


NPTEL ONLINE CERTIFICATION COURSES

Data Structures and Algorithms Using Java

Debasis Samanta

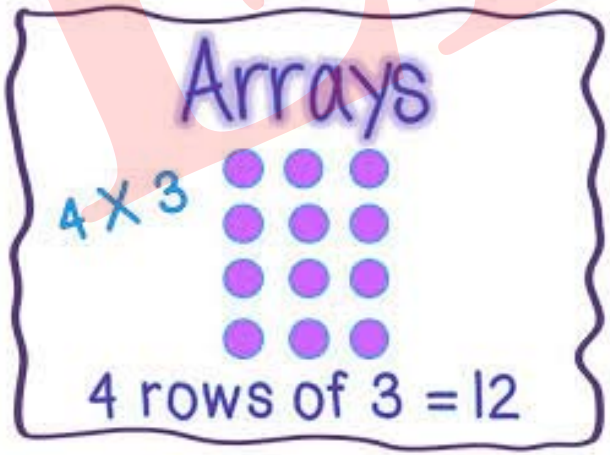
Department of Computer Science & Engineering, IIT Kharagpur

Module 04: Array

Lecture 11 : Array Data Structures

CONCEPTS COVERED

- Concept of Array Data Structures
- Characteristics of an Array
- Types of Array Structures
  - 1D Array
  - Multidimensional Arrays
- Operations on Arrays



Concept of Array Structures

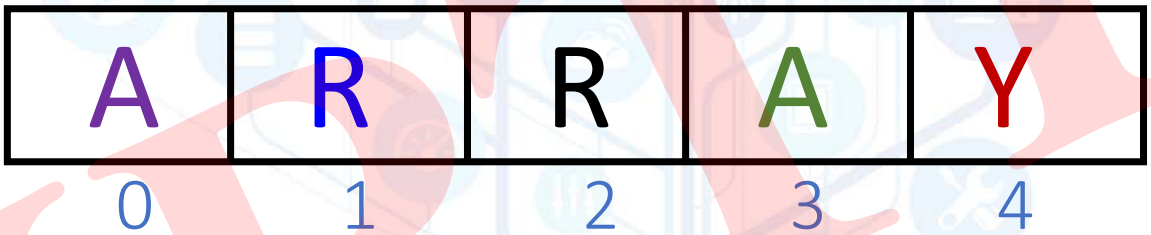
Concept of array

?

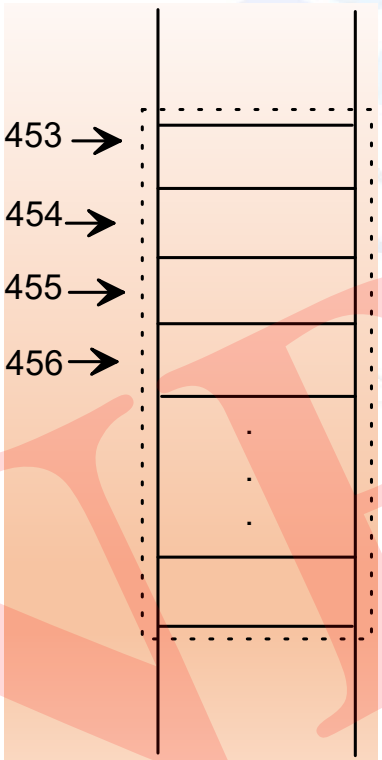
?

?

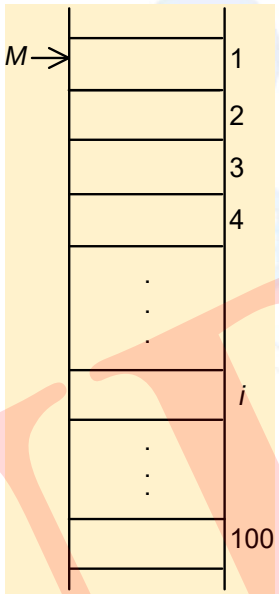
Concept of array

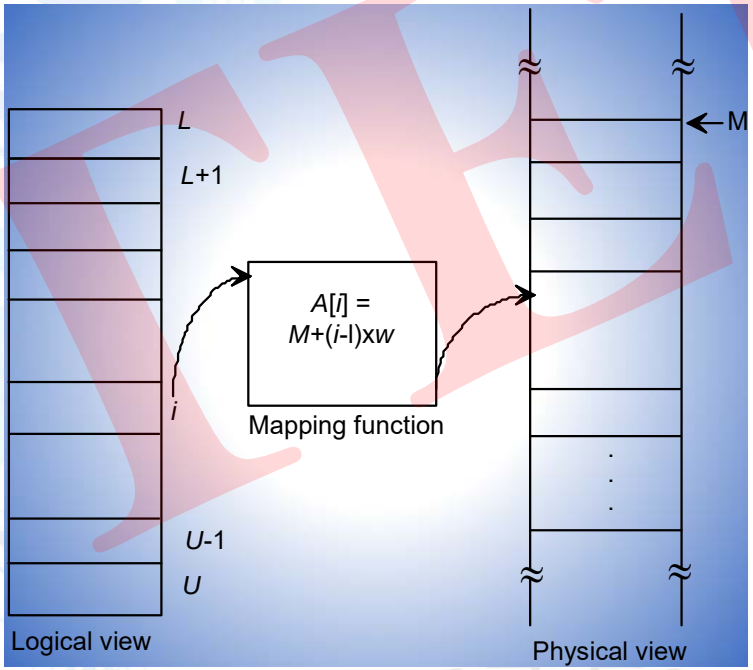


Concept of array



Concept of array





Address (A[i]) = $M + (i - L)$ x w

Size (A) = $U - L + 1$

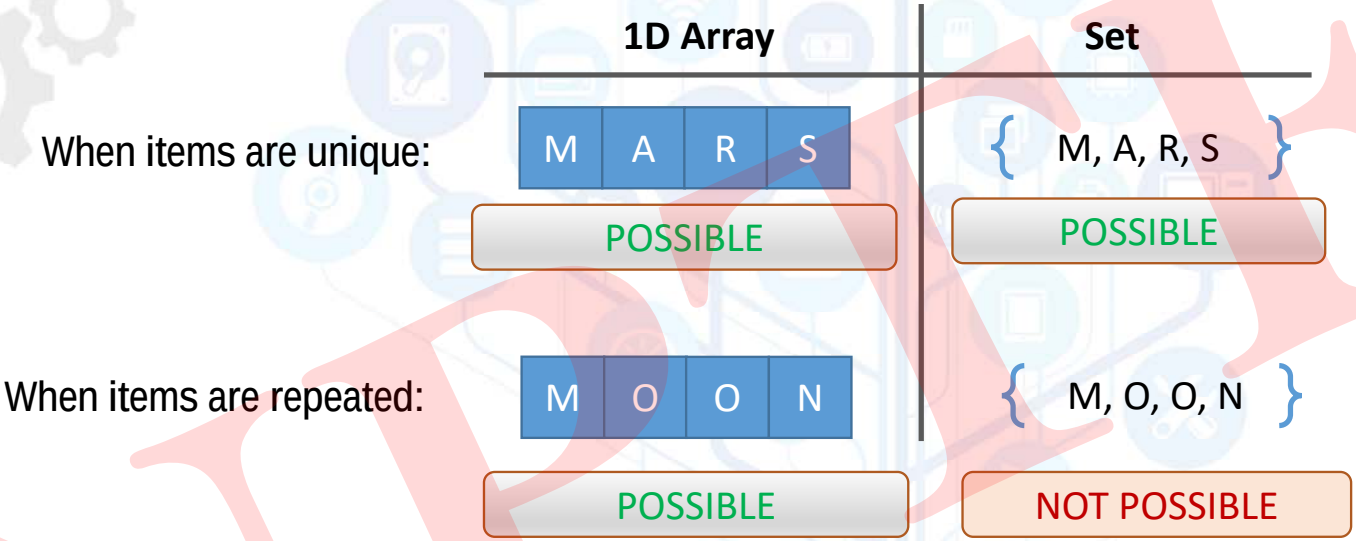
Characteristics of Array Structures

Concept of array

An *array* is a finite, ordered and collection of homogeneous data elements.

- **Finite:** Because it contains only a limited number of elements.
- **Ordered:** All the elements are stored one by one in contiguous locations of computer memory in a linear fashion.
- **Homogeneous:** All elements of an array are of the same data type only.

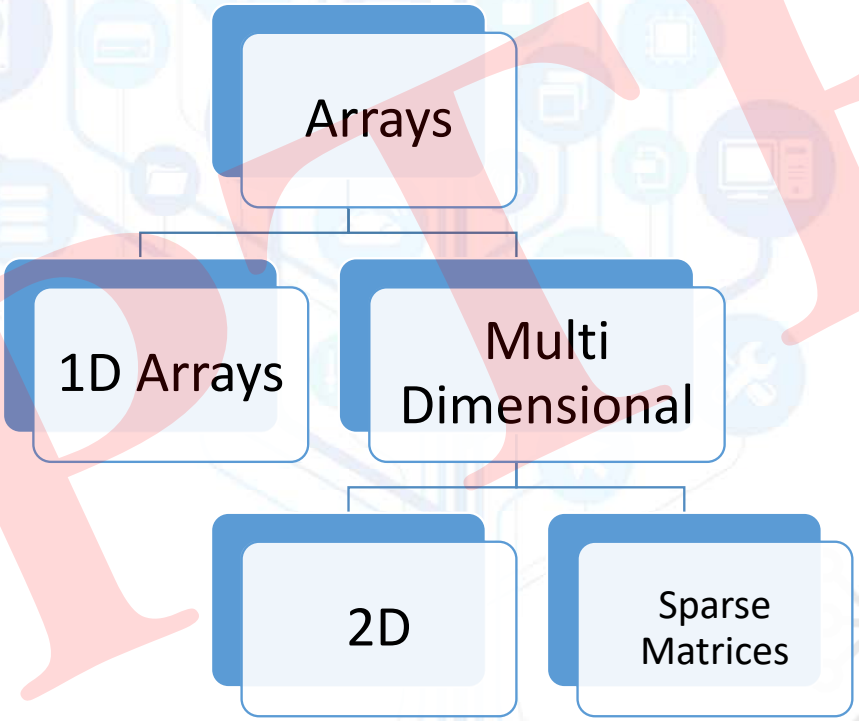
Concept of array



A SET is an array, but an ARRAY is not necessarily a set.

Types of Array Structures

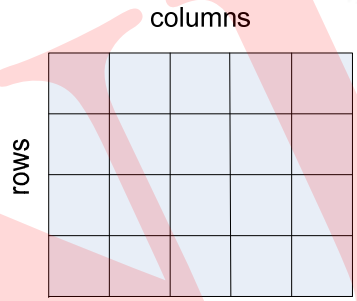
Types of arrays



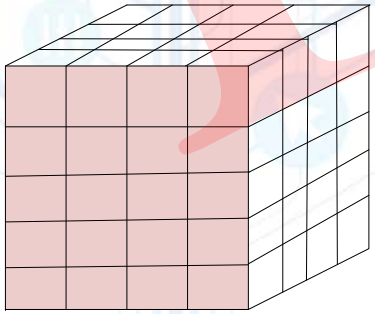
Types of arrays

`marks.length = 16`

1D array



2D array



3D array

Multidimensional Arrays

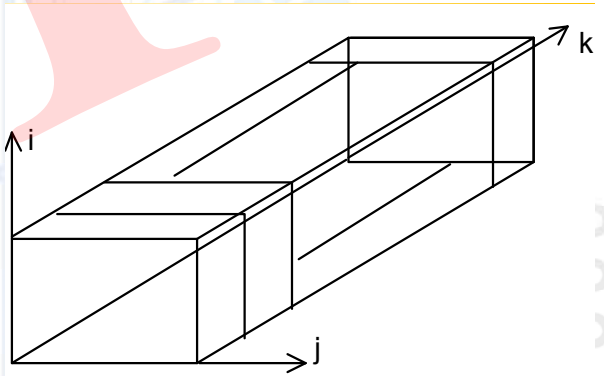
Multidimensional arrays

- More than one indexing to specify a location

2D: row, column

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & a_{24} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & a_{m3} & a_{m4} & \dots & a_{mn} \end{bmatrix}_{m \times n}$$

3D: row, column, height, etc.



