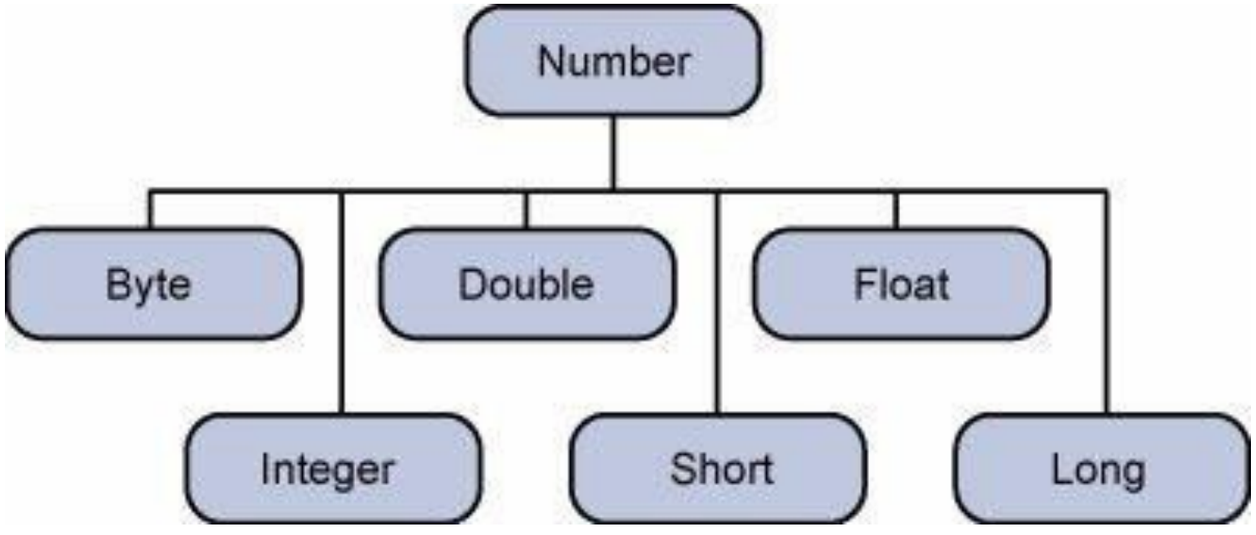
Numbers




```
public class Test{

public static void main(String args[]){
Integer x =5;// boxes int to an Integer object
x =x +10;// unboxes the Integer to a int
System.out.println(x);
}
}
```

Number Methods:

SN	Methods with Description
1	xxxValue() Converts the value of *this* Number object to the xxx data type and returned it.
2	compareTo() Compares *this* Number object to the argument.
3	equals() Determines whether *this* number object is equal to the argument.
4	valueOf() Returns an Integer object holding the value of the specified primitive.
5	toString() Returns a String object representing the value of specified int or Integer.
6	parseInt() This method is used to get the primitive data type of a certain String.
7	abs() Returns the absolute value of the argument.
8	ceil() Returns the smallest integer that is greater than or equal to the argument. Returned as a double.

SN	Methods with Description
9	floor() Returns the largest integer that is less than or equal to the argument. Returned as a double.
10	rint() Returns the integer that is closest in value to the argument. Returned as a double.
11	round() Returns the closest long or int, as indicated by the method's return type, to the argument.
12	min() Returns the smaller of the two arguments.
13	max() Returns the larger of the two arguments.
14	exp() Returns the base of the natural logarithms, e, to the power of the argument.
15	log() Returns the natural logarithm of the argument.
16	pow() Returns the value of the first argument raised to the power of the second argument.

17	sqrt() Returns the square root of the argument.
18	sin() Returns the sine of the specified double value.
19	cos() Returns the cosine of the specified double value.
20	tan() Returns the tangent of the specified double value.
21	asin() Returns the arcsine of the specified double value.
22	acos() Returns the arccosine of the specified double value.
23	atan() Returns the arctangent of the specified double value.
24	atan2() Converts rectangular coordinates (x, y) to polar coordinate (r, theta) and returns theta.
25	toDegrees() Converts the argument to degrees
26	toRadians() Converts the argument to radians.
27	random() Returns a random number.

