Fundamentals of Object Oriented Programming

Dr. Ayesha Hakim

MODULE 3

3	**Arrays String and vectors**		**09**	**CO2**
	3.1	Arrays: Arrays: 1D, 2D, Variable Length array, for-each with Array, Array of objects, Vectors: Vector, ArrayList, Wrapper class. Command line Arguments.		
	3.2	Immutable string ,Methods of String class, String comparison, concatenation, substring, toString method		
	3.3	String-Buffer class, StringBuilder class		

INTRODUCTION TO ARRAYS

- Array is a group of continuous or related data items that share a common name.
- Syntax:

  array_name[value];
- Example:

  salary[10];

ONE-DIMENSIONAL ARRAY

- A list of items can be given one variable name using only one subscript and such a variable is called *a single- subscripted or One-dimensional array.*
- *Express as:*
  x[1],x[2].....x[n]
- *The subscript of an array can be integer constants, integer variables like i, or expressions that yield integers.*

CREATING AN ARRAY

- Like any other variables, array must be declared and created in the computer memory before they are used.
- Creation of an array involves three steps:-

1. Declaring the array
2. Creating memory locations
3. Putting values into the memory locations

DECLARATION OF ARRAYS

- Arrays in Java may be declared in two forms:

Form1

```
type arrayname[ ];
```

Form2

```
Type[ ] arrayname;
```

Example:

```
int       number[ ];
float     average[ ];
int[ ]    counter;
float[ ]  marks;
```

CREATION OF ARRAYS

- Java allows to create arrays using **new** operator only , as shown below:

```
arrayname = new  type[size];
```

- Examples:

```
number   = new int[5];
average  = new float[10];
```

INITIALIZATION OF ARRAYS

- In this step values are put into the array created. This process is known as initialization.

```
arrayname[subscript] = value;
```

- Example:

```
number[0]=35;
number[1]=40;
. . . . . .
number[n]=19;
```

Points to ponder:

- Java creates array starting with a subscript of 0 and ends with a value less than the size specified.
- Trying to access an array beyond its boundaries will generate an error message.
- Array can also be initialize as the other ordinary variables when they are declared.
- Array initializer is a list of values separated by commas and surrounded by curly braces.

ARRAY LENGTH

- All arrays store the allocated size in a variable named **length**.
- To access the length of the array a using a.length.
- Each dimension of the array is indexed from zero to its maximum size minus one.
- Example:

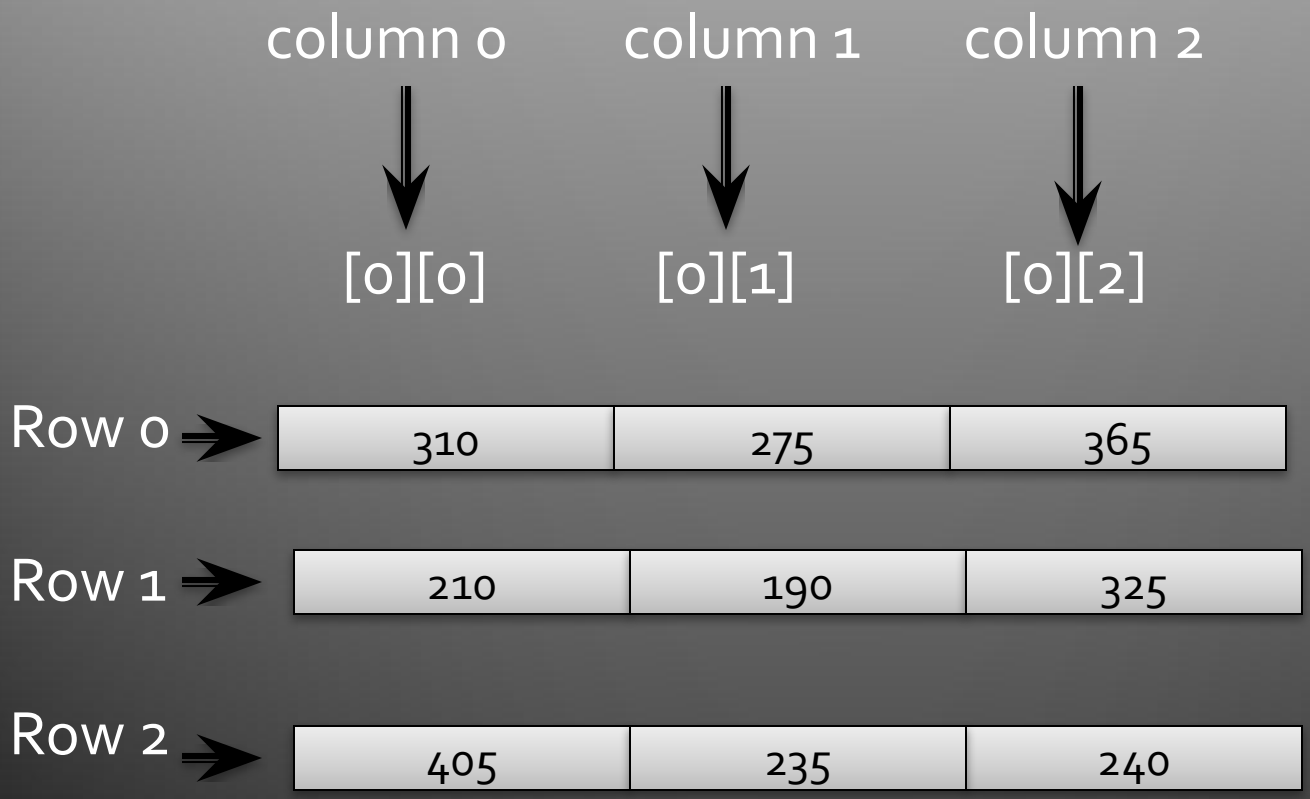
```
int  aSize = a . Length;
```

TWO-DIMENSIONAL ARRAYS

- In this, the first index selects the row and the second index selects the column within that row.
- For creating two-dimensional array, same steps are to be followed as that of simple arrays.
- Example:

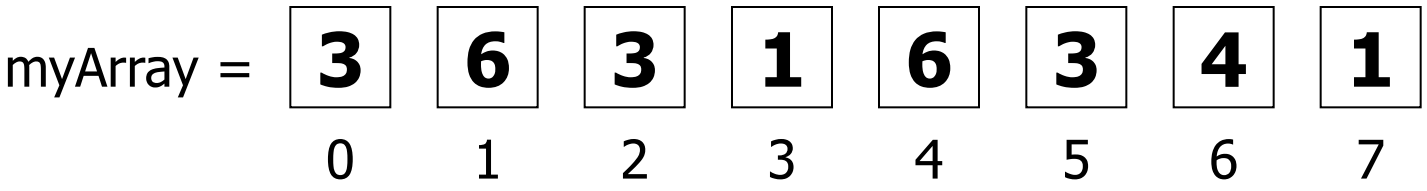
```
int myArray[ ][ ];
myArray = new int [3] [4];
```

REPRESENTATION OF TWO-DIMENSIONAL ARRAY IN MEMORY



Arrays

- An array is a list of similar things
- An array has a fixed:
  - name
  - type
  - length
- These must be declared when the array is created.
- Arrays sizes cannot be changed during the execution of the code



myArray has room for 8 elements

- the elements are accessed by their index
- in Java, array indices start at 0

Declaring Arrays

int myArray[];

declares *myArray* to be an array of integers

myArray = **new** int[8];

sets up 8 integer-sized spaces in memory, labelled *myArray[0]* to *myArray[7]*

int myArray[] = **new** int[8];

combines the two statements in one line

Assigning Values

- refer to the array elements by index to store values in them.

```
myArray[0] = 3;
myArray[1] = 6;
myArray[2] = 3;   ...
```

- can create and initialise in one step:

```
int myArray[] = {3, 6, 3, 1, 6, 3, 4, 1};
```

Iterating Through Arrays

- *for* loops are useful when dealing with arrays:

```
for (int i = 0; i < myArray.length;
  i++) {
   myArray[i] = getsomevalue();
}
```

Arrays of Objects

- So far we have looked at an array of primitive types.
  - integers
  - could also use doubles, floats, characters…
- Often want to have an array of objects
  - Students, Books, Loans ……
- Need to follow 3 steps.

Declaring the Array

1. Declare the array

   private Student studentList[];

   – this declares studentList

2 .Create the array

   studentList = **new** Student[10];

   – this sets up 10 spaces in memory that can hold references to Student objects

3. Create Student objects and add them to the array:

studentList[0] = **new** Student("Cathy", "Computing");

Array of Objects

If you want to store a single object in your program, then you can do so with the help of a variable of type object. But when you are **dealing with numerous objects**, then it is advisable to use an array of objects.

An array of objects is created using the ‘Object’ class.