Multidimensional arrays: Indexing

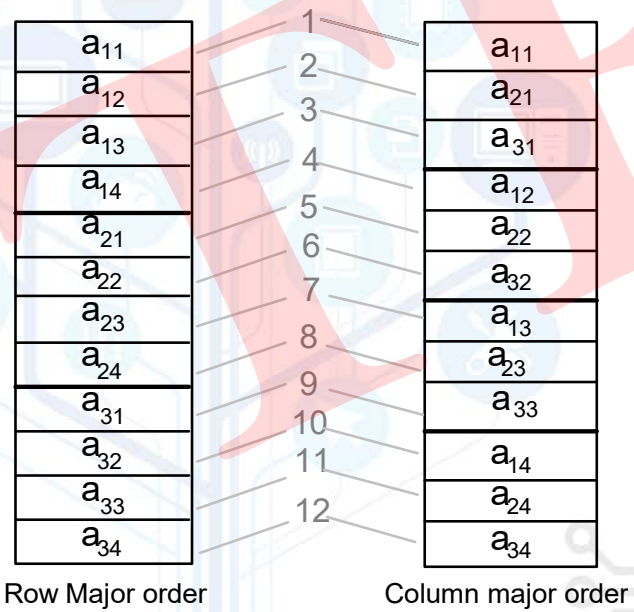
Two-dimensional arrays (alternatively termed as matrices) are the collection of homogeneous elements where the elements are ordered in a number of rows and columns

Indexing: *row, column*

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & a_{24} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & a_{m3} & a_{m4} & \dots & a_{mn} \end{bmatrix}_{m \times n}$$

Multidimensional arrays: Memory representation

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{bmatrix}_{3\times4}$$

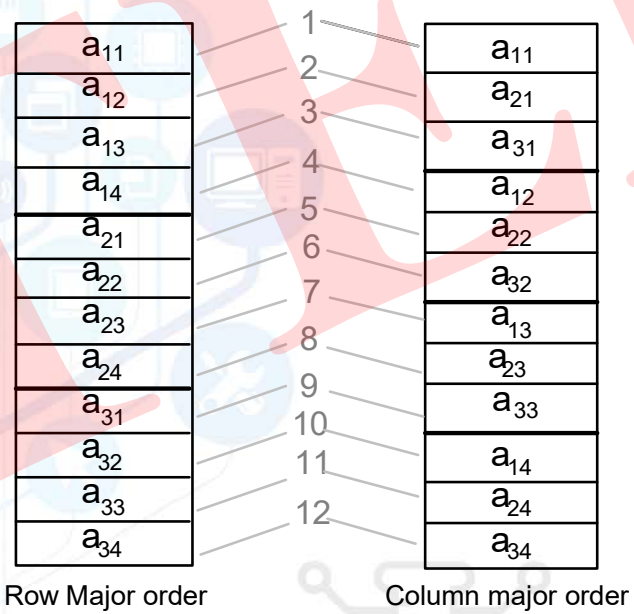


Row Major order

Column major order

Multidimensional arrays: Memory representation

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{bmatrix}_{3\times4}$$



- **Row-major order**

$$\text{Address}(a_{ij}) = M + (i-1) \times n + j - 1$$

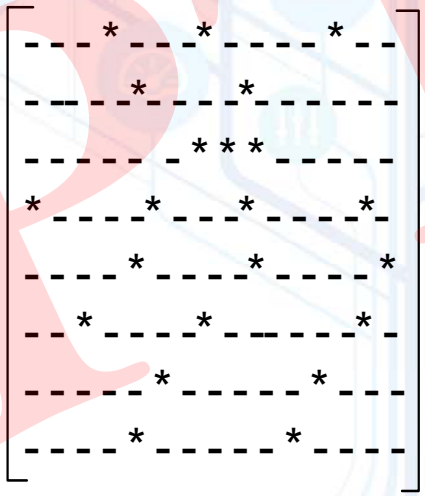
- **Column-major order**

$$\text{Address}(a_{ij}) = M + (j-1) \times m + i - 1$$

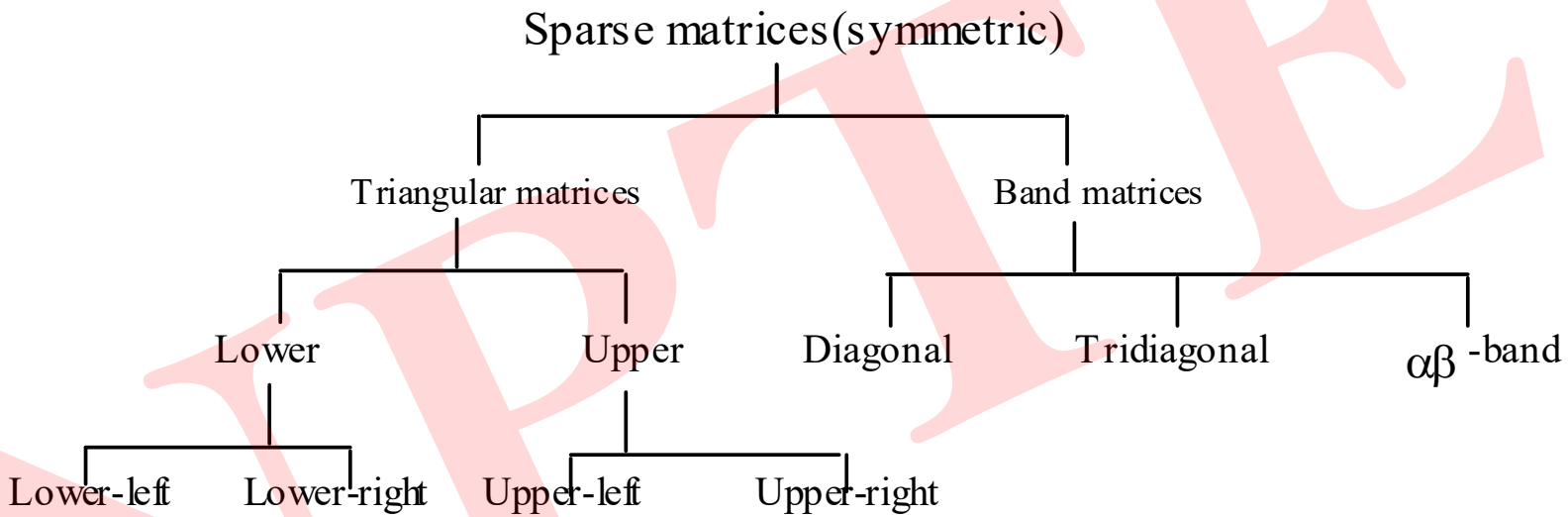
Sparse Matrices

Sparse matrix

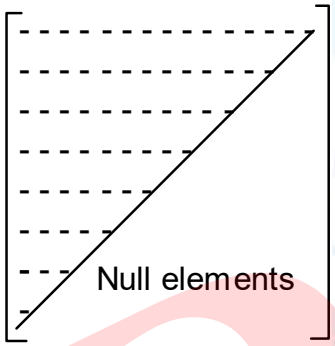
- A *sparse* matrix is a two-dimensional array having the value of majority elements as null



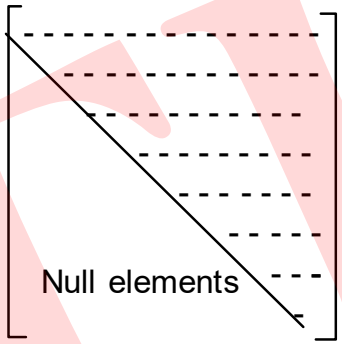
Sparse matrices



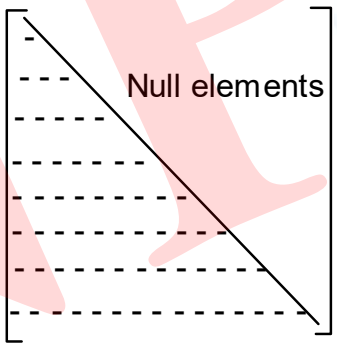
Triangular sparse matrices



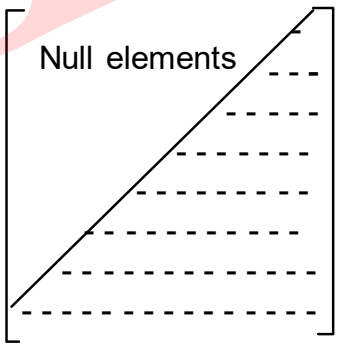
Upper left triangular



Upper right triangular

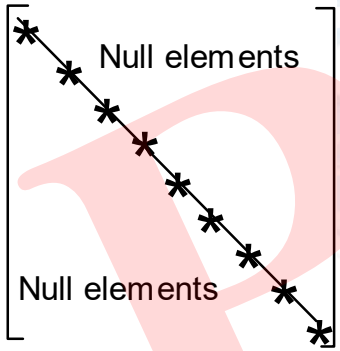


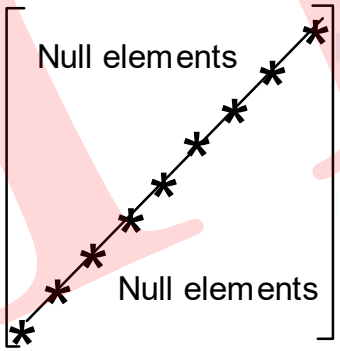
Lower left triangular



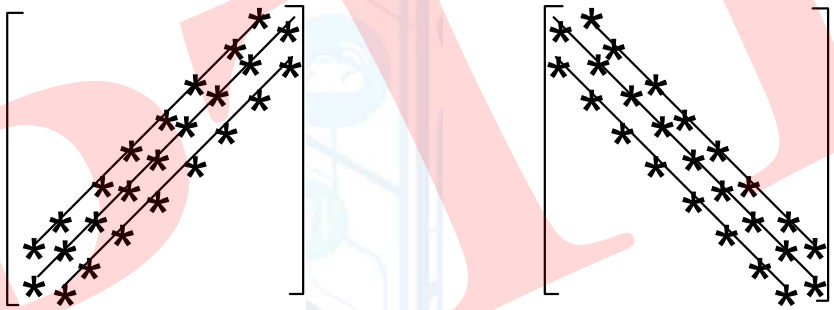
Lower right triangular

Diagonal sparse matrices





Tri-diagonal sparse matrices



Sparse matrices: Memory representation

- Lower triangular matrix: Row-major order

$$\text{Address}\ (a_{ij}) = M + \frac{i(i-1)}{2} + j - 1$$

$$\begin{bmatrix} a_{11} & & & & \\ a_{21} & a_{22} & & & \\ a_{31} & a_{32} & a_{33} & & \\ \vdots & \vdots & \vdots & & \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix}_{n \times n}$$

Sparse matrices: Memory representation

- Lower triangular matrix: Column-major order

Address (*aij*) = $M + (j-1) \times \left(n - \frac{j}{2} \right) + i - 1$

$$\begin{bmatrix} a_{11} & & & & \\ a_{21} & a_{22} & & & \\ a_{31} & a_{32} & a_{33} & & \\ \vdots & \vdots & \vdots & & \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn} \end{bmatrix}_{n \times n}$$

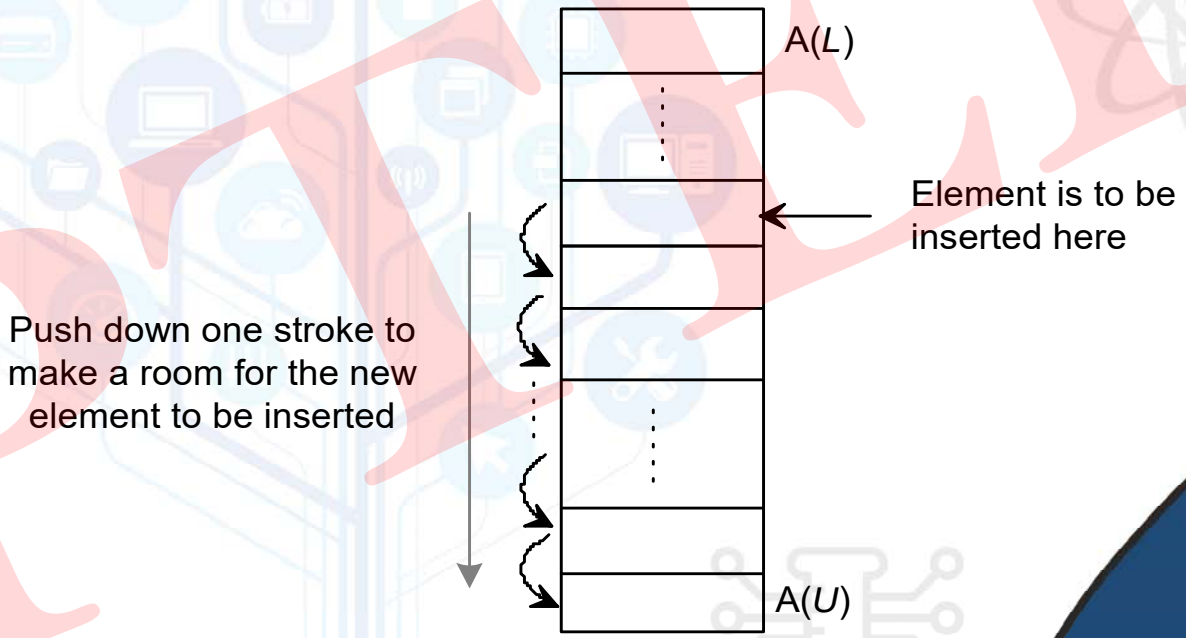
Operations with Arrays

Operations: 1D Arrays

- Insertion
- Deletion
- Traversal
- Searching
- Sorting

Operations: 1D Array insertion

- Insertion
- Deletion
- Traversal
- Searching
- Sorting



Operations: 1D Array insertion algorithm

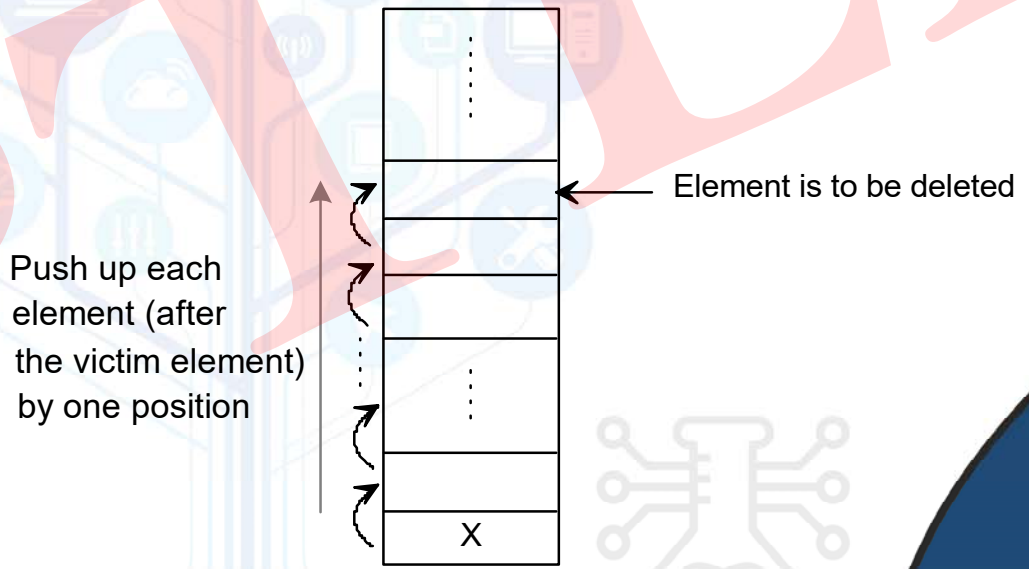
- Insertion
- Deletion
- Traversal
- Searching
- Sorting