Characters

```
char ch ='a';

// Unicode for uppercase Greek omega character char
uniChar ='\u039A';

// an array of chars
char[] charArray ={`a’,'b','c','d','e'};
```

Escape Sequence	Description
\t	Inserts a tab in the text at this point.
\b	Inserts a backspace in the text at this point.
\n	Inserts a newline in the text at this point.
\r	Inserts a carriage return in the text at this point.
\f	Inserts a form feed in the text at this point.
\'	Inserts a single quote character in the text at this point.
\"	Inserts a double quote character in the text at this point.
\\	Inserts a backslash character in the text at this point.

Character Methods:

SN	Methods with Description
1	isLetter() Determines whether the specified char value is a letter.
2	isDigit() Determines whether the specified char value is a digit.
3	isWhitespace() Determines whether the specified char value is white space.
4	isUpperCase() Determines whether the specified char value is uppercase.
5	isLowerCase() Determines whether the specified char value is lowercase.
6	toUpperCase() Returns the uppercase form of the specified char value.
7	toLowerCase() Returns the lowercase form of the specified char value.
8	toString() Returns a String object representing the specified character valuethat is, a one-character string.

```
public class Test{

public static void main(String args[])
{ System.out.println(Character.isLetter('c'));
System.out.println(Character.isLetter('5'));
}
}
```

```
public class Test{

public static void main(String args[])
{ System.out.println(Character.isDigit('c'));
System.out.println(Character.isDigit('5'));
}
}
```

```
public class Test{

public static void main(String args[])
{ System.out.println(Character.isLowerCase('c'));
System.out.println(Character.isLowerCase('C'));
System.out.println(Character.isLowerCase('\n'));
System.out.println(Character.isLowerCase('\t'));
}
}
```

Strings:

Creating strings:

```
public class StringDemo{

public static void main(String args[]){
String string1 ="saw I was ";
System.out.println("Dot "+ string1 +”Tod");
```

Concatenating Strings:

```
"My name is “.concat("Zara");
```

Creating Format Strings:

```
String fs;
fs =String.format("The value of the float variable
is "+ "%f, while the value of the integer “+

"variable is %d, and the string “+
"is %s", floatVar, intVar, stringVar);
System.out.println(fs);

"%f, while the value of the integer "+
"variable is %d, and the string "+
"is %s", floatVar, intVar, stringVar);
```

String Methods:

SN	Methods with Description
1	char charAt(int index) Returns the character at the specified index.
2	int compareTo(Object o) Compares this String to another Object.
3	int compareTo(String anotherString) Compares two strings lexicographically.
4	int compareToIgnoreCase(String str) Compares two strings lexicographically, ignoring case differences.
5	String concat(String str) Concatenates the specified string to the end of this string.
6	boolean contentEquals(StringBuffer sb) Returns true if and only if this String represents the same sequence of characters as the specified StringBuffer.
7	static String copyValueOf(char[] data) Returns a String that represents the character sequence in the array specified.

8 static String copyValueOf(char[] data, int offset, int count)
Returns a String that represents the character sequence in the array specified.

9 boolean endsWith(String suffix)
Tests if this string ends with the specified suffix.

10 boolean equals(Object anObject) Compares this string to the specified object.

11 boolean equalsIgnoreCase(String anotherString)
Compares this String to another String, ignoring case considerations.

12 byte getBytes()
Encodes this String into a sequence of bytes using the platform's default charset, storing the result into a new byte array.

13 byte[] getBytes(String charsetName
Encodes this String into a sequence of bytes using the named charset, storing the result into a new byte array.

String class:

The String class represents character strings. All string literals in Java programs, such as "abc" , are implemented as instances of this class. Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings.

A Java String contains an immutable sequence of Unicode characters. Unlike C/ C++, where string is simply an array of char , A Java String is an object of the class java.lang . ... You can assign a string literal directly into a String variable, instead of calling the constructor to create a String instance.

String class methods

- **public char charAt(int index) ...**
- **public String concat(String s) ...**
- **public boolean equalsIgnoreCase(String s) ...**
- **public int length() ...**
- **public String toUpperCase()**

String Buffer

StringBuffer in java is used to create modifiable String objects. This means that we can use StringBuffer to append, reverse, replace, concatenate and manipulate Strings or sequence of characters. StringBuffer and StringBuilder are called mutable because whenever we perform a modification on their objects their state gets changed. And they were created as mutable because of the immutable nature of String class.