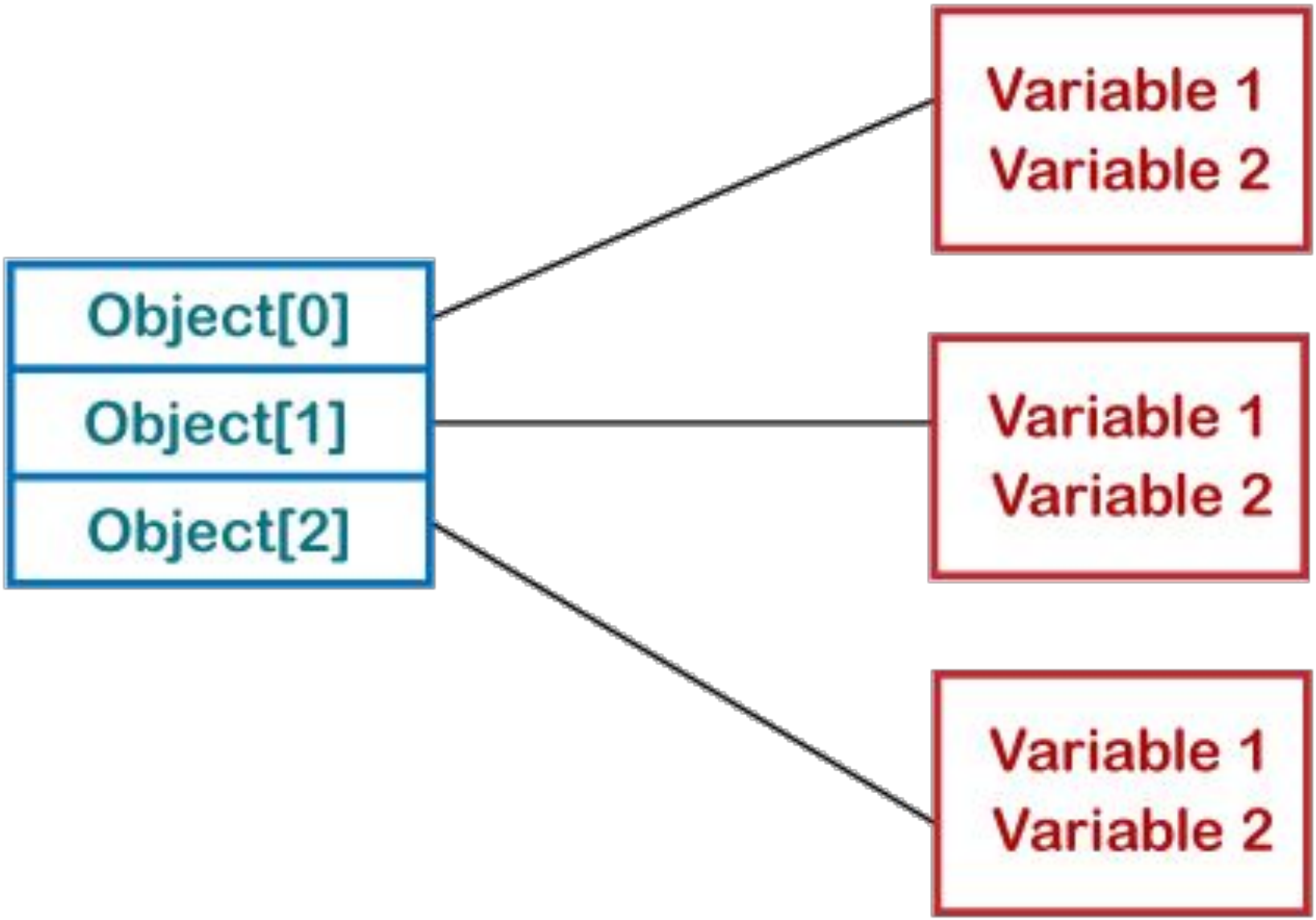
The following statement creates an Array of Objects.

```
Class_name [] objArray;
```

Alternatively, you can also declare an Array of Objects as shown below:

```
Class_name objArray[];
```

Arrays of Objects



Instantiate the array of objects –

Syntax:

```
Class_Name obj[ ]= new Class_Name[Array_Length];
```

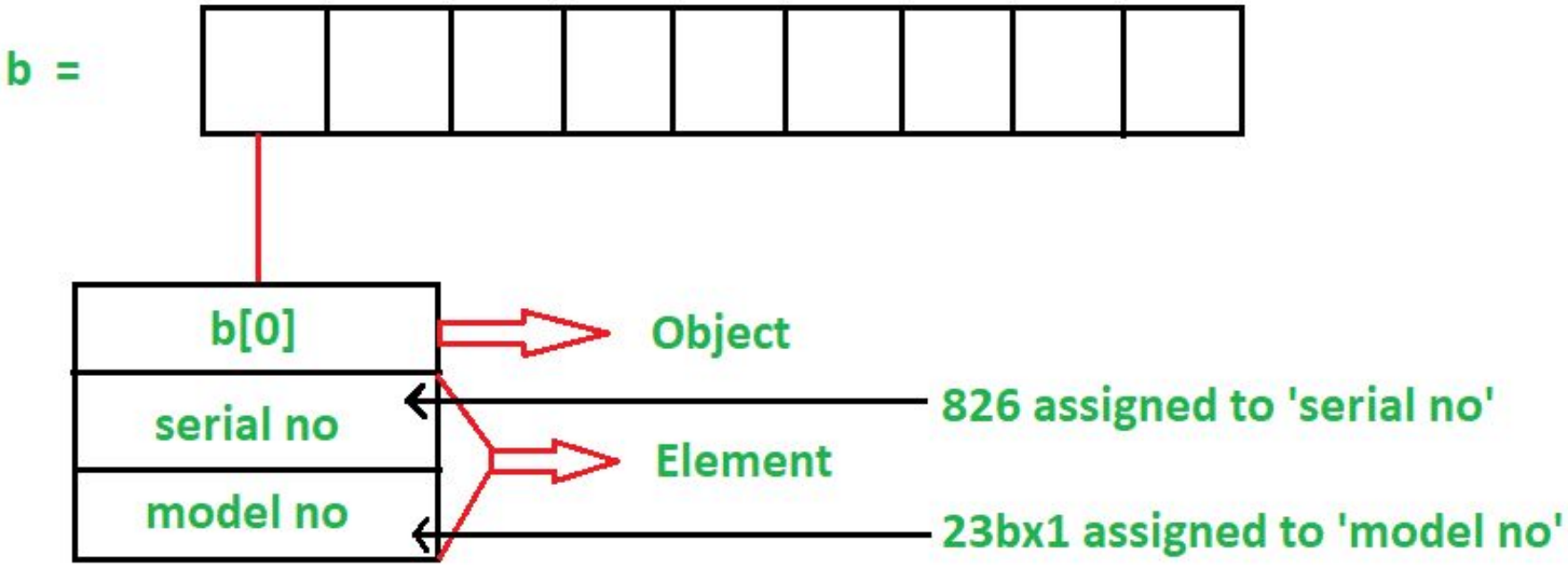
For example, if you have a class Student, and we want to declare and instantiate an array of Student objects with two objects/object references then it will be written as:

```
Student[ ] studentObjects = new Student[2];
```

And once an array of objects is instantiated like this, then the individual elements of the array of objects needs to be created using the new keyword.

bike b[ ] = new bike[9]

b[0]= new bike(826 , 23bx1)



Initialize Array Of Objects

Once the array of objects is instantiated, you have to initialize it with values.

Each element of array i.e. an object needs to be initialized.

1. One way to initialize the array of objects is by **using the constructors**. When you create actual objects, you can assign initial values to each of the objects by passing values to the constructor.
2. You can also have a **separate member method** in a class that will assign data to the objects.

Code:

https://www.geeksforgeeks.org/how-to-create-array-of-objects-in-java/

https://www.javatpoint.com/how-to-create-array-of-objects-in-java (code for ProductList example)

https://www.softwaretestinghelp.com/array-of-objects-in-java/

Activity: Program For An Array Of Objects In Java

WAJP to have an Employee class that has employee Id (empId) and employee name (name) as fields and ‘setData’ & ‘showData’ as methods that assign data to employee objects and display the contents of employee objects respectively.

In the main method of the program, define an array of Employee objects. Note that this will be an array of references and not actual objects. Then using the default constructor, create actual objects for the Employee class. Next, assign data to the objects using the setData method. Lastly, objects invoke the showData method to display the contents of the Employee class objects.

Initialize Array Of Objects - 2 ways

Code 1: The following program shows the initialization of array objects using the constructor.

Here we have used the class Employee. The class has a constructor that takes in two parameters i.e. employee name and employee Id. In the main function, after an array of employees is created, we create individual objects of the class employee. Then we pass initial values to each of the objects using the constructor.

Code 2: This program shows a member function of the Employee class that is used to assign the initial values to the Employee objects.

CODE 1

```
class Main{
   public static void main(String args[]){
     //create array of employee object
    Employee[] obj = new Employee[2] ;

     //create & initialize actual employee objects using constructor
     obj[0] = new Employee(100,"ABC");
     obj[1] = new Employee(200,"XYZ");

     //display the employee object data
     System.out.println("Employee Object 1:");
     obj[0].showData();
     System.out.println("Employee Object 2:");
     obj[1].showData();
  }
}
//Employee class with empId and name as attributes
class Employee{
  int empId;
  String name;
  //Employee class constructor
  Employee(inteid, String n){
     empId = eid;
     name = n;
  }
public void showData(){
   System.out.print("EmpId = "+empId + "   " + " Employee Name = "+name);
   System.out.println();
 }
}
```

OUTPUT

```
Employee Object 1:

EmpId = 100    Employee Name = ABC

Employee Object 2:

EmpId = 200    Employee Name = XYZ
```

CODE 2

```
class Main{
   public static void main(String args[]){
     //create array of employee object
     Employee[] obj = new Employee[2] ;

     //create actual employee object
     obj[0] = new Employee();
     obj[1] = new Employee();

     //assign data to employee objects
     obj[0].setData(100,"ABC");
     obj[1].setData(200,"XYZ");

     //display the employee object data
     System.out.println("Employee Object 1:");
     obj[0].showData();
     System.out.println("Employee Object 2:");
     obj[1].showData();
  }
}
//Employee class with empId and name as attributes
class Employee{
    int empId;
    String name;
    public void setData(intc,String d){
        empId=c;
        name=d;
     }
    public void showData(){
        System.out.print("EmpId = "+empId + "   " + " Employee Name = "+name);
        System.out.println();
     }
}
```

OUTPUT

```
Employee Object 1:

EmpId = 100    Employee Name = ABC

Employee Object 2:

EmpId = 200    Employee Name = XYZ
```

Tutorial Activity : Array of Objects, Jagged Arrays, 'final' keyword

1. WAJP to create a Rectangle class to implement the basic properties of a Rectangle. Declare an array of five Rectangle objects, to store data namely length and width (datatype double). Take values from user and print areas of all 5 rectangles.
2. WAJP to create the Groceries class, declare an array of Groceries objects to pass the product name, price and expiry date (dd/mm/yyyy format) as attributes for 10 products. Display the data as output. To set the data to the Groceries object use the following two approaches:

a. Have a separate member method in the Groceries class

b. Use a constructor

3. Create a Jagged array to get the following output:

```
java -cp /tmp/AP8ZeRiOKV JaggedArrayExample
1 2 3
4 5
6 7 8 9
```

4. Implement Final Variable 'PI' in Java program to calculate & display the area and volume of a cylinder.

Sample Solutions

1. http://www.beginwithjava.com/java/arrays-arraylist/arrays-of-objects.html

   https://www.codesdope.com/course/java-array-of-objects/

1. https://www.scaler.com/topics/array-of-objects-in-java/
2. https://linuxhint.com/jagged-array-in-java-with-example/
3. https://www.shiksha.com/online-courses/articles/final-keyword-in-java/

2-D Arrays (contd.)