Let A be an 1D Array with N elements and K is a positive integer such that K<=N. Following is the algorithm where ITEM is inserted into the K-th position of A.

```
1. Start
2.      Set J = N
3.      Set N = N+1
4.      While J > = K
5.              Set A[J+1] = A[J]
6.              Set J = J-1
7.      End
7.      Set A[K] = ITEM
8. Stop
```

Operations: 1D Array deletion

- Insertion
- Deletion
- Traversal
- Searching
- Sorting



Operations: 1D Array deletion algorithm

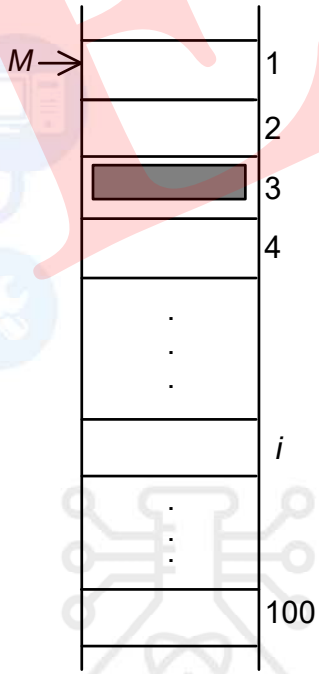
- Insertion
- Deletion
- Traversal
- Searching
- Sorting

Consider A is an 1D array with N elements and K is a positive integer such that K <= N. Following is the algorithm to delete an element available at the K-th position of A.

```
1. Start
2.      Set J = K
3.      While J < N
4.           Set LA[J] = LA[J + 1]
5.           Set J = J+1
6.       End
6.      Set N = N-1
7. Stop
```

Operations: 1D Array traversal

- Insertion
- Deletion
- Traversal
- Searching
- Sorting



Operations: 1D Array traversal algorithm

- Insertion
- Deletion
- Traversal
- Searching
- Sorting

Consider A is an 1D array with N elements. Following is the algorithm to traverse the entire array A to find MIN, MAX and AVERAGE.

```
1.  Start
2.  MIN = A[1], MAX = A[1], SUM = A[1]
3.  Set J = 2
3.      While  J <= N
4.          If A[J] < MIN Then Set MIN = A[j]
5.          If A[J] > MAX Then Set MAX = A[j]
6.          SUM = SUM + A[J]
5.           Set J = J+1
6.       End
6.       Print MIN, MAX, SUM/N
7.  Stop
```

Operations: 1D Array sorting and searching

- Insertion
- Deletion
- Traversal
- Searching
- Sorting

Sorting and Searching operations will be discussed in Week-9 and Week-10, respectively.

Operations with 2D Arrays

Operations: 2D Arrays

- Matrix traversal
- Matrix addition / subtraction
- Matrix multiplication

Operations: 2D Array traversal algorithm

- Matrix traversal
- Matrix addition / subtraction
- Matrix multiplication

1. Consider a 2D matrix with ***M*** rows and ***N*** columns.
2. The index number starts at 0 and counts till (M-1) (for rows) and (N-1) (for columns).

```
1. Start
2.     Set R = 0 and C = 0.
3.     While R < M.
4.         While C < N.
5.             Print A[R][C]
6.         End
7.     End
8. Stop
```

Operations: 2D Array addition/ subtraction algorithm

- Matrix. traversal
- Matrix addition/ subtraction
- Matrix multiplication

1. Consider a 2D matrix, say A with ***M*** rows and ***N*** columns.
2. Consider another 2D matrix, say B with ***P*** rows and ***Q*** columns.
3. Let, Y[M][N] be the output matrix, initially empty.

1. Start
2. Set R = 0, C = 0,
3. If M ≠ P and N ≠ Q Then Step 9
4. While R < M.
5. While C < N.
6. Y[R][C] = A[R][C] ± B[R][C].
7. End
8. End
9. Stop

A B

Operations: 2D Array multiplication algorithm

- Matrix traversal
- Matrix addition/ subtraction
- Matrix multiplication

1. Consider a 2D matrix, say A with ***M*** rows and ***N*** columns.
2. Consider another 2D matrix, say B with ***P*** rows and ***Q*** columns.
3. Let, Y[M][Q] be the output matrix, initially empty.

```
1.  Start
2.  If N ≠ P Then Step 9.
3.  Set AR = 0, AC = 0, BR = 0, BC = 0, R = 0, C = 0
4.  While AR < M and AC < N
5.      While BR < P and BC < Q
6.          While R < M and C < Q
7.              Y[R][C] = Y[R][C] + A[AR][AC]*B[BR][BC]
8.          End
9.      End
10. End
11. Stop
```

Application of Arrays

Collection of data in arrays

1D Array

1D/2D Array

3D Array

REFERENCES

- Classic Data Structures, Debasis Samanta, 2nd Edition, Prentice Hall of India
- https://cse.iitkgp.ac.in/~dsamanta/javads/index.html

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

योगः कर्मसु कौशलम्

THANK YOU !

NPTEL ONLINE CERTIFICATION COURSES

Data Structures and Algorithms Using Java

Debasis Samanta

Department of Computer Science & Engineering, IIT Kharagpur

Module 04: Arrays

Lecture 12 : Programming for Arrays

CONCEPTS COVERED

- Declaration of arrays
  - 1D arrays
  - 2D arrays
  - 3D arrays
- Initialization of arrays
- Operations on Arrays
  - Traversal
  - Insertion
  - Deletion
  - Matrix operations

Declarations of Arrays

Declarations of arrays

- In a program an array should be declared as a variable.
- Like any variable declaration, it is important to declare name, type and size of an array to be used in a program.
- Declarations of arrays in Java.
  - 1D arrays
  - 2D arrays
  - 3D arrays

Declarations of 1D Arrays

Declarations of 1D arrays: Syntax

A one-dimensional array can be **declared** with any one of the following ways:

Way 1: `<type> <arrayName>[];`

Examples:

```
int ages[];          //Declares an array of type integers
Date dob[];          //Array of objects of class Date
```

Way 2: `<type> []<arrayName>;`

Examples:

```
Float[] marks;       //Declares an array of floats
Student[] name;      //Array of objects of students
```

Declarations of 1D arrays: Syntax

Once you have declared an array variable, you can **decide its size and allocate the memory** to store the elements. Following is the syntax to be followed:

```
<arrayName> = new <type> [<size>];
```

Examples:

```
ages = new int[10];          //Defines an array of size 10
name = new Student[100];
```

Alternatively, you can declare and define an array together.

```
<type> <arrayName> [ ] = new <type> [<size>];
```

Example:

```
int x[] = new int[100];
```

Initializations of 1D Arrays

Initialization of 1D arrays: Syntax

You can load the declared array with some elements. There are two ways to do this.

Way 1:

```
<arrayName> [<subscript> ] = <value>;
```

Example:

```
x[0] = 55;
x[1] = 99;
x[2] = 66;
```

Way 2:

```
<type> <arrayName> [ ] = { <list of values> };
```

Example:

```
int x [ ] = {12, 3, 9, 15};
```

Note: Here, declaration, definition and initialization all are at one go!

Initialization of 1D arrays: Syntax

- If you declare a size which may not be the same as the elements you have listed during initialization, then it will discard the extras or initialized with default value whatever be the case may be.

Example:

```
int x[] = new int [5];
 x = {1, 2, 3, 4, 5, 6};  // The last element will be discarded
```

Or

```
x = {1, 2, 3};     //  The last two elements will be set as 0s.
```

Example 12.1: Demonstration of a 1D Array

```
/* Demonstration of a one-dimensional array */
class Array {
     public static void main(String args[]) {
          int month[];
          month = new int[12];
          month[0] = 31;
          month[1] = 28;
          month[2] = 31;
          month[3] = 30;
          month[4] = 31;
          month[5] = 30;
          month[6] = 31;
          month[7] = 31;
          month[8] = 30;
          month[9] = 31;
          month[10] = 30;
          month[11] = 31;
          System.out.println("April has " + month[3] + " days.");
     }
}
```

Declarations of Multidimensional Arrays

Declarations of 2D Arrays

Declarations of 2D arrays: Syntax

- You can create a 2D array using any one of the following:

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

Or

```
int row, column;
. . .
int twoDarray [ ][ ] = new int [row][column];
```

Declarations of 3D arrays: Syntax

- You can create a 3D array using any one of the following:

```
int myArray [][][];
myArray = new int [3][4][5];
```

Or

```
int myArray = new int [3][4][5];
```

Or

```
int row = 3, column = 4, width = 5;
int threeDArray[][] = new int [row] [column][width];
```

Initialization of Multidimensional Arrays

Initializations of 2D Arrays

Initialization of 2D arrays

- You can follow anyone of the following method, for an example to initialize a 2D array of size 2×3.

```
int myArray [2] [3] = {1, 2, 3, 4, 5, 6};
```

Or

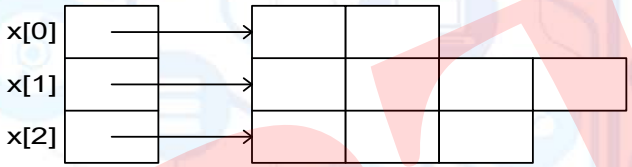
```
int myArray [ ] [ ] = { {1, 2, 3}, {4, 5, 6} };
```

Here, the 2D array will look like

```
1 2 3
4 5 6
```

Variable length 2D arrays

- You can create a variable length multidimensional array, for example, the 1st row with two columns, the 2nd row with four columns and the 3rd row with 3 columns.



- Following is the simple way to do it.

```
int x [ ] [ ] = new int [3][ ];
x[0] = new int [2];
x[1] = new int [4];
x[2] = new int [3];
```

Then you can initialize each array, the way you have done it for one-dimensional array.

An Example

Example 12.2: A general example

```
/* The following program shows the different ways that an array can be
declared and initialized. */

public class ArrayDefinitionDemo {
     public static void main(String[] args) {
          float x[];
          x = new float[100];
          args = new String[10];
          boolean[] isPrime = new boolean[1000];
          int fib[] = {0, 1, 1, 2, 3, 5, 8, 13};
          short[][][] b = new short[4][10][5];
          double a[][] = {{1.1,2.2}, {3.3,4.4}, null, {5.5,6.6}, null};
          a[4] = new double[66];
          a[4][65] = 3.14;
          Object[] objects = {x, args, isPrime, fib, b, a};
     }
}
```

Operations on 1D Arrays

Creating a 1D Array

Example 12.3: Initialization of a 1D array with user input from keyboard

```
class ArrayInitializationDemo {
     int a = new int [100];   // Declaring a 1D array of size 100
     int size;                // Actual size of the array
     void loadArray() {       // Actual size of the array
         Scanner in = new Scanner(System.in);   // To read from keyboard
         System.out.println(“Enter size < 100”);
                  size = in.nextInt();  // read a number
            for(int i = 0;i<size;i++){
               System.out.println("Enter "+ (i+1) + "th number ");
               a[i] = in.nextInt(); // Load the i-th entry
            }
        System.out.println(“Length = “ + a.length);
        System.out.println(a);      // Display the array on the screen
    }
}
```

Example 12.4: Initialization of a 1D array with random numbers

```
/* This program will initialize an array of a given size within a range between MIN and Max.*/

import java.util.*;

class ArrayInitializationRandomDemo {
     int a = new int [100];          // Declaring a 1D array of size 100
     int size;                       // Actual size of the array
     int MIN = 1, MAX = 100;         // Range of the numbers
     int generateRandom(int min, int max) {       // Generating a random number
         int rand = (int)(Math.random() * ((max - min) + 1)) + min;
         return rand;
     }
     public static void main (String args[] {
         Scanner in = new Scanner(System.in);   // To read the size from the user
         System.out.println("Enter size < 100");
         size = in.nextInt();  // read a number from the keyboard
            for(int i = 0;i<size;i++){
               a[i] = generateRandom(MIN, MAX); // Generate and return a random number
            }
       System.out.println("Capacity = " + a.length + "Size: = "+ size);
       System.out.println(a);          // Display the array on the screen
   }
}
```

Traversing a 1D Array

Example 12.5: Printing and duplicating a 1D array

```
/* This program will print on the screen using a declared print method and duplicate an array. */

public class PrintingArrays {
     // Defining a static method to print an array of integer type
     public static void print(int[] a) {
          System.out.printf("{%d", a[0]);
          for (int i = 1; i < a.length; i++) {
              System.out.printf(", %d", a[i]);
          }
          System.out.println("}");
     }

     // The following is an overloaded method to print data of any type
     public static void print(Object[] a) {
          System.out.printf("{%s", a[0]);
          for (int i = 1; i < a.length; i++) {
               System.out.printf(", %s", a[i]);
          }
          System.out.println("}");
     }
```

// Continued to next page ...