String Builder

The StringBuilder Class. StringBuilder objects are like String objects, except that they can be modified. Internally, these objects are treated like variable-length arrays that contain a sequence of characters.

String is immutable whereas StringBuffer and StringBuider are mutable classes. StringBuffer is thread safe and synchronized whereas StringBuilder is not, thats why StringBuilder is more faster than StringBuffer.

StringBuffer	vs	StringBuilder
StringBuffer is synchronized i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously.		StringBuilder is non-synchronized i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously.
StringBuffer is less efficient than StringBuilder.		StringBuilder is more efficient than StringBuffer.

```
public class StringCompareEmp{
   public static void main(String args[]){
      String str = "Hello World";
      String anotherString = "hello world";
      Object objStr = str;

      System.out.println( str.compareTo(anotherString) );
      System.out.println( str.compareToIgnoreCase(anotherString) );
      System.out.println( str.compareTo(objStr.toString()));
   }
}

public class StringCompareequl{
   public static void main(String []args){
      String s1 = "google";
      String s2 = "google";
      String s3 = new String (“google world");
      System.out.println(s1.equals(s2));
      System.out.println(s2.equals(s3));
   }
}
public class StringCompareequl{
   public static void main(String []args){
      String s1 = "google";
      String s2 = "google";
      String s3 = new String (“google world”);
      System.out.println(s1 == s2);
      System.out.println(s2 == s3);
   }
}
```

How to search the last position of a substring

This example shows how to determine the last position of a substring inside a string with the help of strOrig.lastIndexOf(Stringname) method.

```
public class SearchlastString {
   public static void main(String[] args) {
      String strOrig = "Hello world ,Hello Reader";
      int lastIndex = strOrig.lastIndexOf("Hello");

      if(lastIndex == - 1){
         System.out.println("Hello not found");
      } else {
         System.out.println("Last occurrence of Hello is at index "+
lastIndex);
      }
   }
}
```

```
Output:
Last occurrence of Hello is at index 13
```

```
public class HelloWorld{
   public static void main(String []args) {
      String t1 = "HelloWorld";
      int index = t1.lastIndexOf("1");
      System.out.println(index);
   }
}
```

```
public class StringReverseExample{
    public static void main(String[] args) {
        String string = "abcdef";
        String reverse = new StringBuffer(string).reverse().toString();
        System.out.println("\nString before reverse: "+string);
        System.out.println("String after reverse: "+reverse);
    }
}
```

Output:

```
String before reverse:abcdef
String after reverse:fedcba
```

```
import java.io.*;
import java.util.*;

public class HelloWorld {
    public static void main(String[] args) {
        String input = "abcdef";
        char[] try1 = input.toCharArray();
        for (int i = try1.length-1;i >= 0; i--)
          System.out.print(try1[i]);
    }
}
```

```java
//program to search a string

public class SearchStringEmp{
   public static void main(String[] args) {
      String strOrig = "Hello readers";
      int intIndex = strOrig.indexOf("Hello");

      if(intIndex == - 1) {
         System.out.println("Hello not found");
      } else {
         System.out.println("Found Hello at index " +
intIndex);
      }
   }
}

public class HelloWorld {
   public static void main(String[] args) {
      String text = "The cat is on the table";
      System.out.print(text.contains("the"));
   }
}
```

1) StringBuffer append() method

The append() method concatenates the given argument with this string.

2) StringBuffer insert() method

The insert() method inserts the given string with this string at the given position.

3) StringBuffer replace() method

The replace() method replaces the given string from the specified beginIndex and endIndex.

4) StringBuffer delete() method

The delete() method of StringBuffer class deletes the string from the specified beginIndex to endIndex.

5) StringBuffer reverse() method

The reverse() method of StringBuilder class reverses the current string.

6) StringBuffer capacity() method

The capacity() method of StringBuffer class returns the current capacity of the buffer. The default capacity of the buffer is 16. If the number of character increases from its current capacity, it increases the capacity by (oldcapacity*2)+2. For example if your current capacity is 16, it will be (16*2)+2=34.

7) StringBuffer ensureCapacity() method

The ensureCapacity() method of StringBuffer class ensures that the given capacity is the minimum to the current capacity. If it is greater than the current capacity, it increases the capacity by (oldcapacity*2)+2. For example if your current capacity is 16, it will be (16*2)+2=34.