- A two-dimensional array can have values that go not only across but also across.
- Sometimes values can be conceptualized in the form of table that is in the form of rows and columns.
- Suppose we want to store the marks of different subjects. We can store it in a one-dimensional array.

2-D Arrays (contd.)

Subject / Roll No.	Physics	Chemistry	Mathematics	English	Biology
01	60	67	47	74	78
02	54	47	67	70	67
03	74	87	76	69	88
04	39	45	56	55	67

2 D Arrays (contd.)

- If you want a multidimensional array, the additional index has to be specified using another set of square brackets.
- Following statements create a two-dimensional array, named as marks, which would have 4 rows and 5 columns

```
int marks[][]                 //declaration of a two-dimensional array
marks = new int[4][5];        //reference to the array allocated, stored in marks variable
```

2 D Arrays (contd.)

- This statement just allocates a 4 × 5 array and assigns the reference to array variable **marks**.
- The first subscript inside the square bracket signifies the number of rows in the table or matrix and the second subscript stands for the number of columns.
- This 4 × 5 table can store 20 values altogether.
- Its values might be stored in contiguous locations in the memory, but logically, the stored values would be treated as if they are stored in a 4 × 5 matrix.
- The following table shows how the *marks* array is conceptually placed in the memory by the above array creation statement

2 D Arrays (contd.)

60 (marks[0][0])	67 (marks[0][1])	47 (marks[0][2])	74 (marks[0][3])	77 (marks[0][4])
54 (marks[1][0])	47 (marks[1][1])	67 (marks[1][2])	70 (marks[1][3])	67 (marks[1][4])
74 (marks[2][0])	87 (marks[2][1])	76 (marks[2][2])	69 (marks[2][3])	88 (marks[2][4])
39 (marks[3][0])	45 (marks[3][1])	56 (marks[3][2])	55 (marks[3][3])	67 (marks[3][4])

2 D Arrays (contd.)

- Like a one-dimensional array, two-dimensional array may be initialized with values, at the time its creation. For example,
  - int marks[2][4] = {2, 3, 6, 0, 9, 3, 3, 2};
- This declaration shows that the first two rows of a 2 × 4 matrix have been initialized by the values shown in the list above. It can also be written as,
  - int marks[ ][ ] = {(2, 3, 6, 0), (9, 3, 3, 2)};

Variable length array: Jagged Arrays

Java treats multidimensional arrays as **“arrays of arrays”.**

A jagged array is an array of arrays such that member arrays can be of different sizes, i.e., we can create a 2-D array but with a **variable number of columns in each row.**

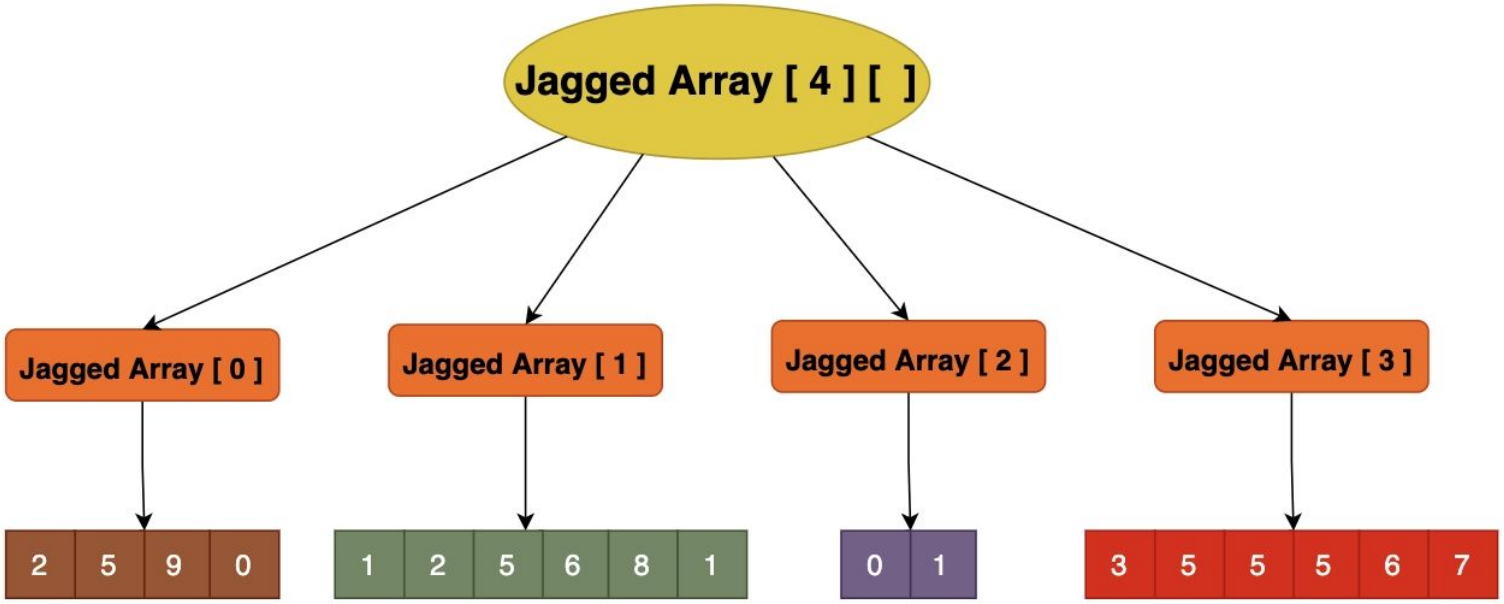
Declaration and Initialization of Jagged array :

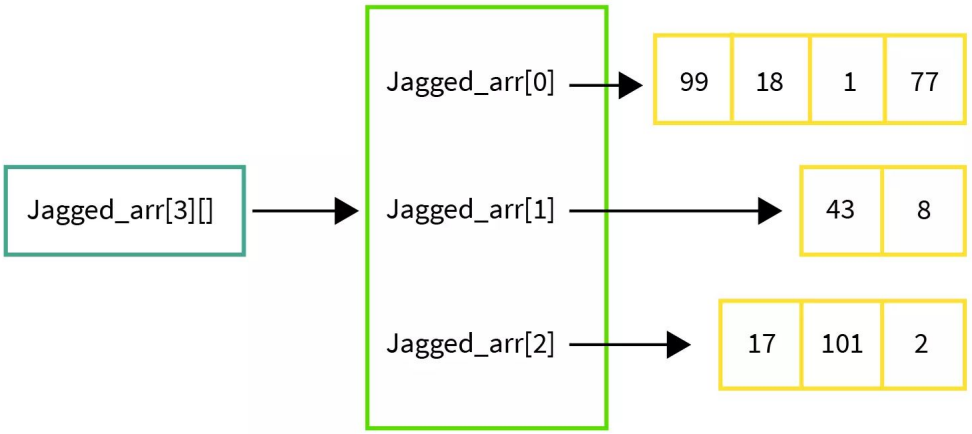
```
Syntax: data_type array_name[][] = new data_type[n][];  //n: no. of rows
            array_name[] = new data_type[n1] //n1= no. of columns in row-1
            array_name[] = new data_type[n2] //n2= no. of columns in row-2
            array_name[] = new data_type[n3] //n3= no. of columns in row-3
                                .
                                .
                                .
            array_name[] = new data_type[nk]  //nk=no. of columns in row-n
```

https://www.javatpoint.com/jagged-array-in-java
https://www.scaler.com/topics/java/jagged-array-in-java/

Alternative, ways to Initialize a Jagged array :

```
int arr_name[][] = new int[][]  {
                new int[] {10, 20, 30 ,40},
                new int[] {50, 60, 70, 80, 90, 100},
                new int[] {110, 120}
                    };

          OR

int[][] arr_name = {
      new int[] {10, 20, 30 ,40},
      new int[] {50, 60, 70, 80, 90, 100},
      new int[] {110, 120}
          };

          OR

int[][] arr_name = {
       {10, 20, 30 ,40},
       {50, 60, 70, 80, 90, 100},
       {110, 120}
          };
```


Jagged Array [4] [ ]
Jagged Array [0]
Jagged Array [1]
Jagged Array [2]
Jagged Array [3]
2 5 9 0
1 2 5 6 8 1
0 1
3 5 5 5 6 7


Jagged_arr[3][]
Jagged_arr[0]
Jagged_arr[1]
Jagged_arr[2]
99 18 1 77
43 8
17 101 2

Jagged Arrays

Jagged arrays have the following advantages in Java:

- **Dynamic allocation**: Jagged arrays allow you to allocate memory dynamically, meaning that **you can specify the size of each sub-array at runtime**, rather than at compile-time.
- **Space utilization:** Jagged arrays can **save memory** when the size of each sub-array is not equal. In a rectangular array, all sub-arrays must have the same size, even if some of them have unused elements. With a jagged array, you can allocate just the amount of memory that you need for each sub-array.
- **Flexibility**: Jagged arrays can be useful when you need to store arrays of different lengths or when the number of elements in each sub-array is not known in advance.
- **Improved performance**: Jagged arrays can be **faster than rectangular arrays** for certain operations, such as accessing elements or iterating over sub-arrays, because the **memory layout is more compact.**

Code: https://www.geeksforgeeks.org/jagged-array-in-java/

```java
// Another Java program to demonstrate 2-D jagged
// array such that first row has 1 element, second
// row has two elements and so on.
class Main {
    public static void main(String[] args)
    {
        int r = 5;

        // Declaring 2-D array with 5 rows
        int arr[][] = new int[r][];

        // Creating a 2D array such that first row
        // has 1 element, second row has two
        // elements and so on.
        for (int i = 0; i < arr.length; i++)
            arr[i] = new int[i + 1];

        // Initializing array
        int count = 0;
        for (int i = 0; i < arr.length; i++)
            for (int j = 0; j < arr[i].length; j++)
                arr[i][j] = count++;

        // Displaying the values of 2D Jagged array
        System.out.println("Contents of 2D Jagged Array");
        for (int i = 0; i < arr.length; i++) {
            for (int j = 0; j < arr[i].length; j++)
                System.out.print(arr[i][j] + " ");
            System.out.println();
        }
    }
}
```

nested for loop

// loop through each row of the jagged array

// loop through each column of the current row

Output

```
Contents of 2D Jagged Array
0
1 2
3 4 5
6 7 8 9
10 11 12 13 14
```

Java Command Line Arguments

The java command-line argument is an argument i.e. passed at the time of running the java program. The arguments **passed from the console i.e., command prompt (not on Online Compilers)** can be received in the java program and it can be used as an input. So, it provides a convenient way to check the behavior of the program for the different values.

Simple example of command-line argument in java

In this example, we are receiving only one argument and printing it. To run this java program, you must pass at least one argument from the command prompt.

```
class CommandLineExample{
public static void main(String args[]){
System.out.println("Your first argument is: "+args[0]);
}
}
```

```
compile by > javac CommandLineExample.java
run by > java CommandLineExample sonoo
```

```
Output: Your first argument is: sonoo
```

Example of command-line argument that prints all the values

In this example, we are printing all the arguments passed from the command-line. For this purpose, we have traversed the array using for loop.

```
class A{
public static void main(String args[]){

for(int i=0;i<args.length;i++)
System.out.println(args[i]);

}
}
```

```
compile by > javac A.java
run by > java A sonoo jaiswal 1 3 abc
```

```
Output: sonoo
        jaiswal
        1
        3
        abc
```

Enhanced for loop

- New in Java 5.0
- a.k.a. the for-each loop
- useful short hand for accessing all elements in an array (or other types of structures) if no need to alter values
- alternative for iterating through a set of values

```
for(Type loop-variable : set-expression)
    statement
```

- logic error (not a syntax error) if try to modify an element in array via enhanced for loop

For-each loop with Array in Java

For-each is another array traversing technique like for loop, while loop, do-while loop introduced in Java5.

- It starts with the keyword **for** like a normal for-loop.
- Instead of declaring and initializing a loop counter variable, you declare a variable that is the same type as the base type of the array, followed by a colon, which is then followed by the array name.
- In the loop body, you can use the loop variable you created rather than using an indexed array element.
- It’s commonly used to iterate over an array or a Collections class (eg, ArrayList)

Syntax:

```
for (type var : array)
{
    statements using var;
  }
```

The above syntax is equivalent to:

```
for (int i=0; i<arr.length;
i++)
{
    type var = arr[i];
    statements using var;
  }
```

EXAMPLE

```
import java.io.*;

class Easy

{

    public static void main(String[] args)

    {

        // array declaration

        int ar[] = { 10, 50, 60, 80, 90 };

        for (int element : ar)

            System.out.print(element + " ");
    }
}
```

OUTPUT

```
10 50 60 80 90
```

```java
// Java program to illustrate
// for-each loop
class For_Each
{
    public static void main(String[] arg)
    {
        {
            int[] marks = { 125, 132, 95, 116, 110 };

            int highest_marks = maximum(marks);
            System.out.println("The highest score is " + highest_marks);
        }
    }
    public static int maximum(int[] numbers)
    {
        int maxSoFar = numbers[0];

        // for each loop
        for (int num : numbers)
        {
            if (num > maxSoFar)
            {
                maxSoFar = num;
            }
        }
    return maxSoFar;
    }
}
```

Output

```
The highest score is 132
```

https://www.geeksforgeeks.org/for-each-loop-in-java/
https://www.javatpoint.com/java-arraylist

ArrayList in Java

The `ArrayList` class is a resizable array, which can be found in the `java.util` package.

The difference between a built-in array and an `ArrayList` in Java, is that the size of an array cannot be modified (if you want to add or remove elements to/from an array, you have to create a new one). While elements can be added and removed from an `ArrayList` whenever you want.

Example

Create an `ArrayList` object called cars that will store strings:

```
import java.util.ArrayList; // import the ArrayList class

ArrayList<String> cars = new ArrayList<String>(); // Create an
ArrayList object
```

Add Items

The `ArrayList` class has many useful methods. For example, to add elements to the `ArrayList`, use the `add()` method:

Example

```
import java.util.ArrayList;

public class Main {
  public static void main(String[] args) {
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");
    System.out.println(cars);
  }
}
```

OUTPUT:

[Volvo, BMW, Ford, Mazda]

For more on ArrayList refer:

https://www.w3schools.com/java/java_arraylist.asp

Java Iterator

An `Iterator` is an object that can be used to loop through collections, like ArrayList and HashSet. It is called an "iterator" because "iterating" is the technical term for looping.

To use an Iterator, you must import it from the `java.util` package.

The `iterator()` method can be used to get an `Iterator` for any collection.

Syntax

```
Iterator itr = c.iterator();
```

Note: *Here "c" is any Collection object. itr is of type Iterator interface and refers to "c".*

Methods of Iterator Interface in Java

The iterator interface defines **three** methods as listed below:

1. hasNext(): Returns true if the iteration has more elements.

```
public boolean hasNext();
```

2. next(): Returns the next element in the iteration. It throws **NoSuchElementException** if no more element is present.

```
public Object next();
```

3. remove(): Removes the next element in the iteration. This method can be called only once per call to next().

```
public void remove();
```

Main.java

Run

```java
// Import the ArrayList class and the Iterator class
import java.util.ArrayList;
import java.util.Iterator;
public class Main {
  public static void main(String[] args) {
    // Make a collection
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");
    // Get the iterator
    Iterator<String> it = cars.iterator();
    // Print the first item
    System.out.println(it.next());
     System.out.println(it.hasNext());  //check if iterator has the elements
      System.out.println(it.next());   //printing the element and move to next
  }
}
```

Output Clear

```
java -cp /tmp/sOQt5EtwI1 Main
Volvo
true
BMW
```

Vectors

A Weakness of Arrays

- Suppose we declare an array of "Student" objects:

```
Student[] students = new Student[10];
```

- What if a new student joins the class?
- The size of an array cannot be increased after it is instantiated

Resizing an Array (the hard way)

```
Student[] students = new Student[10];

// do some stuff…
// now we need to add student 11

Student[] students2 = new Student[11];
for(int i = 0; i < students.length; i++) {
  students2[i] = students[i];
}
student[10] = new Student();
```

VECTORS

- In Java the concept of variable arguments to a function is achieved by the use of **Vector** class contained in the **java.util package.**
- This class can be used to create a **generic dynamic array** known as ***vector*** that can hold *objects of any type and any number.*
- The objects in this do not have to be homogenous.
- Array can be easily implemented as vectors.

VECTORS ARE CREATED AS ARRAY AS FOLLOWS:

```
Vector intVect = new Vector( ); // declaring without size
Vector list = new Vector(3); // declaring with size
```

- A vector can be declared without specifying any size explicitly.
- A vector can accommodate an unknown number of items. Even, when a size is specified, this can be overlooked and a different number of items may be put into the vector.

ADVANTAGES OF VECTOR OVER ARRAYS

- It is convenient to use vectors to store objects.
- A vector can be used to store a list of objects that may vary in size.
- We can add and delete objects from the list as and when required.

DISADVANTAGES OF VECTOR OVER ARRAYS

- We cannot directly store simple data types in a vector.
- We can only store objects. Therefore , we need to convert Simple types to objects. This can be done using the wrapper classes

Some Vector Methods are given as follows:

Vector Methods	Description
list.addElement(item)	It adds the item specified to the list at the end.
list.elementAt(n)	It gives the name of the nth object.
list.size()	It gives the number of objects present
list.removeElement(item)	It removes the specified item from the list.
list.removeElementAt(n)	It removes the item stored in the nth position of the list.
list.removeAllElements()	It removes all the elements in the list.
list.copyInto(array)	It copies all items from list to array.
list.insertElementAt(item, n)	It inserts the item at nth position.

Vectors

- Vector implements a DYNAMIC ARRAY.
- Vectors can hold objects of any type and any number.
- **Vector class** is contained in **java.util package**
- Vector is different from ARRAY in two ways:-
  1. Vector is synchronized.
  2. It contains many legacy methods that are not part of the collection framework.

1. Enumeration.
2. Iterator

Declaring VECTORS

Vector list = new Vector(); ⟶ Creates a default vector, which has an initial size 10.

Vector list = new Vector(int size); ⟶ Creates a vector, whose initial capacity is specified by size.

