Chapter 4

Java Arrays and Strings

Arrays

Array Declaration and Initialization

Parallel Arrays

Array Operations

- Assignment operation
- Array cloning

Passing Arrays to Methods

Array Manipulations:

- Searching Arrays
- Sorting Arrays

Multidimensional Arrays

Strings

String class

String Methods

StringBuffer class

Vector Class

Stack Class

Array Class

ArrayList Class

Additional Array Examples

Arrays

The class Array is declared as

```
public final class Arrays extends Object
```

All Java arrays have a fixed capacity or size. A defined array cannot grow or shrink.

Array Declaration and Initialization

```
final int SIZE = 4;
char [ ] charArray = new char[SIZE];
double [ ] price;
int n[ ] = new int [20];            // int n [ ]; n = new int [20];
int Temp24hr [ ] = new int [24];    //Define a 24 hours temp array
int [ ] PID = {1234, 5678, 0101, 5789};
long [ ] ssNo = {999990000, 9999666666, 999777777, 999888888};
double [ ] salary = {70E3, 56E3, 80E3, 100E3};
int [ ] mileage = new int [ 20];
```

Parallel Arrays

```
final String [] Employees =
      { "Paul", "Ron", "Dick", "Roger"};
String [ ] phone;
String [ ] address;
int [ ] empID;
final NUM_EMPLOYEE = Employees.length;
```

Multidimensional Arrays

```
// 3 x 3 array
int n2dArray[ ][ ];
b = new int[3][3];
// Distance Map Array for 50 cities
int dis50by50[][];
```

Array Operations

- **Assignment operation**

```
      // Define a array
int [ ] intArray = {10, 20, 30, 40, 50, 60};
      // Declare a new array
int [ ] intArrayCopy;
      // This operation establishes a reference. The array data items
      // are not duplicated.
intArrayCopy = intArray;
```

- **Array cloning**

For primitive data types, the array cloning makes a distinct copy or duplicating the array elements.

```
      // Define a array
int [ ] intArray = {10, 20, 30, 40, 50, 60};
      // Declare a new array
int [ ] intArrayCopy;
      // Use the clone() method to make a completely distinct copy of
      // array.
intArrayCopy = intArray.clone(); // {10, 20, 30, 40, 50, 60}
```

Array Manipulations:

- Searching an Array for a specific element
- Determine the array length
- Sorting elements of arrays
- Summing elements of an integer array
- and more

Passing Arrays to Methods

Assume that there are a number of methods in a class:

```
public static double sumOfArray(double [ ] thisArray)
  {
  double sum = 0.0;
  int len = thisArray.length;
  for(int n = 0; I < len; n++)
    {
     sum += thisArray[n];
    }
  return sum;
}

public static double computeAverage(double [ ] thisArray)
  {
   if (thisArray.length > 0)
   return (sum(thisArray) / thisArray.length);
 }

public static double [ ] initArray(int size )
  {
   double [ ] thisArray = new double [size];
  // Setup array elements
  return thisArray;
  }
```

Typical Array Application Examples

- Initializing Arrays
- Summing Array
- Poll analysis
- Histogram plotting applications
- Dice-rolling program
- Passing Arrays
- Sorting Arrays
- Linear Search
- Binary Search
- Multidimensional Arrays
- Double Arrays (Parallel arrays)

Java String and StringBuffer Classes

A string is a sequence of characters. Java has the following two string classes for text manipulation:

- String - for constant text string (immutable)
- StringBuffer - for text strings that can be modified or changed (mutable)

Examples:

- Name
- CustomerName
- StreetAddress
- City
- State
- PhoneNumber
- EmailAddress
- DomainName

Java 2 Platform Std. Ed. v1.3

Class String

- The class **String** is defined in java.lang.String
  import java.lang.String
  import java.lang.*
- The String class represents character strings that cannot be changed after they are created (immutable).
- String constructors
  public String()
  // Create a String object with an empty character sequence.

  public String(char[] value)
  // Create a new String with the sequence of characters contained in the
  // character array argument.

  public String(char[] value, int offset, int length)
  // Create a new String with the sequence of sub-array characters
  // contained in the character array. The offset argument is the index of
  // the first character of the subarray and the length argument specifies
  // the length of the subarray.