Important methods of StringBuilder class

1) StringBuilder append() method

The StringBuilder append() method concatenates the given argument with this string.

2) StringBuilder insert() method

The StringBuilder insert() method inserts the given string with this string at the given position.

3) StringBuilder replace() method

The StringBuilder replace() method replaces the given string from the specified beginIndex and endIndex.

4) StringBuilder delete() method

The delete() method of StringBuilder class deletes the string from the specified beginIndex to endIndex.

5) StringBuilder reverse() method

The reverse() method of StringBuilder class reverses the current string.

6) StringBuilder capacity() method

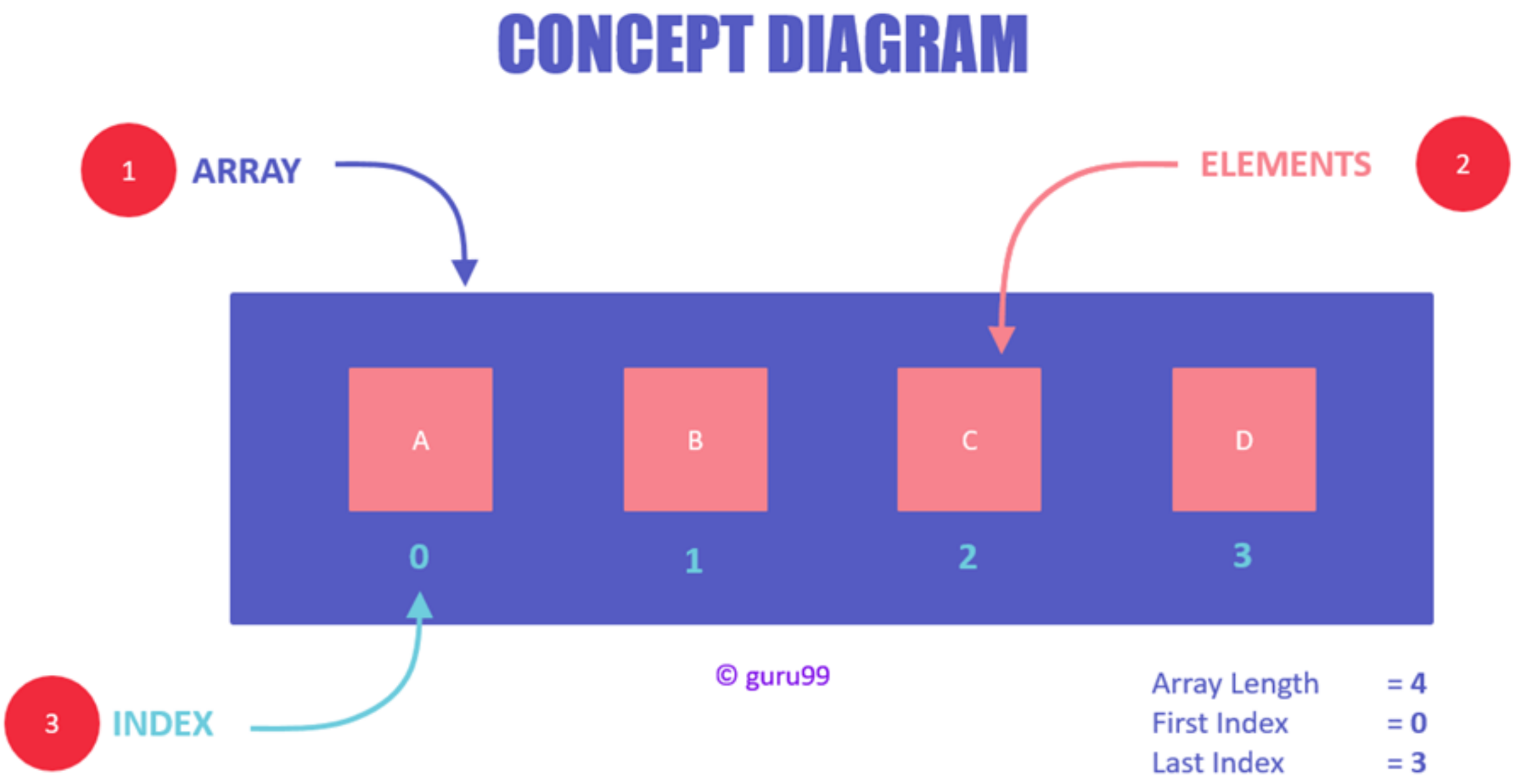
The capacity() method of StringBuilder class returns the current capacity of the Builder. The default capacity of the Builder is 16. If the number of character increases from its current capacity, it increases the capacity by (oldcapacity*2)+2. For example if your current capacity is 16, it will be (16*2)+2=34.