Vector list = new Vector(int size, int incr);

Creates a vector, whose initial capacity is specified by size and whose increment is specified by incr.

Vectors (Contd....)

- A vector can be declared without specifying any size explicitly.
- A vector without size can accommodate an unknown number of items.
- Even when size is specified, this can be overlooked and a different number of items may be put into the vector

IN CONTRAST, AN ARRAY MUST BE ALWAYS HAVE ITS SIZE SPECIFIED.

VECTOR METHODS

void **addElement**(Object ***element***)	The object specified by ***element*** is added to the vector
int **capacity()**	Returns the capacity of the vector
boolean **contains(**Object ***element***)	Returns **true** if **element** is contained by the vector, else **false**
void **copyInto**(Object ***array[]***)	The elements contained in the invoking vector are copied into the array specified by **array[]**
elementAt(int ***index***)	Returns the element at the location specified by ***index***
Object **firstElement().**	Returns the first element in the vector

VECTOR METHODS

void **insertElementAt**(Object ***element***, int ***index***)	*Adds* ***element*** to the vector at the location specified by ***index***
boolean **isEmpty()**	Returns **true** if Vector is empty, **else false**
Object **lastElement()**	Returns the last element in the vector
void **removeAllElements()**	Empties the vector. After this method executes, the size of vector is zero.
void **removeElementAt**(int ***index***)	Removes element at the location specified by ***index***
void **setElementAt**(Object ***element***, int ***index***)	The location specified by ***index*** is assigned ***element***

VECTOR METHODS

void **setSize**(int ***size***)	*Sets the number of elements in the vector to **size.** If the new size is less than the old size, elements are lost. If the new size is larger than the old, null elements are added*
int **size()**	Returns the number of elements currently in the vector

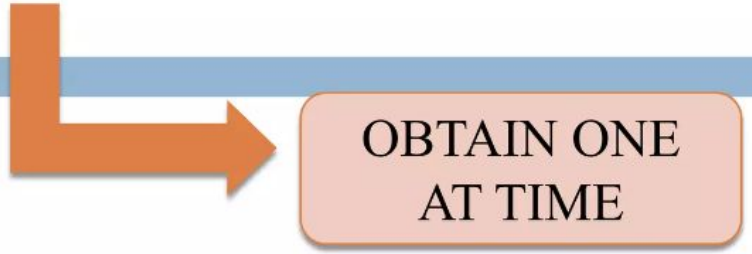
examples: https://www.programiz.com/java-programming/vector#google_vignette

More on Vectors:

https://www.upgrad.com/blog/vector-in-java/

https://www.scaler.com/topics/java/vector-in-java/

ENUMERATION INTERFACE



- Defines the methods by which you can enumerate the elements in a collection of objects

boolean hasMoreElements()

Returns true while there are still more elements to extract and false otherwise

Object nextElement()

Returns the next object in enumeration

Code: https://www.geeksforgeeks.org/java-program-to-iterate-vector-using-enumeration/

Using Iterator to Iterate over Vector Elements

Code: https://www.geeksforgeeks.org/iterate-over-vector-elements-in-java/

https://www.geeksforgeeks.org/difference-between-iterator-and-enumeration-in-java-with-examples/

https://www.geeksforgeeks.org/java-util-vector-class-java/

https://www.geeksforgeeks.org/java-program-to-iterate-vector-using-enumeration/

https://www.geeksforgeeks.org/enumeration-interface-in-java/

Comparator Interface for Sorting:
https://www.geeksforgeeks.org/comparator-interface-java/

Codes Link

https://docs.google.com/document/d/1tM6La9VpnZNpBbQ9tcV11IMaogQTkrpxjKrWp1_O1EU/edit?usp=sharing

WRAPPER CLASSES

- Since, vectors cannot handle primitive data types like **int**, **float**, **long** ,**char**, and **double**.
- Primitive data types may be converted into object types by using the wrapper classes contained in the **java.lang** packages.
- The wrapper classes have a number of unique methods for handling primitive data types and objects.

WRAPPER CLASSES FOR CONVERTING SIMPLE TYPES

Simple Type	Wrapper Class
boolean	Boolean
char	Character
double	Double
float	Float
int	Integer
long	Long

https://www.geeksforgeeks.org/wrapper-classes-java/

Tutorial Activity : for-each loop, ArrayList, Iterator, Vectors, 'super' keyword

1. WAJP using a for-each loop to print the lowest marks in an array storing marks of 10 students.

2. WAJP to traverse ArrayList elements (store names of 5 fruits) using the Iterator interface. Also demonstrate the use of add(), get() and set() methods.

Tutorial Activity : for-each loop, ArrayList, Iterator, Vectors, 'super' keyword

3. WAJP that accepts a shopping list of 5 items from the command line and stores them in a vector. Modify the program to accomplish the following:

a. delete an item in the list

b. add an item at a specified position in list

c. add an item at the end of list

d. print the contents of the vector

4. Demonstrate the use of 'super' keyword using a Java program. Here, Employee class (store 'salary' here) should inherit Person class, so all the properties (id, name) of Person should be inherited to Employee by default. To initialize all the properties, use parent class constructor from child class. Display id, name and salary of 3 employees as an output.

Sample Solutions to Tutorial Questions

1.

`class Vector`

- The `class Vector` can be used to implement a list, replacing a simple array
- The size of a Vector object can grow/shrink during program execution
- The Vector will automatically grow to accommodate the number of elements you put in it

`class Vector` (continued)

- The `class Vector` is contained in the package `java.util`
- Programs must include either:
  - `import java.util.*;`
  - `import java.util.Vector;`

Vector Declaration

- Declare/initialize

```
Vector<Student> students = new Vector<Student>();
```

- The syntax <…> is used to declare the type of object that will be stored in the Vector
- If you add a different type of object, an exception is thrown
- Not strictly necessary, but highly recommended (compiler warning)

Vector Size/Capacity

- The *size* of a vector is the number of elements
- The *capacity* of a vector is the maximum number of elements before more memory is needed
- If size exceeds capacity when adding an element, the capacity is automatically increased
  - Declares a larger storage array
  - Copies existing elements, if necessary
  - Growing the capacity is expensive!
- By default, the capacity doubles each time

Setting Initial Capacity

- If you know you will need a large vector, it may be faster to set the initial capacity to something large

```
Vector<Student> students = new Vector<Student>(1000);
```

Size and capacity

- Get the current size and capacity:

```
Vector<Student> students = new Vector<Student>(1000);
students.size();      // returns 0
students.capacity();  // returns 1000
```

- Setting size and capacity:

```
// adds null elements or deletes elements if necessary
students.setSize(10);

// increases capacity if necessary
students.ensureCapacity(10000);
```

Adding Elements

```
Vector<String> stringList = new Vector<String>();
```

```
stringList.add("Spring");
stringList.add("Summer");
stringList.addElement("Fall");
stringList.addElement("Winter");
```

add and addElement
have identical functionality

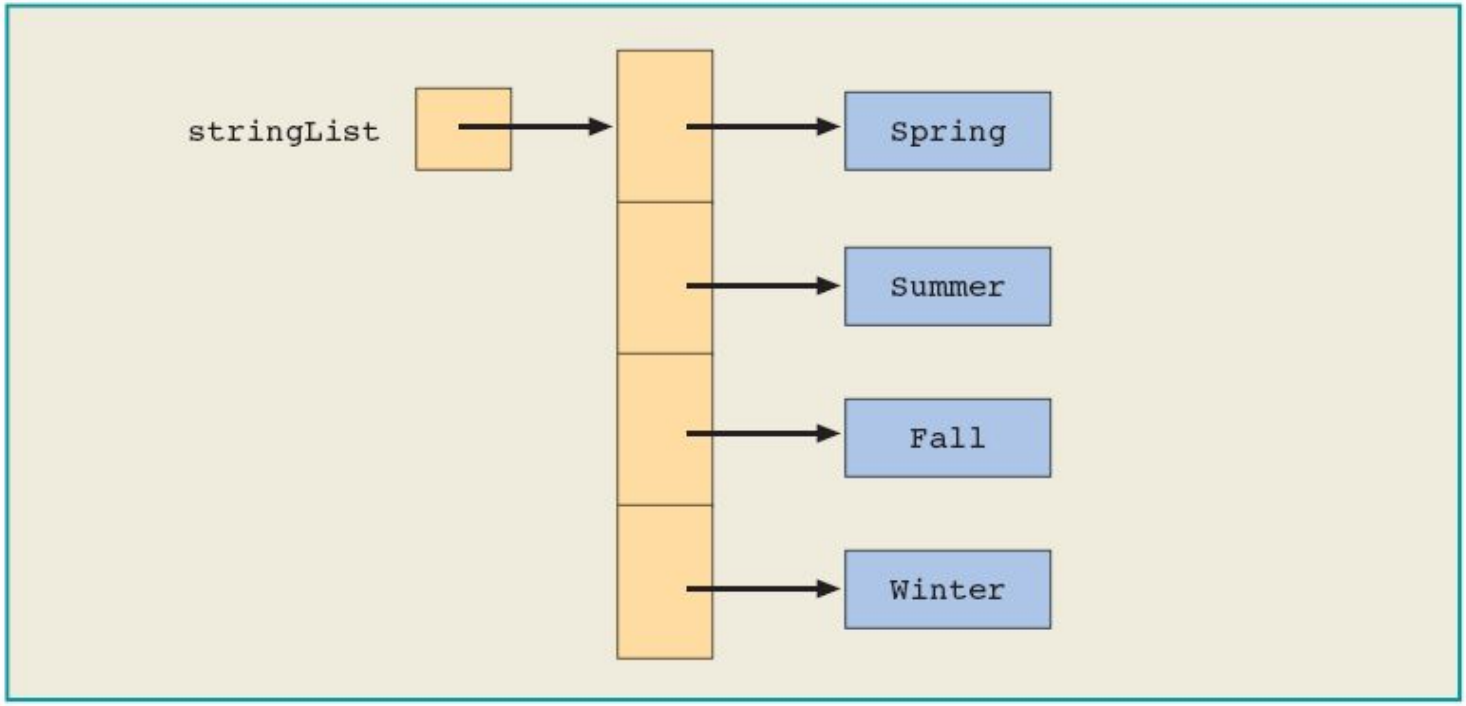


Figure 10-1 `stringList` after adding the four strings

Accessing Elements

```
stringList.get(0);   // "Spring"
stringList.get(3);   // "Winter"
stringList.get(4);   // ArrayIndexOutOfBounds Exception
```