  public String(byte[] bytes)
  // Create a new String and initialize it by converting to the specified array
  // of bytes to the default character encoding.

  public String(byte[] bytes, int offset, int length)
  // Create a new String and initialize it by converting the specified sub-array
  // of bytes to the default character encoding.

public String(byte[] bytes, String enc)
// Create a new String and initialize it by converting the specified array of
// bytes to the specified character encoding.

public String(byte[] bytes, int offset, int length, String enc)
// Create a new String and initialize it by converting the specified sub-array
// of bytes to the specified character encoding.

public String(StringBuffer buffer)
// Create a new String and initialize it using the sequence of characters in
// the buffer argument.

- **Other String Methods and Fields**

char charAt(int n_index)
// return the Unicode at nth position
int compareTo(Object o)
// Compare this String to another Object
int compareTo(String str)
// Compare two Strings
int compareToIgnorecase(String str)
// Compare two strings
String concat(String)
// String concatenation
static String copyValueOf(char [ ] data)
// Returns a String with the chars from data array
static String copyValueOf(char [ ] data, int offset, int count)
// Returns a String with the specified chars from data array
boolean endWith(String suffix)
// Test if this string ends with the specified suffix
boolean equals(Object anotherObi)
// Compare this string to another object

```
boolean equalsIgnoreCase(String anotherString)
        // Compare this string to another String
byte[ ] getBytes()
        // Convert this String into bytes according to default char encoding
void getChars(int srcBegin, int srcEnd, char[ ] dest, int destBegin)
        // Copy characters from this string into destination
        // character array
int hashCode()
        // Returns a hashcode for this string
int indexOf(int ch)
        // Returns the index within this string of the first occurrence of the
        // specified character
int indexOf(int ch, int fromIndex))
        // Returns the index within this string of the first occurrence of the
        // specified character at the specified index)
int indexOf(String str)
        // Returns the index within this string of the first occurrence of the
        // specified substring
int indexOf(String str, int fromIndex)
        // Returns the index within this string of the first occurrence of the
        // specified substring, starting at the specified index
String intern()
        // Returns a canonical representation for the string object
int lastindexOf(int ch)
        // Returns the index within this string of the last occurrence of the
        // specified character.
int lastindexOf(int ch, int fromIndex)
        // Returns the index within this string of the last occurrence of the
        // specified character, searching backward starting at the specified
        // index.
```

```
int lastindexOf(String str)
        // Returns the index within this string of the rightmost occurrence of
        // the specified substring.
int lastindexOf(String str, int fromIndex)
        // Returns the index within this string of the last occurrence of the
        // specified substring.
int length()
          // Return the length of string
String replace(char oldChar, char newChar)
        // Replace all oldChars with newChar
boolean startsWith(String prefix)
        // Test if this string starts with the specified prefix
boolean startsWith(String prefix, int toffset)
        // Test if this string starts with the specified prefix
        // beginning at specified index
String substring(int beginIndex)
        // Return a new string that is a substring of this string
String substring(int beginIndex, int endIndex)
        // Return a new string that is a substring of this string
char [] toCharArray()
        // Converts this string to a new character array
String toLowercase()
        // converts all the characters in string to lower case
String toUppercase()
        // converts all the characters in string to upper   case
String trim()
        // Remove white space from both ends of this string
static String valueOf(boolean b)
        // Returns the string representation of the boolean argument.
static String valueOf(char c)
        // Returns the string representation of the char argument.
```

```
static String valueOf(char [ ] data, int offset, int count)
        // Returns the string representation of a subarray
static String valueOf(double d)
        // Returns the string representation of the double argument.
static String valueOf(float f)
        // Returns the string representation of the float argument.
static String valueOf(int i)
        // Returns the string representation of the i argument.
static String valueOf(long l)
        // Returns the string representation of the long argument.
static String valueOf(Object obj)
        // Returns the string representation of the object argument.
```

The new operator may be used to create a String object

Declaring Strings

Example:

```
String companyNme = new String ("Triple Crown");
```

or

```
String companyName = "Triple Crown";
String [ ] departments = {"Sales", "HumanResources", "Accounting"};
```

