection's elements in a different order.
  - The static Collections method reverseOrder returns a Comparator object that orders the collection's elements in reverse order.

Sorting in Descending Order

- The program output:

  Unsorted array elements:

  [Hearts, Diamonds, Clubs, Spades]

  Sorted list elements:

  [Spades, Hearts, Diamonds, Clubs]

Algorithm shuffle

- shuffle
  - Randomly orders List elements

- ShuffleTest.java
  - In this program we use algorithm shuffle to shuffle a deck of Card objects that might be used in a card game simulator.

Collections Algorithms

- The program output:

  Array elements:

  [Hearts, Diamonds, Clubs, Spades]

  Shuffled list elements:

  [Spades, Clubs, Diamonds, Hearts]

Algorithm reverse, fill, copy, max and min

- reverse
  - Reverses the order of List elements
- Fill
  - Overwrites elements in a List with a specified value.
  - The fill operation is useful for reinitializing a List.
- copy
  - Creates copy of a List
  - takes two arguments a destination List and a source List
  - Each source List element is copied to the destination List
  - The destination List must be at least as long as the source List; otherwise, an IndexOutOfBoundsException occurs.
  - If the destination List is longer, the elements not overwritten

Algorithm reverse, fill, copy, max and min

- max
  - Returns largest element in List
  - Operate on any Collection
- min
  - Returns smallest element in List
  - Operate on any Collection

- Algorithms1.java
  - demonstrates the use of algorithms reverse, fill, copy, min and max. Note that the generic type List is declared to store Characters.

Algorithm reverse, fill, copy, max and min

- The program output:

```
Initial list:
The list is: P C M
Max: P  Min: C

After calling reverse:
The list is: M C P
Max: P  Min: C

After copying:
The list is: M C P
Max: P  Min: C

After calling fill:
The list is: R R R
Max: R  Min: R
```

Algorithm binarySearch

- The binarySearch algorithm locates an object in a List (i.e., a LinkedList or an ArrayList).
- If the object is found, its index is returned. If the object is not found, binarySearch returns a negative value.
- Algorithm binarySearch determines this negative value by first calculating the insertion point and making its sign negative.
- Then, binarySearch subtracts 1 from the insertion point to obtain the return value, which guarantees that method binarySearch returns positive numbers (>=0) if and only if the object is found.

Algorithm binarySearch

- If multiple elements in the list match the search key, there is no guarantee which one will be located first.
- BinarySearchTest.java uses the binarySearch algorithm to search for a series of strings in an ArrayList.

Algorithm binarySearch

- The program output:

```
Sorted list: [black, blue, pink, purple, red, tan, white, yellow]

Searching for: black
Found at index 0

Searching for: red
Found at index 4

Searching for: pink
Found at index 2

Searching for: aqua
Not Found (-1)

Searching for: gray
Not Found (-3)

Searching for: teal
Not Found (-7)
```

Algorithms addAll, frequency and disjoint

- addAll
  - Insert all elements of an array into a collection
  - Takes two arguments, a Collection into which to insert the new element(s) and an array that provides elements to be inserted
- frequency
  - Calculate the number of times a specific element appear in the collection
  - Takes two arguments a Collection to be searched and an Object to be searched for in the collection
- Disjoint
  - Algorithm disjoint takes two Collections and returns true if they have no elements in common

Algorithms addAll, frequency and disjoint

- Algorithms2.java
  - demonstrates the use of algorithms addAll, frequency and disjoint.

Algorithms addAll, frequency and disjoint

- The program output:

```
Before addAll, list2 contains:
black red green

After addAll, list2 contains:
black red green red white yellow blue

Frequency of red in list2: 2

list1 and list2 have elements in common
```

Stack Class

Stack Class

- Stack class in the Java utilities package java.util implements stack data structure
- Class Stack stores references to objects
- Autoboxing occurs when you add a primitive type to a Stack
- Class Stack extends class Vector to implement a stack data structure.

- StackTest.java
  - demonstrates several Stack methods.

