

Session 6

**Collections**

# Contents

- **the java.util package**
  - **Arrays Class**
  - **Collections**

# 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

```

1 // Fig. 21.1: UsingArrays.java
2 // Using Java arrays.
3
4 // Java core packages
5 import java.util.*;
6
7 public class UsingArrays {
8     private int intValues[] = { 1, 2, 3, 4, 5, 6 };
9     private double doubleValues[] = { 8.4, 9.3, 0.2, 7.9, 3.4 };
10    private int filledInt[], intValuesCopy[];
11
12    // initialize arrays
13    public UsingArrays()
14    {
15        filledInt = new int[ 10 ];
16        intValuesCopy = new int[ intValues.length ];
17
18        Arrays.fill( filledInt, 7 ); // fill with 7s
19
20        Arrays.sort( doubleValues ); // sort doubleValues
21
22        System.arraycopy( intValues, 0, intValuesCopy,
23                          0, intValues.length );
24    }
25
26    // output values in each array
27    public void printArrays()
28    {
29        System.out.print( "doubleValues: " );
30
31        for ( int count = 0; count < doubleValues.length; count++ )
32            System.out.print( doubleValues[ count ] + " " );
33
34        System.out.print( "\nintValues: " );
35

```

Use static method  
fill of class Arrays  
to populate array with

7s

Use static method  
sort of class Arrays  
to sort array's  
elements in ascending

order

Use static method arraycopy of  
class System to copy array  
intValues into array

intValuesCopy

```

36     for ( int count = 0; count < intValues.length; count++ )
37         System.out.print( intValues[ count ] + " " );
38
39     System.out.print( "\nfilledInt: " );
40
41     for ( int count = 0; count < filledInt.length; count++ )
42         System.out.print( filledInt[ count ] + " " );
43
44     System.out.print( "\nintValuesCopy: " );
45
46     for ( int count = 0; count < intValuesCopy.length; count++ )
47         System.out.print( intValuesCopy[ count ] + " " );
48
49     System.out.println();
50 }
51
52 // find value in array intValues
53 public int searchForInt( int value )
54 {
55     return Arrays.binarySearch( intValues, value );
56 }
57
58 // compare array contents
59 public void printEquality()
60 {
61     boolean b = Arrays.equals( intValues, intValuesCopy );
62
63     System.out.println( "intValues " + ( b ? "==" : "!=" )
64         + " intValuesCopy" );
65
66     b = Arrays.equals( intValues, filledInt );
67
68     System.out.println( "intValues " + ( b ? "==" : "!=" )
69         + " filledInt" );
70 }

```

Use static method  
**binarySearch** of class **Arrays**  
to perform binary search on  
array

Use static method  
**equals** of class **Arrays** to  
determine whether values  
of the two arrays are  
equivalent

```

71
72 // execute application
73 public static void main( String args[] )
74 {
75     UsingArrays usingArrays = new UsingArrays();
76
77     usingArrays.printArrays();
78     usingArrays.printEquality();
79
80     int location = usingArrays.searchForInt( 5 );
81     System.out.println( ( location >= 0 ?
82         "Found 5 at element " + location : "5 not found" ) +
83         " in intValues" );
84
85     location = usingArrays.searchForInt( 8763 );
86     System.out.println( ( location >= 0 ?
87         "Found 8763 at element " + location :
88         "8763 not found" ) + " in intValues" );
89 }
90
91 } // end class UsingArrays

```

```

doubleValues: 0.2 3.4 7.9 8.4 9.3
intValues: 1 2 3 4 5 6
filledInt: 7 7 7 7 7 7 7 7 7 7
intValuesCopy: 1 2 3 4 5 6
intValues == intValuesCopy
intValues != filledInt
Found 5 at element 4 in intValues
8763 not found in intValues

```

```

1 // Fig. 21.2: UsingAsList.java
2 // Using method asList
3
4 // Java core packages
5 import java.util.*;
6
7 public class UsingAsList {
8     private String values[] = { "red", "white", "blue" };
9     private List list;
10
11     // initialize List and set value at location 1
12     public UsingAsList()
13     {
14         list = Arrays.asList( values ); // get List
15         list.set( 1, "green" ); // change a value
16     }
17
18     // output List and array
19     public void printElements()
20     {
21         System.out.print( "List elements : " );
22
23         for ( int count = 0; count < list.size(); count++ )
24             System.out.print( list.get( count ) + " " );
25
26         System.out.print( "\nArray elements: " );
27
28         for ( int count = 0; count < values.length; count++ )
29             System.out.print( values[ count ] + " " );
30
31         System.out.println();
32     }
33