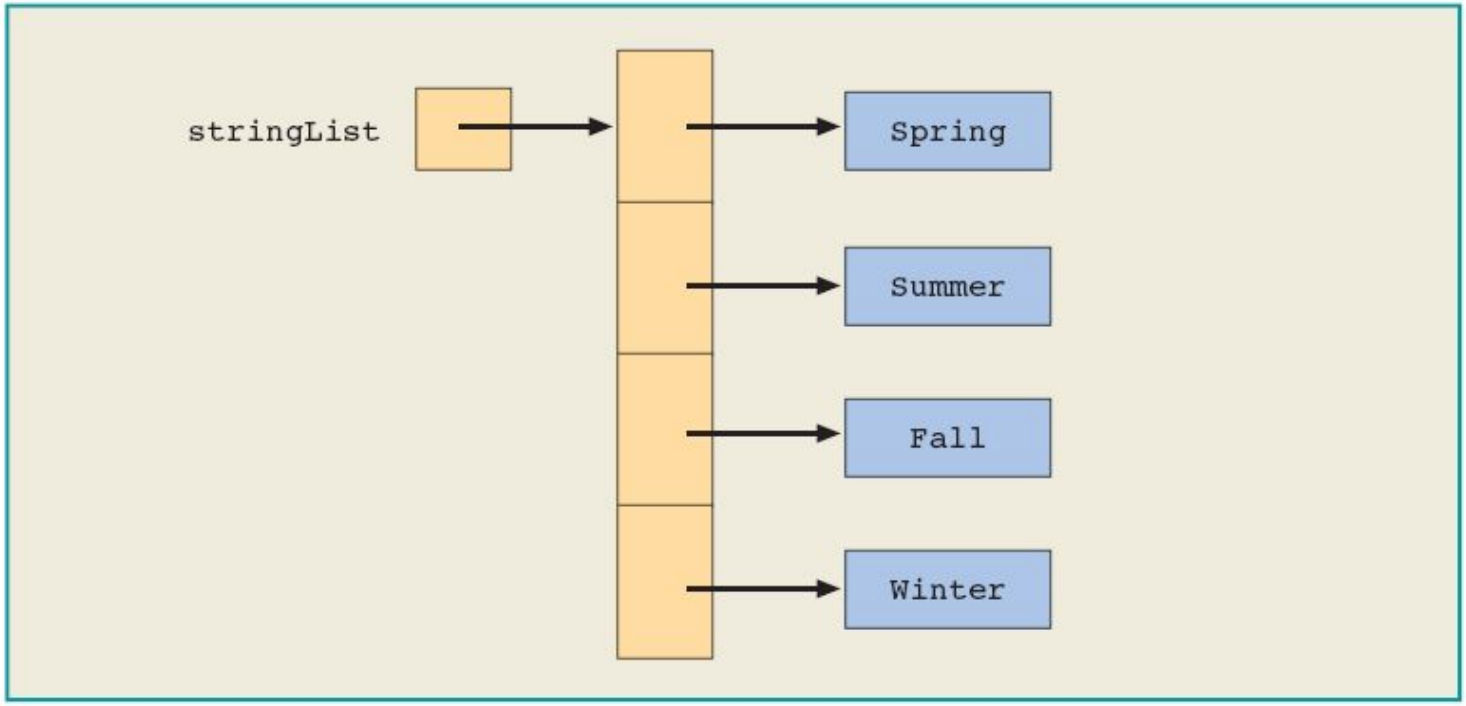


Figure 10-1 `stringList` after adding the four strings

Primitive Data Types and the `class Vector`

- Every component of a `Vector` object is a reference
  - Primitive data types are not objects
- Corresponding to each primitive data type, Java provides a wrapper class
- JDK 5.0 provides ***autoboxing*** and ***auto-unboxing*** of primitive data types

Autoboxing and Unboxing

1. Autoboxing

The automatic conversion of primitive types to the object of their corresponding wrapper classes is known as autoboxing. For example – conversion of int to Integer, long to Long, double to Double, etc.

2. Unboxing

It is just the reverse process of autoboxing. Automatically converting an object of a wrapper class to its corresponding primitive type is known as unboxing. For example – conversion of Integer to int, Long to long, Double to double, etc.

```java
// Java program to demonstrate Autoboxing

import java.util.ArrayList;
class Autoboxing {
    public static void main(String[] args)
    {
        char ch = 'a';

        // Autoboxing- primitive to Character object
        // conversion
        Character a = ch;

        ArrayList<Integer> arrayList
            = new ArrayList<Integer>();

        // Autoboxing because ArrayList stores only objects
        arrayList.add(25);

        // printing the values from object
        System.out.println(arrayList.get(0));
    }
}
```

Output

```
25
```

```java
// Java program to demonstrate Unboxing
import java.util.ArrayList;

class Unboxing {
    public static void main(String[] args)
    {
        Character ch = 'a';

        // unboxing - Character object to primitive
        // conversion
        char a = ch;

        ArrayList<Integer> arrayList
            = new ArrayList<Integer>();
        arrayList.add(24);

        // unboxing because get method returns an Integer
        // object
        int num = arrayList.get(0);

        // printing the values from primitive data types
        System.out.println(num);
    }
}
```

Output

```
24
```

Primitive Data Types and the `class Vector` (continued)

- Creating a `Vector` of `Integer` objects

```
Vector<Integer> list = new Vector<Integer>();


list.add(13);                  // with autoboxing
list.add(new Integer(25)); // without autoboxing

int tmp = list.get(0);      // with autounboxing
int tmp2 = list.get(0).intValue()   //without
```

`Vector` and the foreach loop

- Each `Vector` object is a collection of elements
  - You can use a foreach loop to process its elements
  - Exactly like using a foreach loop with an array
- Syntax:

```
for (type identifier : vectorObject)
  statements
```

Members of the `class Vector`

Table 10-1 Various Members of the `class Vector`

`protected int elementCount;` `protected Object[] elementData; //Array of references`
Constructors
`public Vector()` `  //Creates an empty vector of the default length of 10`
`public Vector(int size)` `  //Creates an empty vector of the length specified by size`
Methods
`public void addElement(Object insertObj)` `  //Adds the object insertObj at the end.`

`public void insertElementAt(Object insertObj, int index)`
`  //Inserts the object insertObj at the position specified by index.`
`  //If index is out of range, this method throws an`
`  //ArrayIndexOutOfBoundsException.`

Members of the `class Vector` (continued)

```
public Object clone()
  //Returns a copy of the vector.
public boolean contains(Object obj)
  //Returns true if this vector object contains the element specified
  //by obj; otherwise it returns false.
public void copyInto(Object[] dest)
  //Copies the elements of this vector into the array dest.
public Object elementAt(int index)
  //Returns the element of the vector at the location specified by index.
public Object firstElement()
  //Returns the first element of the vector.
  //If the vector is empty, the method throws a
  //NoSuchElementException.
public Object lastElement()
  //Returns the last element of the vector.
  //If the vector is empty, the method throws a
  //NoSuchElementException.
public int indexOf(Object obj)
  //Returns the position of the first occurrence of the element
  //specified by obj in the vector.
  //If obj is not in the vector, the method returns -1.
```

Members of the `class Vector` (continued)

Table 10-1 Various Members of the `class Vector` (continued)

Methods

`public int indexOf(Object obj, int index)`
`  //Starting at index, this method returns the position of the`
`  //first occurrence of the element specified by obj in the vector.`
`  //If obj is not in the vector, this method returns -1.`

`public boolean isEmpty()` `  //Returns true if the vector is empty; otherwise it returns false.`

`public int lastIndexOf(Object obj)`
`  //Starting at the last element, using a backward search, this`
`  //method returns the position of the first occurrence of the`
`  //element specified by obj in the vector.`
`  //If obj is not in the vector, this method returns -1.`

`public int lastIndexOf(Object obj, int index)`
`  //Starting at the position specified by index and using a backward`
`  //search, this method returns the position of the first occurrence`
`  //of the element specified by obj in the vector.`
`  //If obj is not in the vector, this method returns -1.`

`public void removeAllElements()` `  //Removes all the elements of the vector.`

Members of the `class` `Vector` (`continued`)

```
public boolean removeElement(Object obj)
  //If an element specified by obj exists in this vector, the
  //element is removed and the value true is returned; otherwise the
  //value false is returned.
public void removeElementAt(int index)
  //If an element at the position specified by index exists, it is
  //removed from the vector.
  //If index is out of range, this method throws an
  //ArrayIndexOutOfBoundsException.
public void setElementAt(Object obj, int index)
  //The element specified by obj is stored at the position specified by
  //index.
  //If index is out of range, this method throws an
  //ArrayIndexOutOfBoundsException.
public int size()
  //Returns the number of elements in the vector.
public String toString()
  //Returns a string representation of this vector.
```

Strings

- Strings are essentially **arrays of characters**
- Strings in Java are **immutable and final**
- The string class provides many functions for manipulating strings
  - Searching/matching operations
  - Replacing characters
  - Finding characters
  - Trimming whitespace
  - Etc.