Stack Class

- The program output:

```
stack contains: 12 (top)

stack contains: 12 34567 (top)

stack contains: 12 34567 1.0 (top)

stack contains: 12 34567 1.0 1234.5678 (top)

1234.5678 popped
stack contains: 12 34567 1.0 (top)

1.0 popped
stack contains: 12 34567 (top)
```

Stack Class

- The constructor creates an empty Stack of type Number.
- Class Number (in package java.lang) is the superclass of most wrapper classes (e.g., Integer, Double) for the primitive types.
- By creating a Stack of Number, objects of any class that extends the Number class can be pushed onto the stack.

Stack Class

- Any integer literal that has the suffix L is a long value.
- An integer literal without a suffix is an int value.
- Similarly, any floating-point literal that has the suffix F is a float value.
- A floating-point literal without a suffix is a double value.

Stack Class

- Because Stack extends Vector, all public Vector methods can be called on Stack objects, even if the methods do not represent conventional stack operations.
- For example, Vector method add can be used to insert an element anywhere in a stack—an operation that could “corrupt” the stack.
- When manipulating a Stack, only methods push and pop should be used to add elements to and remove elements from the Stack, respectively.

Class PriorityQueue and Interface Queue

Class PriorityQueue and Interface Queue

- Interface Queue,
  - extends interface Collection and provides additional operations for inserting, removing and inspecting elements in a queue.
- Class PriorityQueue,
  - one of the classes that implements the Queue interface, orders elements by their natural ordering
  - When adding elements to a PriorityQueue, the elements are inserted in priority order such that the highest-priority element (i.e., the largest value) will be the first element removed from the PriorityQueue.

Class PriorityQueue and Interface Queue

- The common PriorityQueue operations are
  - **offer** to insert an element at the appropriate location based on priority order
    - Method offer throws a NullPointException if the program attempts to add a null object to the queue.
  - **poll** to remove the highest-priority element of the priority queue (i.e., the head of the queue),
  - **peek** to get a reference to the highest-priority element of the priority queue (without removing that element),
  - **clear** to remove all elements in the priority queue and
  - **size** to get the number of elements in the priority queue.

Class PriorityQueue and Interface Queue

- PriorityQueueTest.java
  - demonstrates the PriorityQueue class.

- The program output:

  Polling from queue: 3.2 5.4 9.8

Sets Class

Sets Class

- A Set is a Collection that contains unique elements (i.e., no duplicate elements), including:
  - HashSet
    - Stores elements in hash table
  - TreeSet
    - Stores elements in tree

Sets Class

- SetTest.java
  - Recall that both List and Collection are generic types, so this program creates a List that contains String objects, and
  - It passes a Collection of Strings to method printNonDuplicates.

- The program output:

```
ArrayList: [red, white, blue, green, gray, orange, tan, white, cyan,
    peach, gray, orange]

Nonduplicates are:
orange green white peach gray cyan red blue tan
```

Sets Class

- The collections framework also includes interface SortedSet (which extends Set) for sets that maintain their elements in sorted order either the elements' natural order (e.g., numbers are in ascending order) or an order specified by a Comparator.
- Class treeSet implements SortedSet.

Maps Class

Maps Class

- Maps associate keys to values
- Maps cannot contain duplicate keys, i.e., each key can map to only one value; this is called **one-to-one mapping**.
- Maps differ from Sets in that Maps contain keys and values, whereas Sets contain only values.

Maps Class

- Three of the several classes that implement interface Map are:
  - Hashtable
    - store elements in hash tables
  - HashMap
    - store elements in hash tables
  - TreeMap
    - Store elements in trees
- This section discusses hash tables and provides an example that uses a HashMap to store key/value pairs.

Maps Class

- Interface SortedMap extends Map and maintains its keys in sorted order either the elements' natural order or an order specified by a Comparator.
- Class TreeMap implements SortedMap.

Maps Class

- Map implementation with hash tables
  - Hash tables
    - Data structure that use hashing (convert a key into an array index)
    - Algorithm for determining a key in table
    - Keys in tables have associated values (data)

Maps Class

- WordTypeCount.java
  - uses a HashMap to count the number of occurrences of each word in a string.

Maps Class

- The program Output:

```
Enter a string:
To be or not to be: that is the question
Map contains:
Key             Value
be              1
be:             1
is              1
not             1
or              1
question        1
that            1
the             1
to              2

size:9
isEmpty:false
```

References

References

- H. M. Deitel and P. J. Deitel, **Java™ How to Program**, Sixth Edition, Prentice Hall, 2005. (Chapter 19)

The End