Substrings

For example:

```
String str = "abc";
```

is equivalent to:

```
char data[] = {'a', 'b', 'c'};
String str = new String(data);
```

Here are some more examples of how strings can be used:

```
System.out.println("abc");
String cde = "cde";
System.out.println("abc" + cde);
String c = "abc".substring(2,3);
String d = cde.substring(1, 2);
```

Java 2 Platform Std. Ed. v1.3

java.lang.StringBuffer

Class StringBuffer

public final class **StringBuffer** extends Object implements Serializable

Java supports the String class for creating constant string and StringBuffer class for strings that can be modified.

Constructors

StringBuffer()

    // Construct a string buffer with 16 char length as default

StringBuffer(int n)

    // Construct a string buffer with length of n characters

StringBuffer(String str)

    // Construct a buffer string with string str

Methods

Object.toString()

StringBuffer append(boolean b)

    // Appends the string representation of the boolean

    // argument to the string buffer

StringBuffer append(char c)

    // Appends the string representation of the char

    // argument to the string buffer

StringBuffer append(char[] str)

    // Appends the char array argument to the string buffer

StringBuffer append(char[], int offset, int len)

    // Appends the char subarray argument to the string buffer

StringBuffer append(double d)

    // Appends the string representation of the double

    // argument to the string buffer

```
StringBuffer append(float f)
        // Appends the string representation of the float
        // argument to the string buffer
StringBuffer append(int i)
        // Appends the string representation of the int
        // argument to the string buffer
StringBuffer append(long l)
        // Appends the string representation of the long
        // argument to the string buffer
StringBuffer append(Object obj)
        // Appends the string representation of the Object
        // argument to the string buffer
StringBuffer append(String str)
        // Appends the string to this string buffer
int capacity()
        // Returns the current capacity of the String buffer
char charAt(int index)
        // Return a char that is pointed by the index that refers to the String buffer
StringBuffer delete(int start, int end)
        // Remove the characters stored in the String buffer and refer
        // by the star and end marker
deleteCharAt(int index)
        // Remove the char that is pointed by the index that refers to the String
        // buffer
void ensureCapacity(int minCapacity)
        // Ensure that a min capacity for the String buffer is reserved
void getChars(int srcBegin, int srcEnd, char [ ] dst, int dstBegin)
        // Copy characters from this String buffer to the destination
        // array
StringBuffer insert(int offset, boolean b)
        // Insert the string representation of the boolean argument into this string
```

```
        // buffer
StringBuffer insert(int offset, char c)
        // Insert the string representation of the char argument into this string
        // buffer
StringBuffer insert(int offset, char [ ] str)
        // Insert the string representation of the char array argument into this
        // string buffer
StringBuffer insert(int offset, char [ ] str, int offset, int len)
        // Insert the string representation of the subarray argument into this
        // string buffer
StringBuffer insert(int offset, double d)
        // Insert the string representation of the double argument into this string
        // buffer
StringBuffer insert(int offset, float f)
        // Insert the string representation of the float argument into this string
        // buffer
StringBuffer insert(int offset, int i)
        // Insert the string representation of the int argument into this string
        // buffer
StringBuffer insert(int offset, long l)
        // Insert the string representation of the long argument into this string
        // buffer
StringBuffer insert(int offset, Object obj)
        // Insert the string representation of the Object argument into this string
        // buffer
StringBuffer insert(int offset, String str)
        // Insert the string into this string buffer
int length()
        // Return the character count or length of this string buffer
StringBuffer replace(int start, int end, String str)
        // Replace characters in the String buffer with the specified substring
```

```
StringBuffer reverse()
        // The character sequence contained in this string buffer is replaced by the
        // reverse of the sequence.
void setCharAt(int index, char ch)
        // Set the char at the specified index position to ch
void setLength(int newLength)
        // Set the length of this String buffer
String substring(int start)
        // Return a new String contains in the String buffer beginning from start
        // position
String substring(int start, int end)
        // Return a new String contains in the String buffer specified by the
        // beginning and ending index
String toString()
        // Converts to a string representing the data in this string buffer
```