7) StringBuilder ensureCapacity() method

The ensureCapacity() method of StringBuilder class ensures that the given capacity is the minimum to the current capacity. If it is greater than the current capacity, it increases the capacity by (oldcapacity*2)+2. For example if your current capacity is 16, it will be (16*2)+2=34

Arrays:

An Array is a collection of variables of the same type.



Creating an array:

```
public class TestArray{

public static void main(String[] args){
double[] myList ={1.9,2.9,3.4,3.5};
// Print all the array elements
for(int i =0; i < myList.length; i++)
{
System.out.println(myList[i]);
} } }
```

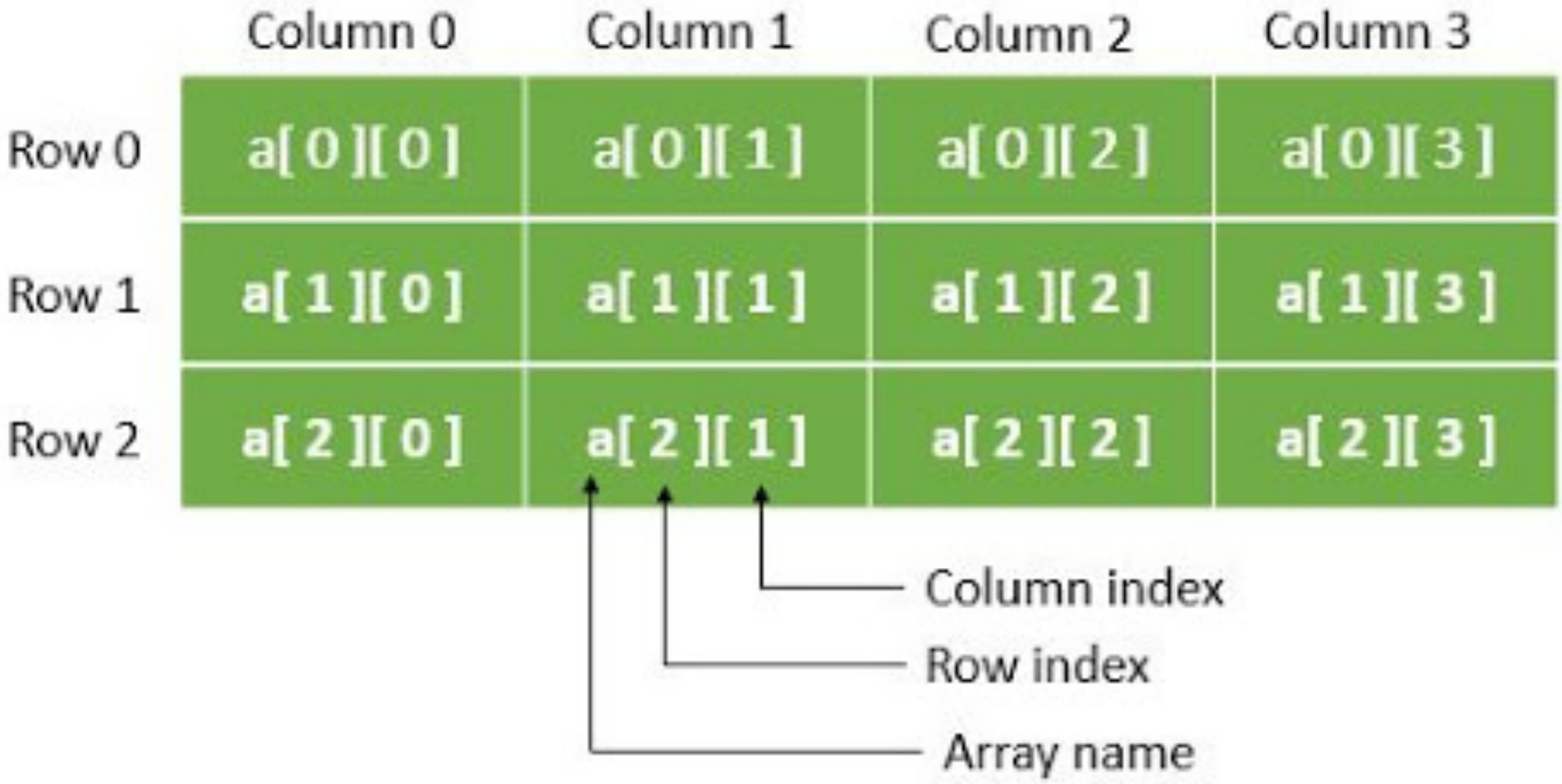
The foreach Loops

JDK 1.5 introduced a new for loop known as foreach loop or enhanced for loop, which enables you to traverse the complete array sequentially without using an index variable.

```
public class TestArray {

   public static void main(String[] args) {
      double[] myList = {1.9, 2.9, 3.4, 3.5};

      // Print all the array elements
      for (double element: myList) {
         System.out.println(element);
      } } }
```


Column 0
Column 1
Column 2
Column 3
Row 0
Row 1
Row 2
a[0][0]
a[0][1]
a[0][2]
a[0][3]
a[1][0]
a[1][1]
a[1][2]
a[1][3]
a[2][0]
a[2][1]
a[2][2]
a[2][3]
Column index
Row index
Array name

Passing Arrays to Methods

```
public static void printArray(int[] array) {
   for (int i = 0; i < array.length; i++) {
      System.out.print(array[i] + " ");
   }
}
```

Returning an Array from a Method

```
public static int[] reverse(int[] list) {
   int[] result = new int[list.length];

   for (int i = 0, j = result.length - 1; i < list.length; i++, j--)
{
      result[j] = list[i];
   }
   return result;
}
```

An array in Java is a type of object that can contain a number of variables. ... You can have arrays of any of the Java primitives or reference variables. The important point to remember is that when created, primitive arrays will have default values assigned, but object references will all be null.