STRINGS

- Strings represent a sequence of characters.
- The easiest way to represent a sequence of characters in JAVA is by using a character array.

```
char Array[ ] = new char [5];
```

Character arrays are not good enough to support the range of operations we want to perform on strings.

In JAVA strings are class objects and implemented using two classes:-

- ✓String
- ✓StringBuffer.

In JAVA strings are not a character array and is not NULL terminated.

DECLARING & INITIALISING

- Normally, objects in Java are created with the *new* keyword.

```
String name;
  name = new String("Craig");
```

OR

```
String name= new String("Craig");
```

- However, String objects can be created "implicitly":

```
String name;
name = "Craig";
```

The String Class

- String objects are handled specially by the compiler.
  - String is the only class which has "implicit" instantiation.
- The String class is defined in the **java.lang package.**
- Strings are immutable.
- The value of a String object can never be changed.
  - For mutable Strings, use the StringBuffer class.

STRING CONCATENATION

- JAVA string can be concatenated using + operator.

```
String name="Ankita";
String surname="Karia";
System.out.println(name+"  "+surname);
```

STRING Arrays

- An array of strings can also be created

```
String cities [ ] = new String[5];
```

- Will create an array of CITIES of size 5 to hold string constants

String Methods

- The **String** class contains many useful methods for string-processing applications.
  - A **String** method is called by writing a **String** object, a dot, the name of the method, and a pair of parentheses to enclose any arguments
  - If a **String** method returns a value, then it can be placed anywhere that a value of its type can be used

```
String greeting = "Hello";
int count = greeting.length();
System.out.println("Length is " + greeting.length());
```

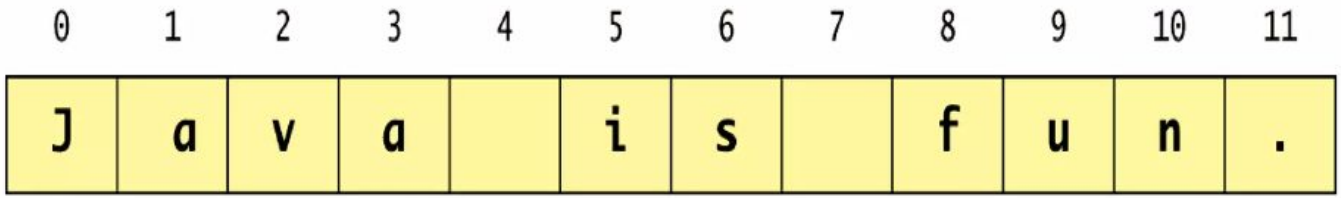
String method

  - Always count from zero when referring to the *position* or *index* of a character in a string

String Indexes

Display 1.5 String Indexes

The 12 characters in the string "Java is fun." have indexes 0 through 11.



Notice that the blanks and the period count as characters in the string.

Some Methods in the Class `String` (Part 1 of 8)

Display 1.4 Some Methods in the Class `String`

`int length()`

Returns the length of the calling object (which is a string) as a value of type `int`.

EXAMPLE

After program executes `String greeting = "Hello!";`
`greeting.length()` returns 6.

`boolean equals(Other_String)`

Returns `true` if the calling object string and the *Other_String* are equal. Otherwise, returns `false`.

EXAMPLE

After program executes `String greeting = "Hello";`
`greeting.equals("Hello")` returns `true`
`greeting.equals("Good-Bye")` returns `false`
`greeting.equals("hello")` returns `false`

Note that case matters. `"Hello"` and `"hello"` are not equal because one starts with an uppercase letter and the other starts with a lowercase letter.

Some Methods in the Class `String` (Part 2 of 8)

Display 1.4 Some Methods in the Class `String`

`boolean equalsIgnoreCase(`*Other_String*`)`

Returns `true` if the calling object string and the *Other_String* are equal, considering uppercase and lowercase versions of a letter to be the same. Otherwise, returns `false`.

EXAMPLE

After program executes `String name = "mary!";`
`greeting.equalsIgnoreCase("Mary!")` returns `true`

`String toLowerCase()`

Returns a string with the same characters as the calling object string, but with all letter characters converted to lowercase.

EXAMPLE

After program executes `String greeting = "Hi Mary!";`
`greeting.toLowerCase()` returns `"hi mary!"`.

Some Methods in the Class `String` (Part 3 of 8)

Display 1.4 Some Methods in the Class `String`

`String toUpperCase()`

Returns a string with the same characters as the calling object string, but with all letter characters converted to uppercase.

EXAMPLE

After program executes `String greeting = "Hi Mary!";`
`greeting.toUpperCase()` returns `"HI MARY!"`.

`String trim()`

Returns a string with the same characters as the calling object string, but with leading and trailing white space removed. Whitespace characters are the characters that print as white space on paper, such as the blank (space) character, the tab character, and the new-line character `'\n'`.

EXAMPLE

After program executes `String pause = "  Hmm  ";`
`pause.trim()` returns `"Hmm"`.

Some Methods in the Class `String` (Part 4 8)

Display 1.4 Some Methods in the Class `String`

`char charAt(`*Position*`)`

Returns the character in the calling object string at the *Position*. Positions are counted 0, 1, 2, etc.

EXAMPLE

After program executes `String greeting = "Hello!";`
`greeting.charAt(0)` returns `'H'`, and
`greeting.charAt(1)` returns `'e'`.

`String substring(`*Start*`)`

Returns the substring of the calling object string starting from *Start* through to the end of the calling object. Positions are counted 0, 1, 2, etc. Be sure to notice that the character at position *Start* is included in the value returned.

EXAMPLE

After program executes `String sample = "AbcdefG";`
`sample.substring(2)` returns `"cdefG"`.

https://www.geeksforgeeks.org/substring-in-java/

Some Methods in the Class `String` (Part 5 of 8)

Display 1.4 Some Methods in the Class `String`

`String substring(`*Start*`, `*End*`)`

Returns the substring of the calling object string starting from position *Start* through, but not including, position *End* of the calling object. Positions are counted 0, 1, 2, etc. Be sure to notice that the character at position *Start* is included in the value returned, but the character at position *End* is not included.

EXAMPLE

After program executes `String sample = "AbcdefG";`
`sample.substring(2, 5)` returns `"cde"`.

`int indexOf(`*A_String*`)`

Returns the index (position) of the first occurrence of the string *A_String* in the calling object string. Positions are counted 0, 1, 2, etc. Returns `-1` if *A_String* is not found.

EXAMPLE

After program executes `String greeting = "Hi Mary!";`
`greeting.indexOf("Mary")` returns 3, and
`greeting.indexOf("Sally")` returns `-1`.

Some Methods in the Class String (Part 6 of 8)

Display 1.4 Some Methods in the Class `String`

`int indexOf(A_String, Start)`

Returns the index (position) of the first occurrence of the string *A_String* in the calling object string that occurs at or after position *Start*. Positions are counted 0, 1, 2, etc. Returns −1 if *A_String* is not found.

EXAMPLE

After program executes `String name = "Mary, Mary quite contrary";`
`name.indexOf("Mary", 1)` returns `6`.
The same value is returned if `1` is replaced by any number up to and including 6.
`name.indexOf("Mary", 0)` returns `0`.
`name.indexOf("Mary", 8)` returns `−1`.

`int lastIndexOf(A_String)`

Returns the index (position) of the last occurrence of the string *A_String* in the calling object string. Positions are counted 0, 1, 2, etc. Returns −1, if *A_String* is not found.

EXAMPLE

After program executes `String name = "Mary, Mary, Mary quite so";`
`greeting.indexOf("Mary")` returns `0`, and
`name.lastIndexOf("Mary")` returns `12`.

Some Methods in the Class `String` (Part 7

Display 1.4 Some Methods in the Class `String`

`int compareTo(A_String)`

Compares the calling object string and the string argument to see which comes first in the lexicographic ordering. Lexicographic order is the same as alphabetical order but with the characters ordered as in Appendix 3. Note that in Appendix 3 all the uppercase letters are in regular alphabetical order and all the lowercase letters are in alphabetical order, but all the uppercase letters precede all the lowercase letters. So, lexicographic ordering is the same as alphabetical ordering provided both strings are either all uppercase letters or both strings are all lowercase letters. If the calling string is first, it returns a negative value. If the two strings are equal, it returns zero. If the argument is first, it returns a positive number.

EXAMPLE

After program executes `String entry = "adventure";`
`entry.compareTo("zoo")` returns a negative number,
`entry.compareTo("adventure")` returns 0, and
`entry.compareTo("above")` returns a positive number.

Display 1.4 Some Methods in the Class String

`int compareToIgnoreCase(A_String)`

Compares the calling object string and the string argument to see which comes first in the lexicographic ordering, treating uppercase and lowercase letters as being the same. (To be precise, all uppercase letters are treated as if they were their lowercase versions in doing the comparison.) Thus, if both strings consist entirely of letters, the comparison is for ordinary alphabetical order. If the calling string is first, it returns a negative value. If the two strings are equal ignoring case, it returns zero. If the argument is first, it returns a positive number.

EXAMPLE

After program executes `String entry = "adventure";`
`entry.compareToIgnoreCase("Zoo")` returns a negative number,
`entry.compareToIgnoreCase("Adventure")` returns `0`, and
`"Zoo".compareToIgnoreCase(entry)` returns a positive number.