Typical Examples and Considerations of Using Strings and Characters

- String Constructors
- String Manipulation Methods
- String Comparisons
- startsWith and endsWith Methods
- Hashcode
- Locating Characters and Substrings
- Substring Methods
- Concatenating Strings
- Miscellaneous String Methods
- The String class valueOf() Methods
- The String class intern() Methods
- StringBuffer Constructors
- StringBuffer length() and capacity() Methods
- StringBuffer Class character manipulation methods
- StringBuffer class append() method
- StringBuffer class insert() method
- Static Character Testing Methods
- Non-static Character Testing Methods
- Tokenizing Strings

Java 2 Platform Std. Ed. v1.3

Vector Class

The Vector class provides an array-based container whose size can be modified.

The Vector class is declared as:

public class **Vector** extends AbstractList implements List, Cloneable, Serializable

import java.util.Vector;

Field Variables

```
Protected int capacityIncrement;
        // The amount by which the capacity of the vector is automatically
        // incremented when its size becomes greater than its capacity
protected int elementCount;
        // The number of valid components in this Vector object
protected Object[ ] elementData;
        // The array buffer into which the components of the vector are stored
```

Constructors

```
Vector()
        // Construct an empty vector, size 10, standard capacity increment is
        // zero
Vector(Collection c)
        // Construct a vector  containing the elements of collections
Vector(int initialCapacity)
        // Construct an empty vector with specified initial capacity
Vector(int initialCapacity, int capacityIncrement)
        // Construct an empty vector with specified initial capacity
        // and capacity increment
```

Methods

```
void add(int index, Object element)
        // Inserts the specified element at the specified position in this Vector
boolean add(Object o)
        // Inserts the specified element at the specified position in this Vector
boolean addAll(Collection c)
        // Appends all of the elements in the specified Collection
        // to the end of this Vector
boolean addAll(int index, Collection c)
        // Inserts all of the elements in the specified Collection
        // into this Vector at the specified position
void addElement(Object obj)
        // Add the specified obj component to the end of this vector
int capacity()
        // Return the current capacity of this vector
void clear()
        // Remove all of the elements from this vector
Object clone()
        // Return a clone of this vector
boolean contains(Object elem)
        // Tests if the specified object is a components in this
        // vector
boolean containsAll(Collection c)
        // Tests if this Vector contains all of the elements in this
        // specified Collection
void copyInto(Object [ ] anArray)
        // Copies the components of this vector into the
        // specified array
Object elementAt(int index)
        //  Returns an enumeration of the components of this vector
```

```
Enumeration elements()
        // Returns an enumeration of the components of this vector
void ensureCpacity(int minCpacity)
        // Increases the capacity of this vector, to ensure that it
        // can hold at least the minimum capacity
boolean equals(Object o)
        // Compare the specified Object with this Vector for equality
Object firstElement()
        // Returns the first component of this vector
Object get(int index)
        // Returns the element at the specified position in this Vector
int hashCode()
        // Return the hash code for this vector
int indexOf(Object obj)
        // Search for the first occurrence of the given argument
int indexOf(Object elem, int index)
        // Search for the first occurrence of the given argument
void insertElementAt(Object obj, int index)
        // Inserts the specified object as a components in this vector at the
        // specified index
boolean isEmpty()
        // Checks if this vector has no components
Object lastElement()
        // Returns the last component of the vector
Object remove(int index)
        // Removes the element at the specified position in this Vector
boolean remove(Object o)
        // Removes the first occurrence of the specified element
        // in this Vector
```

```
boolean removeAll(Collection c)
        // Removes all specified element that are contained
        // in the specified Collection
void removeAllElements()
        // Removes all elements from this vector and sets
        // its size to zero
boolean removeElement(Object obj)
        // Removes the first occurrence of the argument from
        // this Vector
void removeElementAt(int index)
        // Deletes the component at the specified index
void setElementAt(Object obj, int index)
        // Sets the component at the specified index
void setSize(int newSize)
        // Set the size of this vector
int size()
        // Return the number of components in this vector
List subList(int fromIndex, int toIndex)
        // Returns a view of portion of this List
Obect[ ] toArray()
        // Returns an array containing all of the elements in this
        // vector in the correct order
Object[ ] toArray(Object [ ] a)
        // Returns an array containing all of the elements in the
        // Vector in the correct order
String toString()
        // Returns a string representation of this vector
void trimToSize()
        // Trim the capacity of this to be the vector's current size
```