```

Use static method  
**asList** of class **Arrays** to  
return **List** view of array

**values**

Use method **set** of **List**  
object to change the  
contents of element 1 to

**"green"**

**List** method **size**  
returns number of  
elements in **List**

**List** method **get** returns  
individual element in

**List**

```
34 // execute application
35 public static void main( String args[] )
36 {
37     new UsingAsList().printElements();
38 }
39
40 } // end class UsingAsList
```

```
List elements : red green blue
Array elements: red green blue
```



# Introduction

- Java collections framework
  - Provides reusable component
  - Existing data structures
    - Example of code reuse

# Collections Overview

- Collection
  - Data structure (object) that can hold other objects
- Collections framework
  - Interfaces that define operations for various collection types
  - Belong to package **java.util**
    - **Collection**
    - **Set**
    - **List**
    - **Map**

# Interface Collection and Class Collections

- Interface **Collection**
  - Contains *bulk operations*
    - Adding, clearing, comparing and retaining objects
  - Interfaces **Set** and **List** extend interface **Collection**
- Class **Collections**
  - Provides **static** methods that manipulate collections
  - Collections can be manipulated polymorphically

# Lists

- List
  - Ordered **Collection** that can contain duplicate elements
  - Sometimes called a *sequence*
  - Implemented via interface **List**
    - **ArrayList**
    - **LinkedList**
    - **Vector**

```

1 // Fig. 21.3: CollectionTest.java
2 // Using the Collection interface
3
4 // Java core packages
5 import java.awt.Color;
6 import java.util.*;
7
8 public class CollectionTest {
9     private String colors[] = { "red", "white", "blue" };
10
11     // create ArrayList, add objects to it and manipulate it
12     public CollectionTest()
13     {
14         ArrayList list = new ArrayList();
15
16         // add objects to list
17         list.add( Color.magenta ); // add a color object
18
19         for ( int count = 0; count < colors.length; count++ )
20             list.add( colors[ count ] );
21
22         list.add( Color.cyan ); // add a color object
23
24         // output list contents
25         System.out.println( "\nArrayList: " );
26
27         for ( int count = 0; count < list.size(); count++ )
28             System.out.print( list.get( count ) + " " );
29
30         // remove all String objects
31         removeStrings( list );
32
33         // output list contents
34         System.out.println( "\n\nArrayList after calling"
35             " removeStrings: " );

```

Use List method add to  
add objects to  
ArrayList

List method get returns  
individual element in  
List

Method removeStrings takes a  
Collection as an argument; Line  
31 passes List, which extends  
Collection, to this method

```

36
37     for ( int count = 0; count < list.size(); count++ )
38         System.out.print( list.get( count ) + " " );
39     }
40
41     // remove String objects from Collection
42     public void removeStrings( Collection collection )
43     {
44         // get iterator
45         Iterator iterator = list.iterator();
46
47         // loop while collection has items
48         while ( iterator.hasNext() )
49         {
50             if ( iterator.next() instanceof String )
51                 iterator.remove(); // remove String object
52         }
53
54         // execute application
55         public static void main( String args[] )
56         {
57             new CollectionTest();
58         }
59
60     } // end class CollectionTest

```

Obtain Collection

iterator

Iterator method hasNext  
determines whether the  
Iterator contains more  
elements

Iterator method next returns  
next Object in Iterator

Use Iterator method remove  
to remove String from

Iterator

ArrayList:

```

java.awt.Color[r=255,g=0,b=255] red white blue java.awt.Color
[r=0,g=255,b=255]

```

ArrayList after calling removeStrings:

```

java.awt.Color[r=255,g=0,b=255] java.awt.Color[r=0,g=255,b=255]

```

```

1 // Fig. 21.4: ListTest.java
2 // Using LinkLists
3
4 // Java core packages
5 import java.util.*;
6
7 public class ListTest {
8     private String colors[] = { "black", "yellow", "green",
9         "blue", "violet", "silver" };
10    private String colors2[] = { "gold", "white", "brown",
11        "blue", "gray", "silver" };
12
13    // set up and manipulate LinkedList objects
14    public ListTest()
15    {
16        LinkedList link = new LinkedList();
17        LinkedList link2 = new LinkedList();
18
19        // add elements to each list
20        for ( int count = 0; count < colors.length; count++ ) {
21            link.add( colors[ count ] );
22            link2.add( colors2[ count ] );
23        }
24
25        link.addAll( link2 ); // concatenate lists
26        link2 = null;        // release resources
27
28        printList( link );
29
30        uppercaseStrings( link );
31
32        printList( link );
33
34        System.out.print( "\nDeleting elements 4 to 6..." );
35        removeItems( link, 4, 7 );