Example 12.5: Printing and duplicating a 1D array

```
// Continued on ...

    // Defining the main method
    public static void main(String[] args) {
         int[] a = {22, 44, 66, 88};     // Create an array of integers
         print(a);

         int[] b = (int[])a.clone();    // duplicate a[] in b[]
         print(b);

         String[] c = {"AB", "CD", "EF"};  // Create an array of strings
         print(c);

         String[] d = (String[])c.clone(); // duplicate c[] in d[]
         print(d);

         c[1] = "XYZ"; // change c[], but not d[]
         print(c);
         print(d);
    }
}
```

Example 12.6: Reversing a 1D array using a recursive method

```
/* This program shows how an existing array can be reversed its ordering: A genetic
programming using recursion is illustrated. */

public class Generic1DArray <T> {
       private T x; // x is a collection of generic type T
       // Constructor
       public Generic1DArray(T t) {
            x = t;
       }
       // A recursive method to reverse the ordering of array x
       public void reverse1D(int length) {
            if(length > 1) {  // Termination condition check
               //  Swap the end elements
               T temp;
               temp=x[size-length];
               x[size-length]=x[length-1];
               x[length-1]=temp;
               reverse1D(length-1); // Do recursively for the next level
       }
  }  // This completes the definition of the generic class GeneriClass<T>

                                                          // Continued to next ...
```

Example 12.6: Reversing a 1D array using a recursive method

```
// Continued on ...

/* The main program is given below. */

public class Generic1DArrayRecursionDemo {
    public static void main (String args []) {
        // Case 1: Working with integer array
             Generic1Darray<Integer> intA = new Generic1Darray<Integer> (1, 2, 3, 4, 5, 6);
            intA.reverse1Darray(6);    // Reversing the array intA
            System.out.println(intA);   // Print the result

             // Case 2: Working with String array
             Generic1Darray<String> stringA = new Generic1Darray<String> ("A", "E", "I", "O", "U");
            stringA.reverse1Darray(5);    // Reversing the array stringA
            System.out.println(stringA);   // Print the result
    }
}  // This completes the definition of the main class
```

Insertion to a 1D Array

Example 12.7: Insertion into a 1D array

```
/* This program shows how an item can be inserted into an existing array. */
import java.util.*;

public class Insertion1DArray {
    int a[100];    // An integer array with default 100 capacity
    int size=0;    // The number of items present
        public void static create(int size) {// Initialize the array
            if(size > 100) {
                System.out.println(“Could not initialize…”);
                return 0;
            }
            this.size = size;
            Scanner in = new Scanner(System.in);   // To read from keyboard
            for(int i = 0;i<size;i++){ // Reading the number
                System.out.println("Enter "+ (i+1) + "th number ");
                a[i] = in.nextInt(); // Load the i-th entry
            }
            System.out.println(“Capacity = “ + a.length + “Size + “ + size);
            System.out.println(a);      // Display the array on the screen
        }

                                                    // Continued to next ...
```

Example 12.7: Insertion into a 1D array

```
// Continued on ...

    public void static insert(int loc) {// Insertion at loc of the array
        if(size == a.length) {
            System.out.println("Array is overflow: Insertion fails…");
            return 0;
        }
        if ((loc < 0) || (loc > size)){
            System.out.println("Out of range: Insertion fails…");
            return 0;
        }
        Scanner in = new Scanner(System.in);   // To read from keyboard
        int item = in.nextInt();
        for(int i = size;i< loc;i--)          // Shifting operation
            a[i+1] = a[i];                   // Shift to right one place
        a[loc] = item;
        System.out.println("Capacity = " + a.length + "Size + " + size);
        System.out.println(a);      // Display the array on the screen
    }
                                                    // Continued to next ...
```

Example 12.7: Insertion into a 1D array

```
// Continued on ...

    /* The main method ...
    public void static main (String args[]){
       Scanner in = new Scanner(System.in);   // To read from keyboard
       System.out.println("Enter the number of elements : ");
       int n = in.nextInt();
       create(n);

       System.out.println("Enter the new number to be inserted: ");
       n = in.nextInt();
       System.out.println("At position? ");
       int pos = in.nextInt();
       insert(n, pos);

       System.out.println("Capacity = " + a.length + "Size + " + size);
       System.out.println(a);      // Display the array on the screen
    }
}
```

Deletion from a 1D Array

Example 12.8: Deletion from a 1D array

```
/* This program shows how an item can be deleted from an existing array. */
import java.util.*;

public class Deletion1DArray {
    int a[100];    // An integer array with default 100 capacity
    int size=0;    // The number of items present
        public void static create(int size) {// Initialize the array
            if(size > 100) {
                System.out.println(“Could not initialize…”);
                return 0;
            }
            this.size = size;
            Scanner in = new Scanner(System.in);   // To read from keyboard
            for(int i = 0;i<size;i++){ // Reading the number
                System.out.println("Enter "+ (i+1) + "th number ");
                a[i] = in.nextInt(); // Load the i-th entry
            }
            System.out.println(“Capacity = “ + a.length + “Size + “ + size);
            System.out.println(a);      // Display the array on the screen
        }

                                                    // Continued to next ...
```

Example 12.8: Deletion from a 1D array

```
// Continued on ...

    public void static delete(int loc) {// Deletion of the item at loc position
        if(size == 0) {
            System.out.println("Array is underflow: No item to delete!");
            return 0;
        }
        if ((loc < 0) || (loc > size)){
            System.out.println("Out of range: Deletion fails ...");
            return 0;
        }
        int item = a[loc];              // The item at loc
        for(int i = loc;i < size;i++)   // Shifting operation
            a[i] = a[i+1];                  // Shift to left one place
        a[size] = NULL;
        size--;
        System.out.println("Capacity = " + a.length + "Size + " + size);
        System.out.println(a);      // Display the array on the screen
    }
                                                        // Continued to next ...
```

Example 12.8: Deletion from a 1D array

```
// Continued on ...

    /* The main method ...*/
    public void static main (String args[]){
       Scanner in = new Scanner(System.in);   // To read from keyboard
       System.out.println("Enter the number of elements : ");
       int n = in.nextInt();
       create(n);

       System.out.println("Enter the position from number to be deleted: ");
       int pos = in.nextInt();
       int item = delete(pos);
       System.out.println("Deleted item: " + item);

       System.out.println("Capacity = " + a.length + "Size + " + size);
       System.out.println(a);      // Display the array on the screen
    }
}
```

Matrix operations

Example 12.9: Creating a 2D array

```
/* This program shows how a 2D array of integers can be created, initialized and
displayed both in row-major and column major orders. */

import java.util.*;

public class TwoDarray {
    int a[][];      // Declared that a is an array of two-dimensional
    int row;        // The number of rows
    int col;        // The number of columns

    TwoDarray(int row, int col) {
          a = new int[row[col];
          this.row = row; this.col = col;
    }

                                                    // Continued to next ...
```

Example 12.9: Creating a 2D array

```
// Continued on ...

  public void initialize() {
      Scanner in = new Scanner(System.in);
          for(int i = 0; i < row, i++){
              for(int j = 0; j < col,; j==) {
                  System.out.println("Enter a[%d][%d] :" + i + j);
                      a[i][j] = in.nextInt();
              }
          }
    }
                                                    // Continued to next ...
```

Example 12.9: Creating a 2D array

```
// Continued on ...

    // Display the matrix in Row-Major order
    public void printRowMajor() {
        for(int i=0;i<row;i++){
            for(int j=0;j<col;j++){
                System.out.print(a[i][j]);
            }
            System.out.println(" ");
        }
     }
    // Display the matrix in Column-Major order
    public void printColMajor() {
        for(int i=0;i<col;i++){
            for(int j=0;j<row;j++){
                System.out.print(a[j][i]);
            }
            System.out.println(" ");
        }
    }
    // Here add the methods for matrix manipulation : Code is in the next slide

                                                    // Continued to next ...
```

Example 12.9: Defining a method for 2D matrix addition

```
// Continued on ...

  public int[][] addition(int b[][]) {
    // First check if the two matrices are conformable for addition
    if(this.row != b.row) && (this.col != b.col) {
       System.out.println("Error! Matrix are not of proper size.");
       return (0);
    else {
       twoDarray c = new twoDarray[row][col];    // Declare a the output matrix
       for(int i = 0; i < row; i++)
          for(int j = 0; j < col; j++)
          c[ij[j] = this.[i][j] + b[i][j];
       return(c);
    }
  }

                                                  // Continued to next ...
```

Example 12.9: Defining a method for 2D matrix multiplication

```
// Continued on ...

  public int[][] multiplication(int b[][]) {
    // First check if the two matrices are conformable for multiplication
    if(this.col != b.row) {
       System.out.println("Error! Matrices are not of proper size.");
       return (0);
    else {
       twoDarray c = new twoDarray[row][b.col];    // Declare the output matrix
       for(int i = 0; i < row; i++)
           for(int j = 0; j < b.col; j++) {
              c[i][j] = 0;
              for(int k = 0; k < b.row; k++){
                   c[ij[j] = c[i][j] + this.[i][k] * b[k][j];
           }
       return(c);
    }
  }
} // End of the twoDarray class definition

                                                    // Continued to next ...
```

Example 12.9: 2D matrix operations

```
// Continued on ...

     /* The main class is given below.*/
class twoDarrayAdditionDemo {
     public static void main {String args [] {
          twoDarray x = new TwoDarry(3,4);        // Create a 2D array of 3x4
          x.initialize();                         // Initialize the array
          twoDarray y = new TwoDarry(3,4);        // Create a 2D array of 3x4
          y.initialize();                         // Initialize the array

          twoDArray z[][];
          z = x.addition(y);                      // Do addition of two matrix
          z.printRowMajor();                      // Print the array in row-major order

          z = x.multiplication(y);                // Do multiplication of two matrix
          z.printRowMajor();                      // Print the array in row-major order

     }
}
```

REFERENCES

- Classic Data Structures, Debasis Samanta, 2nd Edition, Prentice Hall of India
- https://cse.iitkgp.ac.in/~dsamanta/javads/index.html

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

योगः कर्मसु कौशलम्

THANK YOU !

NPTEL ONLINE CERTIFICATION COURSES

Data Structures and Algorithms Using Java

Debasis Samanta

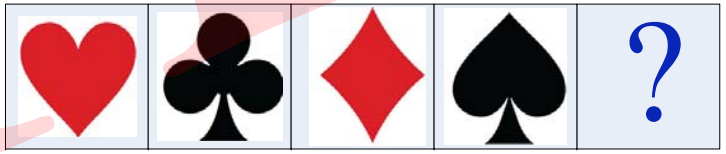
Department of Computer Science & Engineering, IIT Kharagpur

Module 04: Arrays

Lecture 13 : Class ArrayList for Arrays

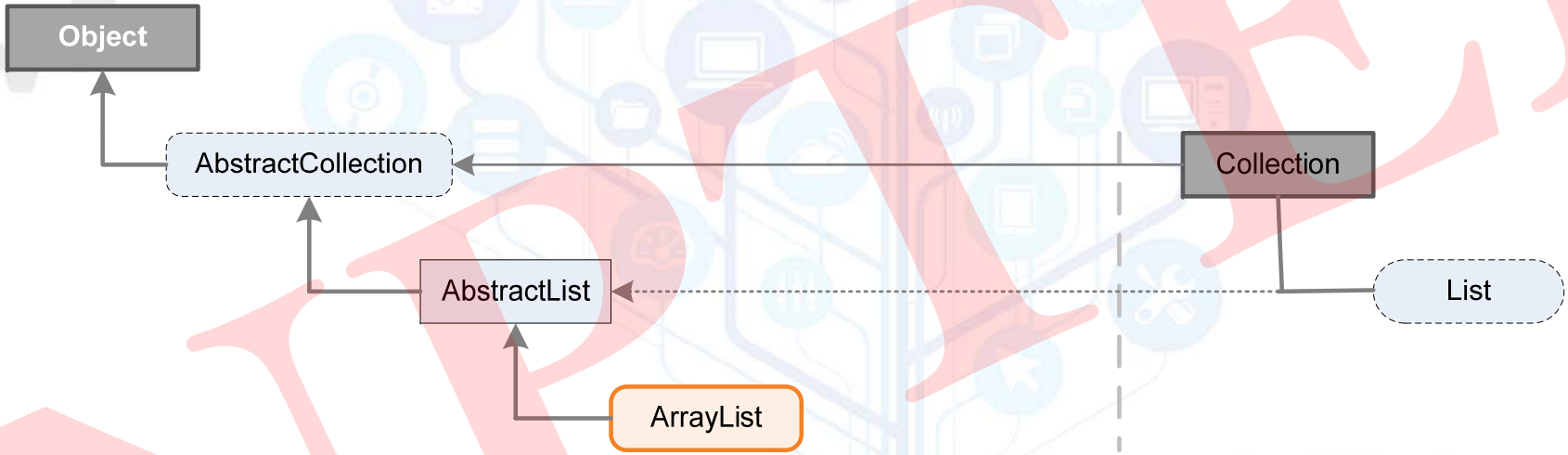
CONCEPTS COVERED

- About ArrayList Class
  - Concept
  - Constructors
  - Methods
- Creating a collection
- Operations
  - Insertion
  - Deletion
  - Traversal
  - Searching
  - Sorting