Stack

import java.util.Stack;

public class **Stack** extends vector

The **Stack** class represents a last-in-first-out (LIFO) stack of objects. It is derived from (extends) class **Vector** with five operations that allow a vector to be treated as a stack:

object push(Object item)

    // Add an object to the top of the stack

Object pop( )

    // Remove the object at the top of the stack

boolean empty()

    // Test to know if the stack is empty.

Object peek( )

    // Look at the object at the top of the stack

int search(Object item)

    // Search and return the 1-based position where an object is on the stack

java.util

Class Arrays

public class Arrays extends Object

This class contains various methods for manipulating arrays such as sorting and searching. It also contains a static factory that allows arrays to be viewed as lists.

Methods

static List asList(Object[] a)

static int binarySearch(byte[ ] a, byte key)

staic int binarySearch(char[ ] a, char key)

static int binarySearch(double[ ] a, double key)

static int binarySearch(float[ ] a, float key)

staic int binarySearch(int[ ] a, int key)

static int binarySearch(long[ ] a, long key)

static int binarySearch(Object[ ] a, Object key)

staic int binarySearch(Object[ ] a, Object key, Comparator c)

static int binarySearch(short[ ] a, short key)

static boolean equals(boolean[] a, boolean [] a2)

static boolean equals(byte[] a, byte[] a2)

static boolean equals(char[] a, char[] a2)

static boolean equals(double[] a, double[] a2)

static boolean equals(float[] a, float[] a2)

static boolean equals(int[] a, int[] a2)

static boolean equals(long[] a, long[] a2)

static boolean equals(Object[] a, Object[] a2)

static boolean equals(short[] a, short[] a2)

static void fill(boolean[] a, boolean val)

static void fill(boolean[] a, int fromIndex, int toIndex, boolean val)

static void fill(byte[] a, byte val)

static void fill(byte[] a, int fromIndex, int toIndex, byte val)

static void fill(char[] a, char val)

static void fill(char[] a, int fromIndex, int toIndex, char val)

static void fill(double[] a, double val)

static void fill(double[] a, int fromIndex, int toIndex, double val)

static void fill(float[] a, float val)

static void fill(float[] a, int fromIndex, int toIndex, float val)

static void fill(int[] a, int val)

static void fill(int[] a, int fromIndex, int toIndex, int val)

static void fill(long[] a, long val)

static void fill(long[] a, int fromIndex, int toIndex, long val)

static void fill(Object [] a, Object val)

static void fill(Object[] a, int fromIndex, int toIndex, Object val)

static void fill(short[] a, short val)

static void fill(short[] a, int fromIndex, int toIndex, short val)

static void sort(byte[] a)

static void sort(byte[] a, int fromIndex, int toIndex)

static void sort(char[] a)

static void sort(char[] a, int fromIndex, int toIndex)

static void sort(double[] a)

static void sort(double[] a, int fromIndex, int toIndex)

static void sort(float[] a)

static void sort(float[] a, int fromIndex, int toIndex)

static void sort(int[] a)

static void sort(int[] a, int fromIndex, int toIndex)

static void sort(long[] a)

static void sort(long[] a, int fromIndex, int toIndex)

static void sort(Object[] a)

static void sort(Object[] a, Comparator)

static void sort(Object[] a, int fromIndex, int toIndex)

static void sort(Object[] a, int fromIndex, int toIndex, Comparator c)

static void sort(short[] a)

static void sort(short[] a, int fromIndex, int toIndex)