```

Create two LinkedList  
objects

Use LinkedList method  
addAll to append link2  
elements to link

Nullify link2, so it can  
be garbage collected

36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70

```
    printList( link );  
}  
  
// output List contents  
public void printList( List list )  
{  
    System.out.println( "\nlist: " );  
  
    for ( int count = 0; count < list.size(); count++ )  
        System.out.print( list.get( count ) + " " );  
  
    System.out.println();  
}  
  
// locate String objects and convert to upper  
public void uppercaseStrings( List list )  
{  
    ListIterator iterator = list.listIterator();  
  
    while ( iterator.hasNext() ) {  
        Object object = iterator.next(); // get item  
  
        if ( object instanceof String ) // check for String  
            iterator.set(  
                ( ( String ) object ).toUpperCase()  
            );  
    }  
}  
  
// obtain sublist and use clear method to delete sublist items  
public void removeItems( List list, int start, int end )  
{  
    list.subList( start, end ).clear(); // remove items  
}
```

Use List method get to obtain object in LinkedList, then print its value

Use ListIterator to traverse LinkedList elements and convert them to upper case (if elements are Strings)

Use List method subList and clear methods to remove LinkedList elements



```
71 // execute application
72 public static void main( String args[] )
73 {
74     new ListTest();
75 }
76
77 } // end class ListTest
```

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

```
list:
BLACK YELLOW GREEN BLUE VIOLET SILVER GOLD WHITE BROWN BLUE GRAY SILVER
```

Deleting elements 4 to 6...

```
list:
BLACK YELLOW GREEN BLUE WHITE BROWN BLUE GRAY SILVER
```

```

1  // Fig. 21.5: UsingToArray.java
2  // Using method toArray
3
4  // Java core packages
5  import java.util.*;
6
7  public class UsingToArray {
8
9      // create LinkedList, add elements and convert to array
10     public UsingToArray()
11     {
12         LinkedList links;
13         String colors[] = { "black", "blue", "yellow" };
14
15         links = new LinkedList( Arrays.asList( colors ) );
16
17         links.addLast( "red" );    // add as last item
18         links.add( "pink" );      // add to the end
19         links.add( 3, "green" );  // add at 3rd index
20         links.addFirst( "cyan" ); // add as first item
21
22         // get LinkedList elements as an array
23         colors = links.toArray( new String[ links.size() ] );
24
25         System.out.println( "colors: " );
26
27         for ( int count = 0; count < colors.length; count++ )
28             System.out.println( colors[ count ] );
29     }
30 }
31

```

Use List method toArray  
to obtain array representation of  
LinkedList

```
32  // execute application
33  public static void main( String args[] )
34  {
35      new UsingToArray();
36  }
37
38 } // end class UsingToArray
```

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

# Collections Class

- Collections Framework provides set of algorithms
  - Implemented as **static** methods
    - **List** algorithms
      - **sort**
      - **binarySearch**
      - **reverse**
      - **shuffle**
      - **fill**
      - **copy**
    - **Collection** algorithms
      - **min**
      - **max**

# Algorithm sort

- **sort**
  - Sorts **List** elements
    - Order is determined by natural order of elements' type
    - Relatively fast

```

1 // Fig. 21.6: Sort1.java
2 // Using algorithm sort
3
4 // Java core packages
5 import java.util.*;
6
7 public class Sort1 {
8     private static String suits[] =
9         { "Hearts", "Diamonds", "Clubs", "Spades" };
10
11     // display array elements
12     public void printElements()
13     {
14         // create ArrayList
15         ArrayList list = new ArrayList( Arrays.asList( suits ) );
16
17         // output list
18         System.out.println( "Unsorted array elements:\n" + list );
19
20         // sort ArrayList
21         Collections.sort( list );
22
23         // output list
24         System.out.println( "Sorted array elements:\n" + list );
25     }
26
27     // execute application
28     public static void main( String args[] )
29     {
30         new Sort1().printElements();
31     }
32
33 } // end class Sort1