STRING BUFFER CLASS

- STRINGBUFFER class creates strings flexible length that can be modified in terms of both length and content.
- STRINGBUFFER may have characters and substrings inserted in the middle or appended to the end.
- STRINGBUFFER automatically grows to make room for such additions

Actually STRINGBUFFER has more characters pre allocated than are actually needed, to allow room for growth

STRING BUFFER FUNCTIONS

- **append*(s2):-*** Appends string s2 to s1 at the end.
- **insert*(n,s2):-*** Inserts the string s2 at the position n of the string s1
- **reverse**():- Returns the reversed object on when it is called.
- **delete*(int n1,int n2):-*** Deletes a sequence of characters from the invoking object.

n1 → Specifies index of first character to remove

n2 → Specifies index one past the lastcharacter to remove

- **deleteCharAt*(int loc):-*** Deletes the character at the index specified by loc.

STRING BUFFER FUNCTIONS

- ***replace(int n1,int n2,String s1):-*** Replaces one set of characters with another set.
- **substring(*int startIndex*)**:- Returns the substring that starts at starts at ***startIndex*** and runs to the end.
- **substring(*int startIndex, int endIndex*)**:- Returns the substring that starts at starts at ***startIndex*** and runs to the ***endIndex-1***

Comparing Strings

- A `String` is a reference type and so they are NOT compared with `==`.
- The `String` class has a method `compareTo()` to compare 2 strings.
  - The characters in each string are compared one at a time from left to right, using the collating sequence.
  - The comparison stops after a character comparison results in a mismatch, or one string ends before the other.
    - If str1 < str2, then `compareTo()` returns an `int` < 0
    - If str1 > str2, then `compareTo()` returns an `int` > 0
  - If the character at every index matches, and the strings are the same length, the method returns `0`

`class String` (Revisited)

Table 10-2 Additional `String` Methods

```
String(String str)
  //Creates a String object and initializes the string
  //object with characters specified by str.
char charAt(int index)
  //Returns the character at the position specified by index.
int indexOf(char ch)
  //Returns the index of the first occurrence of the character
  //specified by ch; if the character specified by ch does not
  //appear in the string, it returns -1.
int indexOf(char ch, int pos)
  //Returns the index of the first occurrence of the character
  //specified by ch; the parameter pos specifies from where to begin
  //the search; if the character specified by ch does not
  //appear in the string, it returns -1.
```

`class String` (Revisited) (continued)

Table 10-2 Additional `String` Methods (continued)

```
int indexOf(String str)
  //Returns the index of the first occurrence of the string
  //specified by str; if the string specified by str does not
  //appear in the string, it returns -1.
int indexOf(String str, int pos)
  //Returns the index of the first occurrence of the string
  //specified by str; the parameter pos specifies where to begin
  //the search; if the string specified by str does not appear
  //in the string, it returns -1.
int compareTo(String str)
  //Compares two strings character-by-character.
  //Returns a negative value if this string is less than str.
  //Returns 0 if this string is the same as str.
  //Returns a positive value if this string is greater than str.
String concat(String str)
  //Returns the string that is this string concatenated with str.
boolean equals(String str)
  //Returns true if this string is the same as str.
int length()
  //Returns the length of this string.
```

class String (Revisited) (continued)

```
String replace(char charToBeReplaced, char charReplacedWith)
  //Returns the string in which every occurrence of
  //charToBeReplaced is replaced with charReplaced.
String substring(int startIndex, int endIndex)
  //Returns the string that is a substring of this string
  //starting at startIndex until endIndex - 1.
String toLowerCase()
  //Returns the string that is the same as this string except that
  //all uppercase letters of this string are replaced with
  //their equivalent lowercase letters.
String toUpperCase()
  //Returns the string that is the same as this string except
  //that all lowercase letters of this string are replaced with
  //their equivalent uppercase letters.
boolean startsWith(String str)
  //Returns true if the string begins with the string specified by
  //str; otherwise, this methods returns false.
boolean endsWith(String str)
  //Returns true if the string ends with the string specified by str;
  //otherwise, this methods returns false.
boolean regionMatches(int ind, String str, int strIndex, int len)
  //Returns true if the substring of str starting at strIndex and the
  //length specified by len is the same as the substring of this
  //String object starting at ind and having the same length.
```

class String (Revisited) (continued)

Table 10-2 Additional String Methods (continued)

```
boolean regionMatches(boolean ignoreCase, int ind,
                      String str, int strIndex, int len)
  //Returns true if the substring of str starting at strIndex and the
  //length specified by len is the same as the substring of this
  //String object starting at ind and having the same length. If
  //ignoreCase is true, then during character comparison the case is
  //ignored.
```

https://learnprogramo.com/menu-driven-program-in-java-program-with-explanation/

Java toString() Method

https://www.geeksforgeeks.org/string-tostring-method-in-java/

STRINGS

- The simplest way to represent a sequence of characters in java is by using a character array.
- Example:

```
char  charArray[ ] = new char[4];
charArray[0]  = 'J';
charArray[1]  = 'a';
```

- In Java, strings are class objects and implemented using two classes,namely, **String** and **StringBuffer.**
- A java string is an instantiated object of the String class.

DECLARATION OF STRING

```
String    stringName;
StringName  =  new  String ("string");
```

- Example:

```
String  firstName;
firstName  =  new  String("Anil");
```

STRING ARRAYS

- Array can be created and used that contain strings.

```
String itemArray[ ] = new String [3];
```

- Above statement create following effects:

- An itemArray of size 3 to hold three string constants.
- Strings can be assign to itemArray element:-
  - using three different statements
  - more efficiently using a for loop

STRING METHODS

- String class defines a number of methods that allow us to accomplish a variety of string manipulation tasks.

STRING BUFFER CLASS

- StringBuffer is a peer class of String.
- While String creates strings of fixed_length, StringBuffer creates strings of flexible length that can be modified in term of both length and content.
- We can insert characters and substrings in the middle of a string to the end.

The `String` Class

- The **`String`** class defines a data type that is used to store a sequence of characters
- **You cannot modify a `String` object**
  - If you attempt to do so, Java will create a new object that contains the modified character sequence

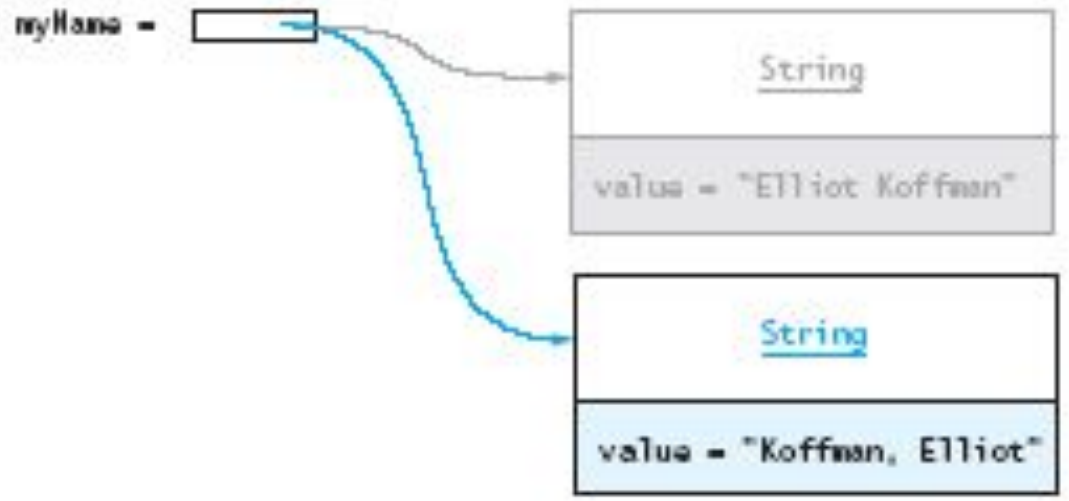


FIGURE A.4
Old and New Strings Referenced by myName

Comparing Objects

- You ***can't use the relational or equality operators*** to compare the values stored in strings (or other objects)

  (You will compare the *pointers*, not the *objects*!)

FIGURE A.5
Two String Objects at Different Addresses with the Same Contents

The `StringBuffer` Class

- Stores character sequences
- Unlike a **`String`** object, you ***can*** change the contents of a **`StringBuffer`** object

TABLE A.8
`StringBuffer` Methods in `java.lang.StringBuffer`

Method	Behavior
`void StringBuffer append(`*anyType*`)`	Appends the string representation of the argument to this `StringBuffer`. The argument can be of any data type.
`int capacity()`	Returns the current capacity of this `StringBuffer`.
`void StringBuffer delete(int start, int end)`	Removes the characters in a substring of this `StringBuffer`, starting at position `start` and ending with the character at position `end - 1`.
`void StringBuffer insert(int offset,` *anyType* `data)`	Inserts the argument `data` (any data type) into this `StringBuffer` at position `offset`, shifting the characters that started at `offset` to the right.
`int length()`	Returns the length (character count) of this `StringBuffer`.
`StringBuffer replace(int start, int end, String str)`	Replaces the characters in a substring of this `StringBuffer` (from position `start` through position `end - 1`) with characters in the argument `str`. Returns this `StringBuffer`.
`String substring(int start)`	Returns a new string containing the substring that begins at the specified index `start` and extends to the end of this `StringBuffer`.
`String substring(int start, int end)`	Return a new string containing the substring in this `StringBuffer` from position `start` through position `end - 1`.
`String toString()`	Returns a new string that contains the same characters as this `StringBuffer` object.

Write a Java program to find the longest Palindromic Substring within a string.

Sol.
https://www.w3resource.com/java-exercises/string/java-string-exercise-32.php

Logic:
https://www.geeksforgeeks.org/longest-palindromic-substring/

https://www.geeksforgeeks.org/longest-palindromic-substring-using-dynamic-programming-2/

https://docs.google.com/document/d/1tM6La9VpnZNpBbQ9tcV11IMaogQTkrpxjKrWp1_O1EU/edit?usp=sharing

StringBuilder Class

https://www.geeksforgeeks.org/stringbuilder-class-in-java-with-examples/

https://www.javatpoint.com/StringBuilder-class


The End!