Declare an array

```
String[] aArray = new String[5];
String[] bArray = {"a","b","c", "d", "e"};
String[] cArray = new String[]{“a","b","c","d","e"};
```

1. Print an array in Java

```
int[] intArray = { 1, 2, 3, 4, 5 };
String intArrayString = Arrays.toString(intArray);

// print directly will print reference value
System.out.println(intArray);
// [I@7150bd4d

System.out.println(intArrayString);
// [1, 2, 3, 4, 5]
```

Create an ArrayList from an array

```
String[] stringArray = { "a", "b", "c", "d", "e" };
ArrayList<String> arrayList = new
ArrayList<String>(Arrays.asList(stringArray));
System.out.println(arrayList);
// [a, b, c, d, e]
```

Check if an array contains a certain value

```
String[] stringArray = { "a", "b", "c", "d", "e" };
boolean b =
Arrays.asList(stringArray).contains("a");
System.out.println(b);
// true
```

Concatenate two arrays

```
int[] intArray = { 1, 2, 3, 4, 5 };
int[] intArray2 = { 6, 7, 8, 9, 10 };
// Apache Commons Lang library
int[] combinedIntArray = ArrayUtils.addAll(intArray,
intArray2);
```

Declare an array inline

```
method(new String[]{"a", "b", "c", "d", “e"});
```

Joins the elements of the provided array into a single String

```
// containing the provided list of elements
// Apache common lang
String j = StringUtils.join(new String[] { "a", "b", "c" }, ",
");
System.out.println(j);
// a, b, c
```

Convert an ArrayList to an array

```
String[] stringArray = { "a", "b", "c", "d", "e" };
ArrayList<String> arrayList = new
ArrayList<String>(Arrays.asList(stringArray));
String[] stringArr = new String[arrayList.size()];
arrayList.toArray(stringArr);
for (String s : stringArr)
   System.out.println(s);
```

Convert an array to a set

```
Set<String> set = new HashSet<String>(Arrays.asList(stringArray));
System.out.println(set);
//[d, e, b, c, a]
```

Reverse an array

```
int[] intArray = { 1, 2, 3, 4, 5 };
ArrayUtils.reverse(intArray);
System.out.println(Arrays.toString(intArray));
//[5, 4, 3, 2, 1]
```