ArrayList Class

Class ArrayList

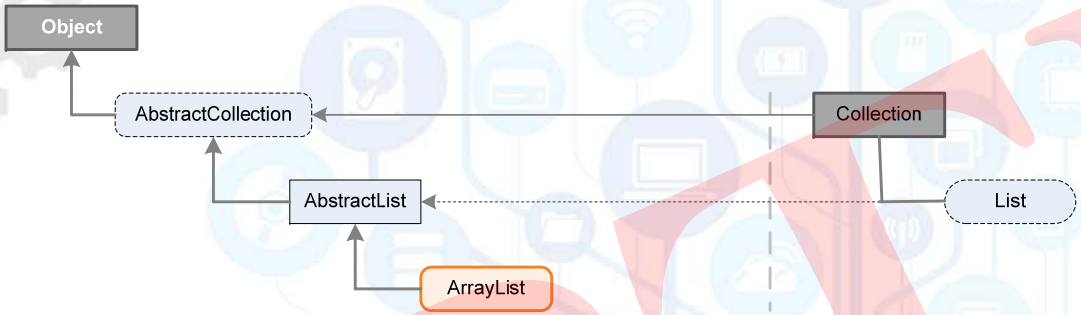


Characteristics of `ArrayList` class

The important points about the `ArrayList` class:

1. `ArrayList` class allows a collection to have duplicate elements.
2. `ArrayList` class maintains insertion order.
3. A collection according to `ArrayList` class is non-synchronized.
4. `ArrayList` allows random access because array works on the index basis.
5. `ArrayList` supports dynamic arrays that can grow as needed.
6. Using `ArrayList` class, manipulation is slow, because a lot of shifting needs to occur if any element is removed from the collection.

Composition of `ArrayList` class



- The `ArrayList` class extends `AbstractList` and implements the `List` interface.
- `ArrayList` is a generic class that has this declaration:

```
class ArrayList<T> { … }
```

Here, T specifies the type of object that the list will hold.

Creating a Collection

Constructor of `ArrayList` class

Constructor	Description
`ArrayList()`	It is used to build an empty array list.
`ArrayList(Collection<? extends T> c)`	It is used to build an array list that is initialized with the elements of the collection c.
`ArrayList(int capacity)`	It is used to build an array list that has the specified initial capacity.

Example 13.1: Creating a simple collection

```java
/* The following program illustrate the use of the simple constructor to declare a collection. */

import java.util.ArrayList;


public class SimpleArrayListExample {
    public static void main(String[] args)       {
          // Creating an ArrayList of String
          ArrayList<String> animals = new ArrayList<String>();
          // Adding new elements to the ArrayList
          animals.add("Lion");
          animals.add("Tiger");
          animals.add("Cat");
          animals.add("Dog");
          // animals.add(2019);  Is it valid?
          // This shows how an entire collection can be displayed
          System.out.println(animals);
    }
}
```

Note:

- An `ArrayList` object is a collection of homogeneous objects.
- Adding an object of another type invites compile-time error or run-time exception.

Example 13.2: Creating an `ArrayList` object with an existing collection

```
// Creating an ArrayList from another collection using the ArrayList(Collection c) constructor.

import java.util.*;
public class CreateArrayListFromCollectionExample {
    public static void main(String[] args)       {
          // Creating a collection first. Let it be with the simple method
          ArrayList<Integer> aList = new ArrayList<Integer>(); //Declaring aList as a collection
          aList.add(2); // Adding elements into the aList collection
          aList.add(3);
          aList.add(5);
          aList.add(7);
          aList.add(11);
          // Creating another collection initially with aList collection
          ArrayList<Integer> numbers = new ArrayList<Integer>(aList);
          numbers.add(13); // Add two more numbers into the numbers collection
          numbers.add(17);
          System.out.println(aList); // Now, yu have two collections: aList and numbers.
          System.out.println(numbers); // Printing the two collections

    }
}
```

Note:

- You cannot add a collection of another type to a collection of other type.
- If you create a collection of type Object, then in that collection you can add object of any type.

Example 13.3: Creating an `ArrayList` object with user defined objects

- Since `ArrayList` supports generics, you can create an ArrayList of any type. It can be of simple types like `Integer`, `String`, `Double` or complex types like an `ArrayList` of `List`, `HashMaps`, or of any user defined objects.

```
/* This program illustrates the creation of an ArrayList collection of user
defined type. */

public class Person {
    private String name;
    private int age;
    public Person(String name, int age)    {
        this.name = name;
        this.age = age;
    }
    public void printData() {
        System.out.println(name + " " + age);
    }
}
```

// Continued to next ...

Example 13.3: Creating an `ArrayList` object with user defined objects

```
// Continued on ...

import java.util.ArrayList;
public class ArrayListUserDefinedObjectDemo{
   public static void main(String[] args) {
    // Declaring pList as a collection of type Person of capacity 5
    ArrayList<Person> pList = new ArrayList<Person>(5);
    pList.add(new Person("Ram", 25));
    Person p2 = new Person("Sita", 22);   // Create a new object
    pList.add(p2);                        // add the object
    pList.add(new Person("John", 34));
    pList.add(p2);                        // Duplicate entry is allowed
    pList.add(new Person("Rahim", 29));   // Five objects are added
    pList.add(new Person("Lilly", 24));
    // No issue to accommodate, list grows dynamically
    pList.forEach(p -> p.printData());
    // An way to access each object in a class
  }
}
```

Insertion into a Collection

Methods of `ArrayList` class

Method	Description
`void add(int index, E element)`	It is used to insert the specified element at the specified position in a list.
`boolean add(E e)`	It is used to append the specified element at the end of a list.
`boolean addAll(Collection<? extends T> c)`	It is used to append all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator.

Example 13.4: Insertion into an `ArrayList` collection

You have noticed how an object can be added to a collection using `add()` method of the `ArrayList` class. There are another methods for the similar tasks, namely `addAll()` and its variations. The smple `add()` method allows to one object, whereas `addAll()` allows to insert a list of objects.

```
// This program illustrates the insertion operation into an ArrayList collection.

import java.util.ArrayList;
public class InsertionArrayListDemo {
     public static void main(String[] args)        {
         // Creating a collection first. Let it be with the simple method
         ArrayList<Integer> odd = new ArrayList<>();  // Declaring aList as a
collection
         // Adding elements into the odd collection
         odd.add(1);
         odd.add(3);
         odd.add(5);
         odd.add(7);
         odd.add(9);
         System.out.println(odd);

                                                    // Continued to next ...
```

Example 13.4: Insertion into an `ArrayList` collection

```
// Continued on ...

        // Creating another collection, say number with elements from odd collection
        ArrayList<Integer> numbers = new ArrayList<Integer>(odd);
        System.out.println(numbers);    // same as odd

        // Creating another collection, say even1
        ArrayList<Integer> even1 = new ArrayList<Integer>();
        // Add numbers into the even1 collection
        even1.add(2);
        even1.add(4);
        even1.add(6);

        // Insert all the elements of even1 collection at the end of number collection
        numbers.addAll(even1);
        System.out.println(numbers);
                                                        // Continued to next ...
```

Example 13.4: Insertion into an `ArrayList` collection

```
// Continued on ...

        // Creating another collection, say any
        ArrayList<Integer> any = new ArrayList<Integer>();
        // Add numbers into “any” collection
        any.add(8);
        any.add(11);
        any.add(13);

        // Add the collection any at 5-th location of the collection numbers
        numbers.addAll(5, any);
        //add an object at a specific location of the colletion numbers
        numbers.add(0,0);
        System.out.println(numbers);
        // What will happen to the following?
        //numbers.add(100,999);  ??
    }
}
```

Accessing Objects in a Collection

Methods of `ArrayList` class

Method	Description
`Object clone()`	It is used to return a shallow copy of an ArrayList.
`boolean contains(Object o)`	It returns true if the list contains the specified element
`int indexOf(Object o)`	It is used to return the index in this list of the first occurrence of the specified element, or -1 if the List does not contain this element.
`E set(int index, E element)`	It is used to replace the specified element in the list, present at the specified position.
`int size()`	It is used to return the number of elements present in the list.
`void trimToSize()`	It is used to trim the capacity of this ArrayList instance to be the list's current size.

Example 13.5: Accessing objects in an `ArrayList` collection

This example shows:

- How to check if an `ArrayList` is empty using the method `isEmpty()`.
- How to find the size of an `ArrayList` using the `size()` method.
- How to access the element at a particular index in an `ArrayList` using the `get()` method.
- How to modify the element at a particular index in an `ArrayList` using the `set()` method.

```
import java.util.ArrayList;
public class AccessingArrayListObjects{
     public static void main(String[] args)         {
     ArrayList<String> topCompanies = new ArrayList<String>();
     // Check if an ArrayList is empty
     System.out.println("Is the topCompanies list empty? :  " + topCompanies.isEmpty());
     topCompanies.add("Google");
     topCompanies.add("Apple");
     topCompanies.add("Microsoft");
     topCompanies.add("Amazon");
     topCompanies.add("Facebook");
                                                    // Continued to next ...
```

Example 13.5: Accessing objects in an `ArrayList` collection

```
// Continued on...

        // Find the size of an ArrayList
        System.out.println("Here are the top " + topCompanies.size() + “Companies in the world");
        System.out.println(topCompanies);   // Print the companies names
        // Retrieve the element at a given index
        String bestCompany = topCompanies.get(0);
        System.out.println("Best Company: " + bestCompany);
        String secondBestCompany = topCompanies.get(1);
        System.out.println("Second Best Company: " + secondBestCompany);
        String lastCompany = topCompanies.get(topCompanies.size() - 1);
        System.out.println("Last Company in the list: " + lastCompany);
        // Modify the element at a given index
        topCompanies.set(4, "Walmart");
        System.out.println("Modified top companies list: " + topCompanies);
    }
}
```

Deletion from a Collection

Methods of `ArrayList` class

Method	Description
`void clear()`	It is used to remove all of the elements from this list.
`E remove(int index)`	It is used to remove the element present at the specified position in the list.
`boolean remove(Object o)`	It is used to remove the first occurrence of the specified element.
`boolean removeAll(Collection<?> c)`	It is used to remove all the elements from the list.
`boolean removeIf(Predicate<? super E> filter)`	It is used to remove all the elements from the list that satisfies the given predicate.
`protected void removeRange(int fromIndex, int toIndex)`	It is used to remove all the elements lies within the given range.
`void replaceAll(UnaryOperator<E> operator)`	It is used to replace all the elements from the list with the specified element.
`void retainAll(Collection<?> c)`	It is used to retain all the elements in the list that are present in the specified collection.

Example 13.6: Deletion from an `ArrayList` collection

*The class **ArrayList** includes many methods like remove, removeAll(), clear(), etc. to remove objects from a collection.*

The next program shows:

- How to remove the element at a given index in an ArrayList | remove(int index)
- How to remove an element from an ArrayList | remove(Object o)
- How to remove all the elements from an ArrayList that exist in a given collection | removeAll()
- How to remove all the elements matching a given predicate | removeIf()
- How to clear an ArrayList | clear()

Example 13.6: Deletion from an `ArrayList` collection

```
import java.util.ArrayList;
import java.util.function.Predicate;

public class DeletionArrayListDemo        {
    public static void main(String[] args) {
        // Create a collection. Initially empty
         ArrayList<String> langs = new ArrayList<String>();

        // Add elements into the collection
        langs.add("C");
        langs.add("C++");
        langs.add("Java");
        langs.add("Python");
        langs.add("R");
        langs.add("Spark");
        System.out.println("Initial List: " + langs);
                                                    // Continued to next ...
```

Example 13.6: Deletion from an `ArrayList` collection

```
// Continued on ...

        // Removing elements from the collection
        langs.remove(5);  // Remove the element at index `5`
        System.out.println("After remove(5): " + langs);
        // Remove the first occurrence of the given element from the ArrayList
        boolean status = langs.remove("Smalltalk");
        System.out.println("Smalltalk is removed : " + status);
        // Remove all the elements that exist in a given collection
        ArrayList<String> script = new ArrayList<String>();
        script.add("SQL");
        script.add("Python");
        script.add("Javascript");
        langs.removeAll(script);    // Remove intersection of langs and script
        System.out.println("After script removal: " + langs);

                                                    // Continued to next ...
```

Example 13.6: Deletion from an `ArrayList` collection

```
// Continued on ...

        // Remove all the elements that satisfy the given predicate
        langs.removeIf(new Predicate<String>() {@Override
            public boolean test(String s) {
                return s.startsWith("C");});

        /* The above removeIf() call can also be written using lambda expression like this :
        langs.removeIf(s -> s.startsWith("C"));
        */
        System.out.println("After Removing all elements that start with \"C\": " + langs);
        // Remove all elements from the ArrayList
        langs.clear();
        System.out.println("List is empty? " + langs.isEmpty());
    }
}
```

Searching a Collection

Methods of `ArrayList` class

Method	Description
`T get(int index)`	It is used to fetch the element from the particular position of the list.
`int indexOf(Object o)`	It is used to return the index in this list of the first occurrence of the specified element, or -1 if the List does not contain this element.
`int lastIndexOf(Object o)`	It is used to return the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element.
`E set(int index, E element)`	It is used to replace the specified element in the list, present at the specified position.