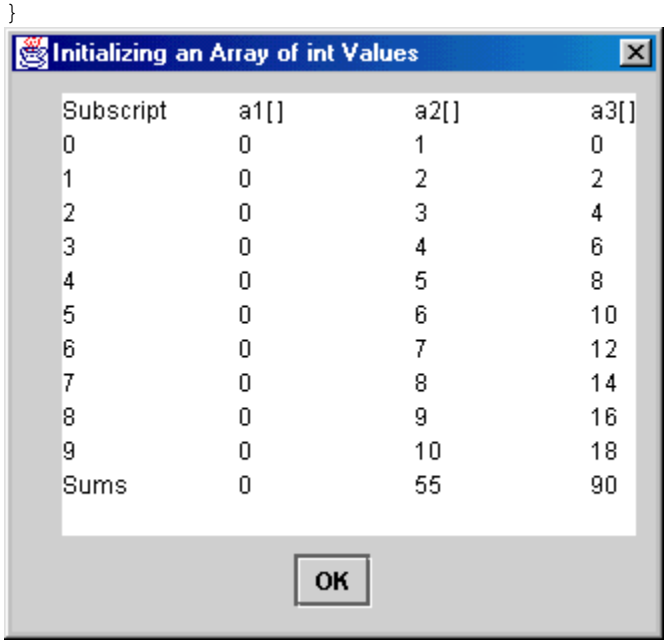
Array Examples

```
// ArrayExs1.java
// This example program combines Fig. 7.3, Fig. 7.4, Fig. 7.5, and Fig.
// 7.6 as shwon in Chapter 7 Arrays of the Book Entitled: "Java How to
// Program by Deitel & Deitel, 3 rd rdition, published by Prentice Hall,
// 1999.
//
// This program performs array declaration, initialization, suming of
// the following three arrays and display results through Window's
// control called TextArea:
// - Unitialized Array
// - Initializing an array
// - Summing an array
import javax.swing.*;
public class ArrayExs1 {
   public static void main( String args[] )
   {
      ////////////////////////////////////////////////
      // Declare an output string
      String outString = "";
      ////////////////////////////////////////////////
      // Declare an uninitialized array
      int a1[];
      a1 = new int[ 10 ];
        ////////////////////////////////////////////////
        // Initialize an array a2[]
      int a2[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
      ////////////////////////////////////////////////
      // Declare another array and assign values
      final int SIZE = 10;
      int a3[];
      a3 = new int[ SIZE ];
      int i;
      for ( i = 0; i < SIZE; i++ )
        {
         a3[ i ] = 2 * i;
        }
       ////////////////////////////////////////////////
       // Preapre output n1[], n2[], and n3[] array contents
      outString += "Subscript\ta1[ ]\ta2[ ]\ta3[ ]\n";
      for( i = 0; i < SIZE; i++ )
       {
         outString += i + "\t" + a1[ i ]
                           + "\t" + a2[ i ]
                           + "\t" + a3[ i ] + "\n";
       }
      ////////////////////////////////////////////////
      //  Summing three arrays
      int sumA1 = 0, sumA2 = 0, sumA3 = 0;
      for (i = 0; i < SIZE; i++)
        {
            sumA1 += a1[i];
            sumA2 += a2[i];
            sumA3 += a3[i];
        }

      outString += "Sums" + "\t" + sumA1
                                   + "\t" + sumA2
                                   + "\t" + sumA3 + "\n";
```

```
      // Declare one TextArea with 12 rows, 30 columns
      JTextArea outTextArea = new JTextArea( 12, 10 );
      outTextArea.setText( outString );

      JOptionPane.showMessageDialog( null, outTextArea,
         "Initializing an Array of int Values",
         JOptionPane.PLAIN_MESSAGE );
      System.exit( 0 );
   }
}
```