```

Create ArrayList

Use Collections method  
sort to sort ArrayList

```
Unsorted array elements:  
[Hearts, Diamonds, Clubs, Spades]  
Sorted array elements:  
[Clubs, Diamonds, Hearts, Spades]
```

```

1 // Fig. 21.7: Sort2.java
2 // Using a Comparator object with algorithm sort
3
4 // Java core packages
5 import java.util.*;
6
7 public class Sort2 {
8     private static String suits[] =
9         { "Hearts", "Diamonds", "Clubs", "Spades" };
10
11     // output List elements
12     public void printElements()
13     {
14         // create List
15         List list = Arrays.asList( suits );
16
17         // output List elements
18         System.out.println( "Unsorted array elements:\n" + list );
19
20         // sort in descending order using a comparator
21         Collections.sort( list, Collections.reverseOrder() );
22
23         // output List elements
24         System.out.println( "Sorted list elements:\n" + list );
25     }
26
27     // execute application
28     public static void main( String args[] )
29     {
30         new Sort2().printElements();
31     }
32
33 } // end class Sort2

```

Method `reverseOrder` of class `Collections` returns a `Comparator` object that represents the collection's reverse

order

Method `sort` of class `Collections` can use a `Comparator` object to sort a `List`



Unsorted array elements:

[Hearts, Diamonds, Clubs, Spades]

Sorted list elements:

[Spades, Hearts, Diamonds, Clubs]

# Algorithm shuffle

- **shuffle**
  - Randomly orders **List** elements

```
1 // Fig. 21.7: Sort2.java
2 // Using a Comparator object with algorithm sort
3
4 // Java core packages
5 import java.util.*;
6
7 public class shuffleList {
8     private static String suits[] =
9         { "Hearts", "Diamonds", "Clubs", "Spades" };
10
11     // output List elements
12     public void printElements()
13     {
14         // create List
15         List list = Arrays.asList( suits );
16
17         // output List elements
18         System.out.println( "Unsorted array elements:\n" + list );
19
20         // shuffle list elements in random
21         Collections.shuffle( list );
22
23         // output List elements
24         System.out.println( "Shuffled list elements:\n" + list );
25     }
26
27     // execute application
28     public static void main( String args[] )
29     {
30         new shuffleTest().printElements();
31     }
32
33 } // end class Sort2
```

# **reverse, fill, copy, max and min**

- **reverse**
  - Reverses the order of **List** elements
- **fill**
  - Populates **List** elements with values
- **copy**
  - Creates copy of a **List**
- **max**
  - Returns largest element in **List**
- **min**
  - Returns smallest element in **List**

```

1 // Fig. 21.9: Algorithms1.java
2 // Using algorithms reverse, fill, copy, min and max
3
4 // Java core packages
5 import java.util.*;
6
7 public class Algorithms1 {
8     private String letters[] = { "P", "C", "M" }, lettersCopy[];
9     private List list, copyList;
10
11     // create a List and manipulate it with algorithms from
12     // class Collections
13     public Algorithms1()
14     {
15         list = Arrays.asList( letters );      // get List
16         lettersCopy = new String[ 3 ];
17         copyList = Arrays.asList( lettersCopy );
18
19         System.out.println( "Printing initial statistics: " );
20         printStatistics( list );
21
22         Collections.reverse( list );          // reverse order
23         System.out.println( "\nPrinting statistics after
24             "calling reverse: " );
25         printStatistics( list );
26
27         Collections.copy( copyList, list );    // copy List
28         System.out.println( "\nPrinting statistics after " +
29             "copying: " );
30         printStatistics( copyList );
31
32         System.out.println( "\nPrinting statistics after " +
33             "calling fill: " );
34         Collections.fill( list, "R" );
35         printStatistics( list );

```

Use method **reverse** of class **Collections** to obtain **List** in reverse order

Use method **copy** of class **Collections** to obtain copy of **List**

Use method **fill** of class **Collections** to populate **List** with the letter "R"

```

36  }
37
38  // output List information
39  private void printStatistics( List listRef )
40  {
41      System.out.print( "The list is: " );
42
43      for ( int k = 0; k < listRef.size(); k++ )
44          System.out.print( listRef.get( k ) + " " );
45
46      System.out.print( "\nMax: " + Collections.max( listRef ) );
47      System.out.println(
48          "   Min: " + Collections.min( listRef ) );
49  }
50
51  // execute application
52  public static void main( String args[] )
53  {
54      new Algorithms1();
55  }
56
57  } // end class Algorithms1

```

Obtain maximum value in

List

Obtain minimum value in

List

Printing initial statistics:

The list is: P C M

Max: P Min: C

Printing statistics after calling reverse:

The list is: M C P

Max: P Min: C

Printing statistics after copying:

The list is: M C P

Max: P Min: C

Printing statistics after calling fill:

The list is: R R R

Max: R Min: R

# Algorithm `binarySearch`

- `binarySearch`
  - Locates **Object** in **List**
    - Returns index of **Object** in **List** if **Object** exists
    - Returns negative value if **Object** does not exist

```

1 // Fig. 21.10: BinarySearchTest.java
2 // Using algorithm binarySearch
3
4 // Java core packages
5 import java.util.*;
6
7 public class BinarySearchTest {
8     private String colors[] = { "red", "white", "blue", "black",
9         "yellow", "purple", "tan", "pink" };
10    private ArrayList list;           // ArrayList reference
11
12    // create, sort and output list
13    public BinarySearchTest()
14    {
15        list = new ArrayList( Arrays.asList( colors ) );
16        Collections.sort( list );    // sort the ArrayList
17        System.out.println( "Sorted ArrayList: " + list );
18    }
19
20    // search list for various values
21    public void printSearchResults()
22    {
23        printSearchResultsHelper( colors[ 0 ] ); // first item
24        printSearchResultsHelper( colors[ 3 ] ); // middle item
25        printSearchResultsHelper( colors[ 7 ] ); // last item
26        printSearchResultsHelper( "aardvark" ); // below lowest
27        printSearchResultsHelper( "goat" );    // does not exist
28        printSearchResultsHelper( "zebra" );    // does not exist
29    }
30
31    // helper method to perform searches
32    private void printSearchResultsHelper( String key )
33    {
34        int result = 0;
35

```

Sort List in ascending  
order



```
36     System.out.println( "\nSearching for: " + key );
37     result = Collections.binarySearch( list, key );
38     System.out.println(
39         ( result >= 0 ? "Found at index " + result :
40         "Not Found (" + result + ")" ) );
41 }
42
43 // execute application
44 public static void main( String args[] )
45 {
46     new BinarySearchTest().printSearchResults();
47 }
48
49 } // end class BinarySearchTest
```

Use method `binarySearch` of class `Collections` to search `List` for specified key

Sorted ArrayList: 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: aardvark  
Not Found (-1)

Searching for: goat  
Not Found (-3)

Searching for: zebra  
Not Found (-9)

# Sets

- **Set**
  - **Collection** that contains unique elements
  - **HashSet**
    - Stores elements in hash table
  - **TreeSet**
    - Stores elements in tree

```

1 // Fig. 21.11: SetTest.java
2 // Using a HashSet to remove duplicates
3
4 // Java core packages
5 import java.util.*;
6
7 public class SetTest {
8     private String colors[] = { "red", "white", "blue",
9         "green", "gray", "orange", "tan", "white", "cyan",
10         "peach", "gray", "orange" };
11
12     // create and output ArrayList
13     public SetTest()
14     {
15         ArrayList list;
16
17         list = new ArrayList( Arrays.asList( colors ) );
18         System.out.println( "ArrayList: " + list );
19         printNonDuplications( list );
20     }
21
22     // create set from array to eliminate duplicates
23     public void printNonDuplications( Collection collection )
24     {
25         // create a HashSet and obtain its iterator
26         HashSet set = new HashSet( collection );
27         Iterator iterator = set.iterator();
28
29         System.out.println( "\nNonDuplications are: " );
30
31         while ( iterator.hasNext() )
32             System.out.print( iterator.next() + " " );
33
34         System.out.println();
35     }

```

Create HashSet  
from Collection

object

Use Iterator to  
traverse HashSet  
and print

nonduplicates

```
36
37 // execute application
38 public static void main( String args[] )
39 {
40     new SetTest();
41 }
42
43 } // end class SetTest
```

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

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

```

1 // Fig. 21.12: SortedSetTest.java
2 // Using TreeSet and SortedSet
3
4 // Java core packages
5 import java.util.*;
6
7 public class SortedSetTest {
8     private static String names[] = { "yellow", "green", "black",
9         "tan", "grey", "white", "orange", "red", "green" };
10
11     // create a sorted set with TreeSet, then manipulate it
12     public SortedSetTest()
13     {
14         TreeSet tree = new TreeSet( Arrays.asList( names ) );
15
16         System.out.println( "set: " );
17         printSet( tree );
18
19         // get headSet based upon "orange"
20         System.out.print( "\nheadSet (\"orange\"): " );
21         printSet( tree.headSet( "orange" ) );
22
23         // get tailSet based upon "orange"
24         System.out.print( "tailSet (\"orange\"): " );
25         printSet( tree.tailSet( "orange" ) );
26
27         // get first and last elements
28         System.out.println( "first: " + tree.first() );
29         System.out.println( "last : " + tree.last() );
30     }
31
32     // output set
33     public void printSet( SortedSet set )
34     {
35         Iterator iterator = set.iterator();