Example 13.7: Searching an `ArrayList` collection

The next program shows how to:

- Check if an `ArrayList` contains a given element | `contains()`
- Find the index of the first occurrence of an element in an `ArrayList` | `indexOf()`
- Find the index of the last occurrence of an element in an `ArrayList` | `lastIndexOf()`

Example 13.7: Searching an ArrayList collection

```
import java.util.ArrayList;

public class SearchElementsInArrayListExample       {
     public static void main(String[] args)         {
         ArrayList<String> names = new ArrayList<String>();
         names.add("John");
         names.add("Alice");
         names.add("Bob");
         names.add("Steve");
         names.add("John");
         names.add("Steve");
         names.add("Maria");
         // Check if an ArrayList contains a given element
         System.out.println("Bob exist? : " + names.contains("Bob"));
         // Find the index of the first occurrence of an element in an ArrayList
         System.out.println("indexOf \"Steve\": " + names.indexOf("Steve"));
         System.out.println("indexOf \"Mark\": " + names.indexOf("Mark"));
         // Find the index of the last occurrence of an element in an ArrayList
         System.out.println("lastIndexOf John : " + names.lastIndexOf("John"));
         System.out.println("lastIndexOf Bill: " + names.lastIndexOf("Bill"));
     }
}
```

Sorting a Collection

Methods of `ArrayList` class

Method	Description
`void sort(Comparator<? super E> c)`	It is used to sort the elements of the list on the basis of specified comparator.

Example 13.7: Sorting an `ArrayList` collection

- Sorting an `ArrayList` is a very common task that you will encounter in your programs. In this section, I'll show you how to :
  - Sort an `ArrayList` using `Collections.sort()` method.
  - Sort an `ArrayList` using `ArrayList.sort()` method.
  - Sort an `ArrayList` of user defined objects with a custom comparator.

Sorting using Collection.sort()

Example 13.8: Sorting an `ArrayList` collection

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class ArrayListCollectionsSortExample {
    public static void main(String[] args) {
        List<Integer> numbers = new ArrayList<Integer>();
        numbers.add(13);
        numbers.add(7);
        numbers.add(18);
        numbers.add(5);
        numbers.add(2);

        System.out.println("Before : " + numbers);

        // Sorting an ArrayList using Collections.sort() method
        Collections.sort(numbers);

        System.out.println("After : " + numbers);
    }
}
```

Sorting using ArrayList.sort()

Example 13.9: Sorting an `ArrayList` collection

```
import java.util.*;

public class ArrayListSortExample {
    public static void main(String[] args) {
        List<String> names = new ArrayList<>();
        names.add("Lisa");
        names.add("Preeti");
        names.add("Jay");
        names.add("Soma");

        System.out.println("Names : " + names);

        // Sort an ArrayList using its sort() method.
        // You must pass a Comparator to the ArrayList.sort() method.
        names.sort(new Comparator<String>() {
            @Override
            public int compare(String name1, String name2) {
                return name1.compareTo(name2);
            }
        });
        System.out.println("Sorted Names : " + names);
    }
}
```

Sorting using Custom Comparator

Example 13.10: Sorting an `ArrayList` collection

```
import java.util.*;

class Person {
    private String name;
    private Integer age;

    public Person(String name, Integer age){
        this.name = name;
        this.age = age;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
                        // Continued to next ...
```

```
public Integer getAge() {
        return age;
    }
    public void setAge(Integer age) {
        this.age = age;
    }
    @Override
    public String toString() {
        return "{" +
                "name='" + name + '\'' +
                ", age=" + age +
                '}';
    }
}
                                // Continued to next ...
```

Example 13.10: Sorting an `ArrayList` collection

```
// Continued on ...

public class ArrayListObjectSortExample {
    public static void main(String[] args) {
        List<Person> people = new ArrayList<>();
        people.add(new Person("Sachin", 47));
        people.add(new Person("Chris", 34));
        people.add(new Person("Rajeev", 25));
        people.add(new Person("David", 31));

        System.out.println("Person List : " + people);

        // Sort People by their Age
        people.sort((person1, person2) -> {
            return person1.getAge() - person2.getAge();
        });

        // A more concise way of writing the above sorting function
        people.sort(Comparator.comparingInt(Person::getAge));

        System.out.println("Sorted Person List by Age : " + people);

        // Sort using Collections.sort() method by passing the custom Comparator
        Collections.sort(people, Comparator.comparing(Person::getName));
        System.out.println("Sorted Person List by Name : " + people);
    }
}
```

Traversing a Collection

Example 13.11: Traversing an `ArrayList` collection

The next few example shows how to iterate over an `ArrayList` using

- forEach and lambda expression.
- iterator()
- forEachRemaining() method.
- listIterator().
- Simple for-each loop.
- for loop with index.

Example 13.11: Traversing an `ArrayList` collection

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.ListIterator;

public class IterateOverArrayListDemo {
     public static void main(String[] args)        {
         ArrayList<String> tvShows = new ArrayList<String>();
         tvShows.add("Nimki Mukhiya");
         tvShows.add("Game of Thrones");
         tvShows.add("Mahabharat");
         tvShows.add("Balika Badhu");

         System.out.println("Traversing using forEach()  and lambda \n");
         tvShows.forEach(tvShow -> { System.out.println(tvShow); });

                                                    // Continued to next page ...
```

Example 13.11: Traversing an `ArrayList` collection

```
// Continued on ...

        System.out.println("\n=== Iterate using an iterator() ===");
        Iterator<String> tvShowIterator = tvShows.iterator();
        while (tvShowIterator.hasNext()) {
             String tvShow = tvShowIterator.next();
             System.out.println(tvShow);
        }

        System.out.println("Traversing usingiterator() and forEachRemaining()");
        tvShowIterator = tvShows.iterator();
        tvShowIterator.forEachRemaining(tvShow -> {
                System.out.println(tvShow);
                });

                                                    // Continued to next page ...
```

Example 13.11: Traversing an `ArrayList` collection

```
// Continued on ...

        System.out.println("Traversing using a listIterator() \n");
        // Here, we start from the end of the list and traverse backwards.
        ListIterator<String> tvShowListIterator = tvShows.listIterator(tvShows.size());
        while (tvShowListIterator.hasPrevious()){
            String tvShow = tvShowListIterator.previous();
            System.out.println(tvShow);
        }

        System.out.println("\n=== Iterate using simple for-each loop ===");
        for(String tvShow: tvShows)       {
            System.out.println(tvShow);
        }

        System.out.println("\n=== Iterate using for loop with index ===");
        for(int i = 0; i < tvShows.size(); i++){
            System.out.println(tvShows.get(i));
        }
    }
}
```

Example 13.12: Traversing an `ArrayList` collection

- The `iterator()` and `listIterator()` methods are useful when you need to modify the `ArrayList` while traversing.
- Consider the next example, where we remove elements from the `ArrayList` using `iterator.remove()` method while traversing through it.

Example 13.12: Traversing an `ArrayList` collection

```
import java.util.*;
public class ArrayListIteratorRemoveExample {
    public static void main(String[] args) {
        List<Integer> numbers = new ArrayList<Integer>();
        numbers.add(13);
        numbers.add(18);
        numbers.add(25);
        numbers.add(40);

        Iterator<Integer> numbersIterator = numbers.iterator();
        while (numbersIterator.hasNext()) {
            Integer num = numbersIterator.next();
            if(num % 2 != 0) {
                numbersIterator.remove();
            }
        }
        System.out.println(numbers);
    }
}
```

Bulk Operation

Bulk operations on `ArrayList` collection

- When working with `ArrayList`, you will sometimes want to obtain an actual array that contains the contents of the list. You can do this by calling `toArray( )`, which is defined by `Collection`. Several reasons exist why you might want to convert a collection into an array, such as:
    - To obtain faster processing times for certain operations
    - To pass an array to a method that is not overloaded to accept a collection
    - To integrate collection-based code with legacy code that does not understand collections

Methods of `ArrayList` class

Method	Description
`Object[] toArray()`	It is used to return an array containing all of the elements in this list in the correct order.
`<T> T[] toArray(T[] a)`	It is used to return an array containing all of the elements in this list in the correct order.
`Object clone()`	It is used to return a shallow copy of an ArrayList.
`void sort(Comparator<? super E> c)`	It is used to sort the elements of the list on the basis of specified comparator.
`Spliterator<E> spliterator()`	It is used to create spliterator over the elements in a list.
`List<E> subList(int fromIndex, int toIndex)`	It is used to fetch all the elements lies within the given range.
`void trimToSize()`	It is used to trim the capacity of this ArrayList instance to be the list's current size.

Bulk operations on `ArrayList` collection

- Whatever the reason, converting an `ArrayList` to an array is a trivial matter. As explained earlier, there are two versions of `toArray( )`, which are shown again here for your convenience:

```
Object[ ] toArray( );
   T[ ] toArray();
```

- The method should be called for the collection object. The first returns an array of `Object`. The second returns an array of elements that have the same type as T. Normally, the second form is more convenient because it returns the proper type of array.

Example 13.13: Bulk copy of an `ArrayList` collection

```
import java.util.*;
class ArrayListToArray {
    public static void main(String args[])          {
        // Create an array list.
        ArrayList<Integer> al = new ArrayList<Integer>();
        // Add elements to the array list.
        al.add(1);
        al.add(2);
        al.add(3);
        al.add(4);
        System.out.println("Contents of al: " + al);
        // Get the array.
        Integer ia[] = new Integer[al.size()];
        ia = al.toArray(ia);
        // Object[] ia = al.toArray();   Alternatively
        int sum = 0;
        // Sum the array.
        for(int i : ia)
           sum += i;
           System.out.println("Sum is: " + sum);
    }
}
```

REFERENCES

- The Complete Reference, Herbert Schildt, 9th Edition, Oracle Press
- https://cse.iitkgp.ac.in/~dsamanta/javads/index.html
- https://docs.oracle.com/javase/tutorial/

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

योगः कर्मसु कौशलम्

THANK YOU !

NPTEL ONLINE CERTIFICATION COURSES

Data Structures and Algorithms Using Java

Debasis Samanta

Department of Computer Science & Engineering, IIT Kharagpur

Module 04: Arrays

Lecture 14 : Arrays for Arrays

CONCEPTS COVERED

- About Class Arrays in JCF
- Composition of Class Arrays
  - Methods
- Working with Arrays
  - Creating
  - Searching
  - Sorting
  - Traversal

About Class Arrays

About class Arrays

About class `Arrays`

- This class name is in plural form "**Arrays**" because it can handle java arrays of different primitive types like *byte*, *char*, *short*, *int*, *long*, *float*, *double*, and *universal type* Object.
- This class `Arrays` has been designed to bridge the gap between collections and Java arrays. More specifically, there are often times when loops are used to do some tasks on an array like fill an array with a particular value, sort an array, search in an array, etc.
- This class provides several static methods to dynamically create and access Java arrays. In fact, it consists of only static methods and the methods of `Object` class.
- The methods of this class can be used by the class name itself. That is, no object of the class `Arrays` need to be created to access a method in it.

About class `Arrays`

- The methods of this class can be used by the class name itself. That is, no object of the class Arrays need to be created to access a method in it.

- The following is the syntax to access a method in Arrays.

```
Arrays.<methodName (. . .)>;
```

Here, **<T>** denotes a type parameter. This type parameter includes `boolean`, `byte`, `char`, `short`, `int`, `long`, `float`, `double`, and `Object`.

Methods of Class Arrays

Methods of class Arrays

Method	Description
`<T>ListasList(<T> array)`	It returns a List that is backed by an array of type T. Both the list and array refer to the same location.
`Int binarySearch(<T> array[ ], <T> value)`	It searches for the specified element in the array with the help of Binary Search algorithm. It returns the index of an array location, if found else NULL. Array of type boolean is not applicable to such a method.
`<T>[ ] copyOf(<T>[ ] source, int len)`	It returns a copy of an array source and len is the length of the copy. If the copy is longer than source, then the copy is padded with zeros (for numeric arrays), nulls (for object arrays), or false (for boolean arrays). If the copy is shorter than source, then the copy is truncated.