Sorting Arrays

```
//////////////////////////////////////////////////////////////////////////
//
// ArraySortExs2.java
//
// The program Fig.7-11 as shown in Chapter 7 Arrays of the Book
// Entitled: "Java How to Program by Deitel & Deitel, 3rd. edition,
// published by Prentice Hall, 1999 is referenced.
//
// The data in the array is randomly generated, and bubble sorted
// sorts an array's values into ascending order.
//
//////////////////////////////////////////////////////////////////////////
import java.util.Arrays;
import java.awt.*;
import javax.swing.*;

public class ArraySortExs2 {
   public static void main(String args[])
   {
      ////////////////////////////////////////////////////
      //
      String outString_1 = "Original Data Array a[]\n";
      JTextArea outTextArea_1 = new JTextArea();

        ////////////////////////////////////////////////////
        //
      int a[] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };



      for ( int i = 0; i < a.length; i++ )
        {
         outString_1 += "   " + a[ i ];
        }


      ////////////////////////////////////////////////////
        // Using Arrays.sort ---- Tuned QuickSort algorithm
        Arrays.sort(a);

        outString_1 += "\nSorted Data Array (Quick Sort) a[]\n";
        for ( int i = 0; i < a.length; i++ )
        {
         outString_1 += "   " + a[ i ];
        }

        JOptionPane.showMessageDialog(null, outString_1, "Quick Sort",
      JOptionPane.PLAIN_MESSAGE);

      ////////////////////////////////////////////////////
      //
      String outString_2 = "Original Data Array b[]\n";
      JTextArea outTextArea_2 = new JTextArea();


      ////////////////////////////////////////////////////
```

```
      //
      int b[] = { 10, 6, 2, 56, 23, 12, 29, 12, 24, 37 };

      for ( int i = 0; i < b.length; i++ )
        {
         outString_2 += "   " + b[ i ];
        }


      ////////////////////////////////////////////////////
      // Using Arrays.sort ---- Tuned QuickSort algorithm
        bubbleSort(b);

        outString_2 += "\nSorted Data Array (Bubble Sort) b[]\n";
        for ( int i = 0; i < b.length; i++ )
        {
         outString_2 += "   " + b[ i ];
        }


      JOptionPane.showMessageDialog(null, outString_2, "Bubble Sort",
      JOptionPane.PLAIN_MESSAGE);

      System.exit(0);

   }




   // Bubble sort
   public static void bubbleSort( int b[] )
   {
      for ( int pass = 1; pass < b.length; pass++ )// Outer passes
        {
         for ( int i = 0; i < b.length - 1; i++ )  // Inner pass
          {
            if ( b[ i ] > b[ i + 1 ] )             // one comparison
               swap( b, i, i + 1 );                // one swap
          }
        }
   }

   // Swapping elements
   public static void swap( int c[], int first, int second )
   {
      int temp;

      temp = c[ first ];
      c[ first ] = c[ second ];
      c[ second ] = temp;
   }
}
```


Quick Sort
Original Data Array a[]
2 6 4 8 10 12 89 68 45 37
Sorted Data Array (Quick Sort) a[]
2 4 6 8 10 12 37 45 68 89
OK
Bubble Sort
Original Data Array b[]
10 6 2 56 23 12 29 12 24 37
Sorted Data Array (Bubble Sort) b[]
2 6 10 12 12 23 24 29 37 56
OK

java.util

Class ArrayList

This class implements the List interface using an array. By using an Object array, an ArrayList can store any reference type, since all reference types are classes that are directly or indirectly extend Object.

public class **ArrayList** extends AbstractList implements List, Cloneable, Serializable

The class ArrayList is a resizable-array implementation of the List interface. This class is equivalent to the class Vector, except that it is unsynchronized.

Constructors

ArrayList()
    // Constructs an empty array

ArrayList(Collection c)
    // A collection represents a group of objects known as its elements

ArrayList(int initialCapacity)
    // Constructs an empty list with the specified initial capacity

Methods

void add(int index, Object element)
    // Inserts the specified element at the specified position in this list

boolean add(Object o)
    // Appends the specified element to the end of this list

boolean addAll(Collection c)
    // Appends all of the elements in the specified Collection to the end of this
    // list

boolean addAll(int index, Collection c)
    // Inserts all of the elements in the specified Collection into
    // this list, starting at the specified position