```

Create TreeSet  
from names array

Use TreeSet method  
headSet to get TreeSet  
subset less than

"orange"

Use TreeSet method  
tailSet to get TreeSet  
subset greater than or equal  
"orange"

Methods first and last  
obtain smallest and largest  
TreeSet elements,


respectively

```

36
37     while ( iterator.hasNext() )
38         System.out.print( iterator.next() + " " );
39
40     System.out.println();
41 }
42
43 // execute application
44 public static void main( String args[] )
45 {
46     new SortedSetTest();
47 }
48
49 } // end class SortedSetTest

```

Use Iterator to  
traverse HashSet  
and print values



```

set:
black green grey orange red tan white yellow

headSet ("orange"): black green grey
tailSet ("orange"): orange red tan white yellow
first: black
last : yellow

```

# Maps

- **Map**
  - **HashMap**
  - Associates keys to values
  - Cannot contain duplicate keys
    - Called *one-to-one mapping*

```

1 // Fig. 21.13: MapTest.java
2 // Using a HashMap to store the number of words that
3 // begin with a given letter
4
5 // Java core packages
6 import java.util.*;
7
8 public class MapTest {
9     private static String names[] = { "one", "two", "three",
10         "four", "five", "six", "seven", "two", "ten", "four" };
11
12     // build a HashMap and output contents
13     public MapTest()
14     {
15         HashMap map = new HashMap();
16         Integer i;
17
18         for ( int count = 0; count < names.length; count++ ) {
19             i = ( Integer ) map.get(
20                 new Character( names[ count ].charAt( 0 ) ) );
21
22             // if key is not in map then give it value one
23             // otherwise increment its value by 1
24             if ( i == null )
25                 map.put(
26                     new Character( names[ count ].charAt( 0 ) ),
27                     new Integer( 1 ) );
28             else
29                 map.put(
30                     new Character( names[ count ].charAt( 0 ) ),
31                     new Integer( i.intValue() + 1 ) );
32         }
33
34         System.out.println(
35             "\nnumber of words beginning with each letter:  " );

```

Create

HashMap

Use method get to retrieve  
a Character from

HashMap

Use method put to store  
a Character with an  
Integer key in

HashMap



```

36     printMap( map );
37 }
38
39 // output map contents
40 public void printMap( Map mapRef )
41 {
42     System.out.println( mapRef.toString() );
43     System.out.println( "size: " + mapRef.size() );
44     System.out.println( "isEmpty: " + mapRef.isEmpty() );
45 }
46
47 // execute application
48 public static void main( String args[] )
49 {
50     new MapTest();
51 }
52
53 } // end class MapTest

```

```

    number of words beginning with each letter:
        {t=4, s=2, o=1, f=3}
        size: 4
        isEmpty: false

```

# **Java Native Interface and Methods**

- JNI is a mechanism that allows
  - a Java Program to call a function in a C or C++ program.
  - a C or C++ program to call a method in a Java program.

# Purpose and Features

- JNI stands for **Java Native Interface**
- JNI specifies a communication protocol between Java code - and external, native code.
- It enables your Java code to interface with native code written in other languages (such as C, C++)
- Native code typically accesses the CPU and registers directly and is thus faster than interpreted code (like Java)
- Java native methods are methods declared in your Java code (much like you declare an abstract method), but which are actually implemented in another programming language.

# How to write Native Methods

- Write Java code
- Compile Java code
- Create C header (.h file)
- Write C code
- Create shared code Library (or DLL)
- Run application

# Hello World java program

```
public class HelloWorld{  
    public native void Hello();  
  
    static{  
        System.loadLibrary("HelloWorld");//load HelloWorld.dll  
    }  
    public static void main(String args[])  
    {  
        HelloWorld hw=new HelloWorld();  
        hw.Hello();  
    }  
}
```

# Hello.c

```
#include "HelloWorld.h"
#include<jni.h>
#include<stdio.h>

JNIEXPORT void JNICALL Java_HelloWorld_Hello(JNIEnv
*env, jobject obj)
{
    printf("Hello World\n");
    return;
}
```

**Thank you!**