Methods of class Arrays

Method	Description
`<T>[ ] copyOfRange(<T>[ ] source, int start, int end)`	It returns a copy of a range within an array in from start to end-1.
`boolean equals(<T> array1[ ], <T> array2 [ ])`	It returns true if two arrays are equivalent, otherwise, it returnsfalse. Here, array1 and array2 should be comparable for equality.
`boolean deepEquals(Object[ ] a, Object[ ] b)`	The method can be used to determine if two arrays, which might contain nested arrays, are equal.
`void fill(<T> array[ ], <T> value)`	It fills an array with a specified value.
`void fill(<T> array[ ], int start, int end, <T> value)`	It fills an array with a specified value to a subset of an array from position start to end-1.
`void sort(byte array[ ])`	It sorts an array so that it is arranged in ascending order.
`void sort(<T> array[ ], int start, int end)`	It sorts a subarray in ascending order between the position start and end-1.
`void parallelSort(byte array[ ])`	it sorts, into ascending order, portions of an array in parallel and then merges theresults.
`void parallelSort(<T> array[ ], int start, int end)`	It sorts a subarray in ascending order between the position start and end-1 in parallel.
`<T>Spliteratorspliterator(T array[ ])`	It returns a spliterator to an entire array. Here, array is the array that the spliterator will cycle through.

Methods of class Arrays

Method	Description
`<T>Spliteratorspliterator(T array[ ], int start, int end)`	It enables you to specify a range to iterate within the array.
`<T> Stream stream(T array[ ])`	It supports a Stream interface. Here, array is the array to which
`DoubleBinaryOperatorfunc)`	cumulativeresult of an operation applied to all previous elements. For example, if the operation is addition, then on return, the array elements will contain the values associated with the running sum of the original values. Many other versions are provided, including those that operate on types int, long, and generic, and those that let you specify a range within the array on which to operate.

Note:

Arrays also provides `toString( )` and `hashCode( )` for the various types of arrays.

In addition, `deepToString( )` and `deepHashCode( )` are provided, which operate effectively on arrays that contain nested arrays.

Operation with Arrays

Example 14.1: Converting an array to a collection object

```
// This program illustrates the use of  asList() method.

import java.util.Arrays;

public class ArrayToListDemo {
    public static void main(String[] args)  {
        int intArr[] = { 10, 20, 15, 22, 35}; // An array of int
        System.out.println("Integer Array as List: "
        + Arrays.asList(intArr)); // To convert the elements as List
    }
}
```

Example 14.2: Stream representation of arrays

```
/* This method returns a sequential stream with the specified
array as its source. */

import java.util.Arrays;

public class ArraysToStreamDemo {
    public static void main(String[] args){
        // Get the Array
        int intArr[] = { 10, 20, 15, 22, 35};
        // To get the Stream from the array
        System.out.println("Integer Array: "
                           + Arrays.stream(intArr));
    }
}
```

Example 14.3: Converting array items into String objects

```
/* This method returns a String representation of the contents of
this Arrays. */

import java.util.Arrays;

public class ArraysToStringDemo {
    public static void main(String[] args){
        // Get the Array
        int intArr[] = { 10, 20, 15, 22, 35};
        // To print the elements in one line
        System.out.println("Integer Array: "
                           + Arrays.toString(intArr));
    }
}
```

Example 14.4: Converting array items into String objects

```
// This program illustrates the use of Arrays.deepToString() method

import java.util.Arrays;

public class ArraysToDeepStringDemo {
    public static void main(String[] args){
        // Get the Array
        int intArr[][] = { { 10, 20, 15, 22, 35} };
        // To get the deep String of the arrays
        System.out.println("Integer Array: "
                           + Arrays.deepToString(intArr));
    }
}
```

Example 14.5: Copying an array to collection

```
// This program illustrates the use of copyOf() method.

import java.util.Arrays;

public class CopyOfArraysDemo {
    public static void main(String[] args) {
        int intArr[] = { 10, 20, 15, 22, 35};  // An input array
        // To print the elements in one line
        System.out.println("Integer Array: ” + Arrays.toString(intArr));
        System.out.println("\nNew Arrays by copyOf:\n");
        System.out.println("Integer Array: "
                           + Arrays.toString( Arrays.copyOf(intArr, 10)));
    }
}
```

Example 14.6: Copying a range of an array to collection

```
// This program illustrates the use of Arrays.copyOfRange() method.

import java.util.Arrays;

public class RangeCopyArraysDemo {
    public static void main(String[] args){
        int intArr[] = { 10, 20, 15, 22, 35}; // Get an array of int
        // To print the elements in one line
        System.out.println("Integer Array: "
                           + Arrays.toString(intArr));
        System.out.println("\nNew Arrays by copyOfRange:\n");
        // To copy the array into an array of new length
        System.out.println("Integer Array: "
                           + Arrays.toString(
                           Arrays.copyOfRange(intArr, 1, 3)));
    }
}
```

Example 14.7: Comparing two arrays

```
// This program illustrates the use of Arrays.deepEquals() method

import java.util.Arrays;

public class CompareArraysDemo {
    public static void main(String[] args) {
        int intArr1[][] = { { 10, 20, 15, 22, 35} }; // An input array
        int intArr2[][] = { { 10, 15, 22} }; // Another input array

        // To compare both the arrays
        System.out.println("Integer Arrays on comparison: "
                           + Arrays.deepEquals(intArr1, intArr2));
    }
}
```

Example 14.8: Hashcode generation of array items

```
//This program illustrates the use of Arrays.deepHashCode() method.

import java.util.Arrays;

public class ArraysOfHashcodeDemo {
    public static void main(String[] args){
        // Get the Array
        int intArr[][] = { { 10, 20, 15, 22, 35} };
        // To get the dep hashCode of the arrays
        System.out.println("Integer Array: "
                           + Arrays.deepHashCode(intArr));
    }
}
```

Example 14.9: Entering an item into an array

```
// This program illustrates the use of Arrays.fill() method

import java.util.Arrays;

public class InsertiontoArraysDemo {
    public static void main(String[] args){
        // Get the Arrays
        int intArr[] = { 10, 20, 15, 22, 35};
        int intKey = 22;
        Arrays.fill(intArr, intKey);
        // To fill the arrays
        System.out.println("Integer Array on filling: "
                           + Arrays.toString(intArr));
    }
}
```

Searching with Arrays

Example 14.10: Binary search on an array

```
  // This program illustrates the use of Binary Search method.

import java.util.Arrays;

public class BinarySerachArraysDemo {
    public static void main(String[] args){
        // Get the Array
        int intArr[] = { 10, 20, 15, 22, 35};
        Arrays.sort(intArr);
        int intKey = 22;
        System.out.println(intKey
                           + " found at index = "
                           + Arrays.binarySearch(intArr, intKey));
    }
}
```

Example 14.11: Binary search on a sub-list of an array

```
/* This program illustrates the use of Binary Search method within a sub list. */

import java.util.Arrays;

public class ArraysBinarySerachSubListDemo {
    public static void main(String[] args){
        int intArr[] = { 10, 20, 15, 22, 35}; // An int array as input
        Arrays.sort(intArr);    // Sort the array
        int intKey = 22;
        System.out.println ( intKey+ " found at index = "
            + Arrays.binarySearch(intArr, 1, 3, intKey));
    }
}
```

Sorting with Arrays

Example 14.12: Simple sorting

```
// This program illustrates the use of Arrays.sort() method

import java.util.Arrays;

public class SortingArraysDemo {
    public static void main(String[] args){
        // Get the Array
        int intArr[] = { 10, 20, 15, 22, 35 };
        // To sort the array using normal sort-
        Arrays.sort(intArr);
        System.out.println("Integer Array: "
                           + Arrays.toString(intArr));
    }
}
```

Example 14.13: Sorting a sublist

```
/* This program illustrates the use of second version of Arrays.sort() method.
*/

import java.util.Arrays;

public class SortingSublistDemo {
    public static void main(String[] args){
        // Get the Array
        int intArr[] = { 10, 20, 15, 22, 35};
        // To sort the array using normal sort
        Arrays.sort(intArr, 1, 3);
        System.out.println("Integer Array: "
                           + Arrays.toString(intArr));
    }
}
```

Example 14.14: Parallel sorting

```java
// This program illustrates the use of Arrays.parallelSort() method.

import java.util.Arrays;

public class ParallelSortDemo {
    public static  void main(String[] args){
        // Get the Array
        int intArr[] = { 10, 20, 15, 22, 35};
        // To sort the array using parallelSort
        Arrays.parallelSort(intArr);
        System.out.println("Integer Array: "
                           + Arrays.toString(intArr));
    }
}
```

Example 14.15: Comparator method for sorting

```
// This program illustrates the use  of Comparator interface

import java.util.*;
import java.lang.*;
import java.io.*;

// A class to represent a student.
class Student {
    int rollno;
    String name, address;
    // Constructor
    public Student(int rollno, String name, String address){
        this.rollno = rollno;
        this.name = name;
        this.address = address;
    }
                                        // Continued to next page ...
```

Example 14.15: Comparator method for sorting

```
 // Continued on...

    // Overriding toString() for Student class for printing
    public String toString(){
        return this.rollno + " " + this.name + " " + this.address;
    }
} // End of class Student

class Sortbyroll implements Comparator<Student> {
    // Used for sorting in ascending order of roll number
    public int compare(Student a, Student b){
        return a.rollno - b.rollno;
    }
}
```

Example 14.15: Comparator method for sorting

```
// Continued on...

// Driver class
class Sorting{
    public static void main(String[] args){
        Student[] arr = { new Student(111, "bbbb", "london"),
                          new Student(131, "aaaa", "nyc"),
                          new Student(121, "cccc", "jaipur") };
        System.out.println("Unsorted");
        for(int i = 0; i<arr.length; i++)
            System.out.println(arr[i]);
        Arrays.sort(arr, 1, 2, new Sortbyroll());
        System.out.println("\nSorted by rollno");
        for(int i = 0; i<arr.length; i++)
            System.out.println(arr[i]);
    }
}
```

Traversing Arrays

Example 14.16: Traversing an array using `spliterator()`

```
/* This method returns a Spliterator covering all of the specified
Arrays. */

import java.util.Arrays;

public class TraverseSpliterator {
    public static void main(String[] args){
        // Get the Array
        int intArr[] = { 10, 20, 15, 22, 35};
        // To sort the array using normal sort
        System.out.println("Integer Array: "
                           + Arrays.spliterator(intArr));
    }
}
```

Example 14.17: Traversal of a sublist of arrays

```
/* This method returns a Spliterator covering all of the specified
Arrays. */

import java.util.Arrays;

public class ArraysTraversalDemo {
    public static void main(String[] args){
        // Get the Array
        int intArr[] = { 10, 20, 15, 22, 35};
        // To sort the array using normal sort
        System.out.println("Integer Array: "
                           + Arrays.spliterator(intArr, 1, 3));
    }
}
```

REFERENCES

- The Complete Reference, Herbert Schildt, 9th Edition, Oracle Press
- https://cse.iitkgp.ac.in/~dsamanta/javads/index.html
- https://docs.oracle.com/javase/tutorial/

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

योगः कर्मसु कौशलम्

THANK YOU !

NPTEL ONLINE CERTIFICATION COURSES

Data Structures and Algorithms Using Java

Debasis Samanta

Department of Computer Science & Engineering, IIT Kharagpur

Module 04: Arrays

Lecture 15 : Vector Class for Arrays

CONCEPTS COVERED

- About the Vector Class
- Operations with Vectors
  - Creating vectors
  - Insertion
  - Deletion
  - Accessing elements in vectors
  - Bulk operations

About class `Vector`

- There are some popular Java utility classes, which are either obsolete or deprecated. There are many such classes, which JDK 5 has retrofitted to build very popular collections. Such classes are called legacy classes.

- **Dictionary**
- **Hashtable**
- **Propertioe**
- **Stack**
- **`Vector`**

- The legacy classes are defined in `java.util` package and the class Vector is one important of them.

About Class Vector

About the Vector as collection

1. All vectors are started with an initial capacity.
2. After this initial capacity is reached, the next time that you attempt to store an object in the vector, the vector automatically allocates space for that object in addition to few extra additional space for other objects.
3. By allocating more than just the required memory, the vector reduces the number of allocations that must take places as the vector grows. This reduction is important, because allocations are costly in terms of time.
4. The amount of extra space allocated during each reallocation is determined by the increment that you specify when you create the vector.
5. If you don't specify an increment, the vector's size is doubled by each allocation cycle.

Operations with Vector

Operations with vector

1. Creating a vector
2. Insertion of an item into a vector
3. Removing an item from a vector
4. Accessing a vector and its content.
5. Some frequently used bulk operations

Creating Vectors

Operation with `Vector`: Create

- Mainly how a new element can be added into a vector.
- The constructors are well defined in the `Vector` class, which can be used to create vector as a collection.

Constructor	Description
`Vector( )`	This constructor creates a default vector, which has an initial size of 10.
`Vector(int size)`	This constructor accepts an argument that equals to the required size, and creates a vector whose initial capacity is specified by size.
`Vector(int size, int incr)`	This constructor creates a vector whose initial capacity is specified by size and whose increment is specified by incr. The increment specifies the number of elements to allocate each time that a vector is resized upward.
`Vector(Collection c)`	This constructor creates a vector that contains the elements of collection c.