void clear()
    // Clears all of the elements from this list

Object clone()
    // Returns a shallow copy of this ArrayList instance

```
boolean contains(Object element)
        // Returns true if the specified elements is contained in this list

void ensureCapacity(int minCapacity)
        // Increases the capacity, ensure that it can holds at least the number
        // of elements specified

Object get(int index)
        // Returns the element at the specified position in this list

int indexOf(Object elem)
        // Searches for the first occurrence of the given arguments

boolean isEmpty()
        // Tests if this list has no elements

int lastIndexOf(Object elem)
        // Returns the index of the last occurrence of the specified object in this list

Object remove(int index)
        // Removes the element at the specified position in this list

protected void removeRange(int fromIndex, int toIndex)
        // Removes from this List all of the elements whose index is between
        // fromIndex, inclusive and toIndex, exclusive

Object set(int index, Object element)
        // Replaces at the elements at the specified position in this list with
        // the specified element

int size()
        // Returns the number of elements in this list

Object[ ] toArray()
        // Returns an array containing all of the elements in this list in the correct
        // order

Object[ ] toArray(Object[ ] a)
        // Returns an array containing all of the elements in this list in the correct
        // order.

void trimToSize()
        // Trims the capacity of this ArrayList instance to be the list's current size
```

java.util
Class LinkedList

public class LinkedList exetends AbstractSequentiaList implements List, Cloneable, Serializable

All of the stack, queue, deque operations could be easily recast in terms of the standard list operation.

Constructors

LinkedList()
    // Constructs an empty list

LinkedList(Collection c)
    // Constructs a list containing the elements of the specified collection, in
    // order they are returned by the collection's iterator

Methods

void add(int index, Object element)
    // Inserts the specified element at the specified position in this list

boolean add(Object o)
    // Appends the specified element to the end of this list

boolean addAll(Collection c)
    // Appends all of the elements in the specified Collection to the end of this
    // list

boolean addAll(int index, Collection c)
    // Inserts all of the elements in the specified Collection into
    // this list, starting at the specified position

void addFirst(Object o)
    // Inserts the given elements at the beginning of this list

void addLast(Object o)
    // Appends the given elements at the end of this list

void clear()
    // Removes all of the elements from this list

Object clone()
    // Returns a shallow copy of this ArrayList instance

```
boolean contains(Object o)
        // Returns true if the specified elements is contained in this list

Object get(int index)
        // Returns the element at the specified position in this list

Object getFirst()
        // Returns the first element in this lists

Object getLast()
        // Returns the last element in this lists

boolean isEmpty()
        // Tests if this list has no elements

int indexOf(Object o)
        // Returns the index in this list of the first occurrence of the specified
        // object in this list

int lastIndexOf(Object o)
        // Returns the index of the last occurrence of the specified object in this list

ListIterator listIterator(int index)
        // Returns a list-iterator of the elements in this list

Object remove(int index)
        // Removes the element at the specified position in this list

boolean remove(Object o)
        // Removes the first occurrence of the specified element in this list

Object removeFirst()
        // Removes and returns the first elements from this list

Object removeLast()
        // Removes and returns the last elements from this list

Object set(int index, Object element)
        // Replaces at the elements at the specified position in this list with
        // the specified element

int size()
        // Returns the number of elements in this list
```

Object[ ] toArray()
    // Returns an array containing all of the elements in this list in the correct
    // order

Object[ ] toArray(Object[ ] a)
    // Returns an array containing all of the elements in this list in the correct
    // order.

Reflection
java.lang.reflect.Array

public final class Array extends Objects

    // This array class provides methods for creating and accessing

    // dynamically array.

    // It permits widening conversions to occur during a get or set

    // operation, but throws an exception: IllegalArgumentException if

    // narrowing conversion would occur.

Methods

    static Object get(Object oArray, int index)

    static boolean getBoolean(Object oArray, int index)

    static byte getByte(Object oArray, int index)

    static char getChar(Object oArray, int index)

    static double getDouble(Object oArray, int index)

    static float getFloat(Object oArray, int index)

    static int getInt(Object oArray, int index)

    static int getLength(Object oArray)

    static long getLong(Object oArray, int index)

    static short getShort(Object oArray, int index)

    static Object newInstance(Class componentType, int length)

    static Object newInstance(Class componentType, int [ ] dimension)

    static void set(Object oArray, int index, Object value)

    static void setBoolean(Object oArray, int index, boolean z)

static void setByte(Object oArray, int index, byte b)

static void setChar(Object array, int index, char c)

static void setDouble(Object oArray, int index, double d)

static void setFloat(Object oArray, int index, float f)

static void setInt(Object oArray, int index, int i)

static void setLong(Object oArray, int index, long l)

static void setShort(Object oArray, int index, short s)