Example 15.1: Creating vector

```
// This program illustrates how a vector can be created

import java.util.*;

class VectorCreateDemo1 {
    public static void main(String[] arg)  {
        Vector v = new Vector();  // Create a vector of default size 10
        v.add(1);
        v.add(2);
        v.add("Debasis");
        v.add(3.4);
        v.add("Samanta");
        System.out.println("Vector is "+ v);
    }
}
```

Example 15.2: Creating a vector to store various types of elements

The following program uses a vector to store various types of numeric objects. It demonstrates several of the legacy methods defined by Vector. It also demonstrates the Enumeration interface.

```
import java.util.*;

public class VectorCreateDemo2 {
   public static void main(String args[]) {
      // initial size is 3, increment is 2
      Vector v = new Vector(3, 2);
      System.out.println("Initial size: " + v.size());
      System.out.println("Initial capacity: " + v.capacity());

      v.addElement(new Integer(1));
      v.addElement(new Integer(2));
      v.addElement(new Integer(3));
      v.addElement(new Integer(4));
      System.out.println("Capacity after four additions: " + v.capacity());

      v.addElement(new Double(5.45));
      System.out.println("Current capacity: " + v.capacity());

                                                  // Continued to next ...
```

Example 15.2: Creating a vector to store various types of elements

```
// Continued on ...

      v.addElement(new Double(6.08));
      v.addElement(new Integer(7));
      System.out.println("Current capacity: " + v.capacity());

      v.addElement(new Float(9.4));
      v.addElement(new Integer(10));
      System.out.println("Current capacity: " + v.capacity());

      v.addElement(new Integer(11));
      v.addElement(new Integer(12));
      System.out.println("First element: " + (Integer)v.firstElement());
      System.out.println("Last element: " + (Integer)v.lastElement());

      if(v.contains(new Integer(3)))
         System.out.println("Vector contains 3.");

      // Enumerate the elements in the vector.
      Enumeration vEnum = v.elements();
      System.out.println("\nElements in vector:");
      while(vEnum.hasMoreElements())
         System.out.print(vEnum.nextElement() + " ");
      System.out.println();
   }
}
```

Insertion Operation

Operation with `vector`: Insertion

- There is a default `add()` method there in the interface `List`.
- The legacy methods which are defined in the class Vector to perform insertion operations listed in the following table.

Method	Description
`void addElement(E element)`	The object specified by *element* is added to the vector.
`void insertElementAt(E element, int index)`	Adds element to the vector at the location specified by index.

Example 15.3: Insertion of elements into a vector

```
/* The following program shows how to insert the specified object as a
component in this vector at the specified index. */

import java.util.*;
class VectorInsertionDemo1 {
    public static void main(String[] arg) {
        Vector vec = new Vector(7);
        // use add() method to add elements in the vector
        vec.add(1);
        vec.add(2);
        vec.add(3);
        vec.add(4);
        vec.add(5);
        vec.add(6);
        vec.add(7);
        // insert 10 at the index 7
        vec.insertElementAt(10, 7);
        // checking vector
        System.out.println(" Vector: " + vec);
    }
}
```

Example 15.4: Inserting an element into a vector at specific position

```
// This program illustrates how an element can be inserted at a specified position in
vector. */

import java.util.*;

class VectorInsertionDemo2 {
    public static void main(String[] arg) {
        Vector v = new Vector(5); // Create a default vector of size 5
        v.add(1, 1);
        v.add(2, 2);
        v.add(0, "Debasis");
        v.add(3, "Samanta");
        v.add(4, 3);
        v.add(5, 6.9);    // Vector will grow automatically
        // insert 10 at the index 7
        v.insertElementAt(7, 10);
        System.out.println("Vector is "+ v);
    }
}
```

Example 15.5: Appending a set element into a vector

```
/* To append all of the elements in the specified Collection to the end of
this Vector. */

import java.util.*;
class VectorInsertionDemo3 {
    public static void main(String[] arg)  {
        ArrayList arr = new ArrayList();
        arr.add(3);
        arr.add("Oracle");
        arr.add("Java");
        arr.add(4);

        Vector v = new Vector(); // Creating a default vector
        v.addAll(arr); // copying all element of array list into vector
        System.out.println("vector v:"+ v); // checking vector v
    }
}
```

Deletion Operation

Operation with `vector`: Deletion

- The methods which are defined in the class Vector to perform deletion operations are listed in the following table.

Method	Description
`void removeAllElements( )`	Empties the vector. After this method executes, the size of the vector is zero.
`boolean removeElement(Object element)`	Removes element from the vector. If more than one instance of the specified object exists in the vector, then it is the first one that is removed. Returns true if successful and false if the object is not found.
`void removeElementAt(int index)`	Removes the element at the location specified by index.

Example 15.6: Deletion of elements

```
// To remove all of the elements from a vector.

import java.util.*;
class VectorDeletionDemo1 {
    public static void main(String[] arg)  {
        Vector v = new Vector(); // The initial size of the vector is 10
        v.add(0, 1);
        v.add(1, 2);
        v.add(2, “IIT");
        v.add(3, “Kharagpur");
        v.add(4, 3);
        System.out.println("Vector is: "+ v);
        v.clear(); // Clearing the vector
        System.out.println(“After clearing: "+ v); // checking vector
    }
}
```

Example 15.7: Deletion the first and last element from a vector

```
/* To remove an element at a specified location. It also illustrates the
removal of a specific element. */

import java.util.*;
class VectorDeletionDemo2 {
    public static void main(String[] arg) {
        Vector v = new Vector(); // Create a vector of (default) capacity 10
        v.add(1);
        v.add(2);
        v.add("India");
        v.add("Japan");
        v.add(4);
        v.removeElementAt(0); // Removing the element at 0, if it occurs
        System.out.println("After removal: "+ v); // Checking vector
        v.removeElement("Japan");
        System.out.println("After removal: "+ v); // Checking vector
    }
}
```

Example 15.8: Deletion of common element from a vector

```
/* The following program shows how to retain only the elements in this Vector
that are contained in the specified Collection. */

import java.util.*;
class VectorDeletionDemo3 {
    public static void main(String[] arg)
    {
        Vector vec = new Vector(7);
        Vector vecRetain = new Vector(4);

        // use add() method to add elements in the vector
        vec.add(1);
        vec.add(2);
        vec.add(3);
        vec.add(4);
        vec.add(5);
        vec.add(6);
        vec.add(7);
                                                    // Continued to next ...
```

Example 15.8: Deletion of common element from a vector

```
// Continued to next ...

        // This elements will be retained
        vecRetain.add(5);
        vecRetain.add(3);
        vecRetain.add(2);

        System.out.println("Calling retainAll()");
        vec.retainAll(vecRetain);

        // Let us print all the elements available in vector
        System.out.println("Numbers after removal :- ");

        Iterator itr = vec.iterator();

        while (itr.hasNext()) {
            System.out.println(itr.next());
        }
    }
}
```

Accessing Operation

Operation with vector

- The miscellaneous methods defined in class Vector to perform accessibility operations are listed in the following table.

Method	Description
int capacity( )	Returns the capacity of the vector.
boolean contains(Object *element*)	Returns **true** if *element* is contained by the vector, and returns **false** if it is not.
E elementAt(int *index*)	Returns the element at the location specified by *index*.
Enumeration<E> elements( )	Returns an enumeration of the elements in the vector.
void ensureCapacity(int *size*)	Sets the minimum capacity of the vector to *size*.
E firstElement( )	Returns the first element in the vector.
int indexOf(Object *element*)	Returns the index of the first occurrence of *element*. If the object is not in the vector, –1 is returned.
int indexOf(Object element, int start)	Returns the index of the first occurrence of element at or after start. If the object is not in that portion of the vector, –1 is returned.
boolean isEmpty( )	Returns true if the vector is empty, and returns false if it contains one or more elements.
E lastElement( )	Returns the last element in the vector.
int lastIndexOf(Object element)	Returns the index of the last occurrence of element. If the object is not in the vector, –1 is returned.
int lastIndexOf(Object element, int start)	Returns the index of the last occurrence of element before start. If the object is not in that portion of the vector, –1 is returned.
void setElementAt(E element, int index)	The location specified by index is assigned element.
int size( )	Returns the number of elements currently in the vector.

Example 15.9: Searching an element

```
/* The following program shows to check if a specific element is present in a vector
or not. */

import java.util.*;
class VectorSearchDemo1 {
    public static void main(String[] arg) {

        // create default vector
        Vector v = new Vector();

        v.add(1);
        v.add(2);
        v.add("C++");
        v.add("Python");
        v.add(3);

        // check whether vector contains "Java"
        if (v.contains("Java"))
            System.out.println("The element exists");
        else
            System.out.println("The element does not exist");
    }
}
```

Example 15.10: Searching a specific item at a specific position

```java
/* The following program shows how to return the index of the last occurrence of the
specified element in this vector, or -1 if this vector does not contain the element.
*/

import java.util.*;
class VectorSearchDemo2 {
    public static void main(String[] arg) {

        // create default vector of capacity 10
        Vector v = new Vector();

        v.add(1);
        v.add(2);
        v.add("Oracle");
        v.add(2);
        v.add("Java");
        v.add(4);

        // Checking last occurrence of 2
        System.out.println("last occurrence of 2 is: " + v.lastIndexOf(2));
    }
}
```

Example 15.11: Replacement after Searching

```
/* The following program shows how to set the component at the specified index of
this vector to be the specified object. */
*/
 import java.util.*;
class VectorUpdateDemo {
    public static void main(String[] arg)
    {
        // Create default vector of capacity 10
        Vector v = new Vector();

        v.add(1);
        v.add(2);
        v.add("Mother");
        v.add("Merry");
        v.add(4);

        // Set 4 at the place of 2
        v.setElementAt(4, 1);

        System.out.println("vector: " + v);
    }
}
```

Bulk Operations

Operation with `vector`: Bulk

- The miscellaneous methods defined in the class Vector to perform bulk operations are listed in the following table.

Method	Description
`Object clone( )`	Returns a duplicate of the invoking vector.
`void copyInto(Object array[ ])`	The elements contained in the invoking vector are copied into the array specified by *array*.
`Enumeration<E> elements( )`	Returns an enumeration of the elements in the vector.
`String toString( )`	Returns the string equivalent of the vector.
`void trimToSize( )`	Sets the vector's capacity equal to the number of elements that it currently holds.

Example 15.12: Cloning a vector

```
// This program duplicates a vector

import java.util.*;
class VectorCloningDemo {
    public static void main(String[] arg) {
        Vector v = new Vector();
        Vector v_clone = new Vector();
        v.add(0, 1);
        v.add(1, 2);
        v.add(2, "Oracle");
        v.add(3, "Java");
        v.add(4, 3);
        v_clone = (Vector)v.clone();
        System.out.println("Clone of v: "+ v_clone);
    }
}
```

Example 15.13: Copying a vector into a collection

```
/* The following program shows how to copy the components of this vector into
the specified array. */

import java.util.*;
class VectorCopyDemo {
    public static void main(String[] arg) {
        Vector vec = new Vector(7);
        // Use add() method to add elements in the vector
        vec.add(1);
        vec.add(2);
        vec.add(3);
        vec.add(4);
        vec.add(5);
        vec.add(6);
        vec.add(7);

        Integer[] arr = new Integer[7];

        // Copy a vector into array arr
        vec.copyInto(arr);

        System.out.println("elements in array arr: ");
        for (Integer num : arr) {
            System.out.println(num);
        }
    }
}
```

Example 15.14: Equality checking of two collections

```
/* The following program shows how to compare the specified Object with this
Vector for equality.. */

import java.util.*;
class VectorCheckDemo {
    public static void main(String[] arg) {
        // Create default vector of capacity 10
        Vector v = new Vector();

        v.add(1);
        v.add(2);
        v.add("Oracle");
        v.add("Java");
        v.add(4);

        // Second vector
        Vector v_2nd = new Vector();
        v_2nd.add(1);
        v_2nd.add(2);
        v_2nd.add("Java");
        v_2nd.add("Oracle");
        v_2nd.add(4);

        if (v.equals(v_2nd))
            System.out.println("both vectors are equal");
    }
}
```

Example 15.15: String representation of a vector

```
/* The following program shows how the toString() method is used to return a string
representation of this Vector, containing the String representation of each element. */

import java.util.*;
class VectorStringDemo {
    public static void main(String[] arg)  {

        // create default vector of capacity 10
        Vector v = new Vector();

        v.add(1);
        v.add(2);
        v.add(”Debasis");
        v.add(”Samanta");
        v.add(4);

        // string equivalent of vector
        System.out.println(" String equivalent of vector: " + v.toString());
    }
}
```

Example 15.16: Hashcode representation of a vector

```
/* The following program shows how to return the hash code value for this Vector. */

import java.util.*;
class VectorHashcodeDemo {
    public static void main(String[] arg)
    {
        Vector vec = new Vector(7);

        // use add() method to add elements in the vector
        vec.add(1);
        vec.add(2);
        vec.add(3);
        vec.add(4);
        vec.add(5);
        vec.add(6);
        vec.add(7);

        // checking hash code
        System.out.println("Hash code: " + vec.hashCode());
    }
}
```

REFERENCES

- The Complete Reference, Herbert Schildt, 9th Edition, Oracle Press
- https://cse.iitkgp.ac.in/~dsamanta/javads/index.html
- https://docs.oracle.com/javase/tutorial/

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

योगः कर्मसु कौशलम्

THANK YOU !