|  |
| --- |
| Arrays |

**What is an Array ?**

* An array is a linear data which is used to store elements of a specific type
* Array stores elements in a contiguous memory allocation
* The size of an array once declared cannot be changed during the program execution
* Each element of an array has a specific index

**Types Of Arrays**

Arrays

One Dimensional Array

Multi Dimensional Array

Three Dimensional

n Dimensional

Two Dimensional

**One Dimensional Array**

* An array with a bunch of values declared with a single index

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 5 | 4 | 10 | 11 | 18 | 87 | 12 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |

**Two Dimensional Array**

* An array with a bunch of values declared with a double index

Column Index

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 | 33 | 58 | 29 |
| 1 | 81 | 87 | 68 |
| 2 | 30 | 59 | 93 |

Row Index

* Element 81 has an index of i10
* Element 59 has an index of i21

|  |
| --- |
| Arrays Creation, Insertion Of Elements And Accessing Elements |

**Creation Of 1D Array**

The syntax for creation of one dimensional array is as follows

|  |
| --- |
| Datatype [ ] Reference\_Name = *new* Datatype [Size Of Array] |

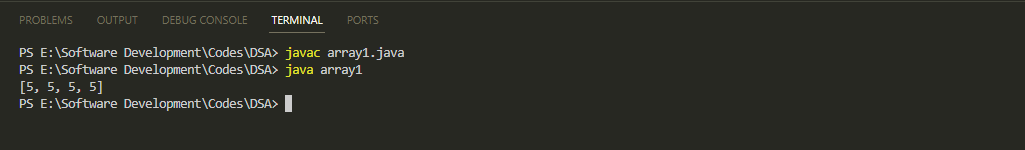
|  |
| --- |
| public class array1 {      public static *void* main(String[] *args*) {  *int*[] arrayName = new *int*[4];          arrayName[0] = 5;          arrayName[1] = 5;          arrayName[2] = 5;          arrayName[3] = 5;      }  } |

To check if the array is created or not, we can use the toString method from the Arrays Library

For that we need to import the Arrays Library (java.util.Arrays)

|  |
| --- |
| import java.util.Arrays;  public class array1 {      public static *void* main(String[] *args*) {  *int*[] arrayName = new *int*[4];          arrayName[0] = 5;          arrayName[1] = 5;          arrayName[2] = 5;          arrayName[3] = 5;          System.out.println(Arrays.toString(arrayName));      }  } |

Output



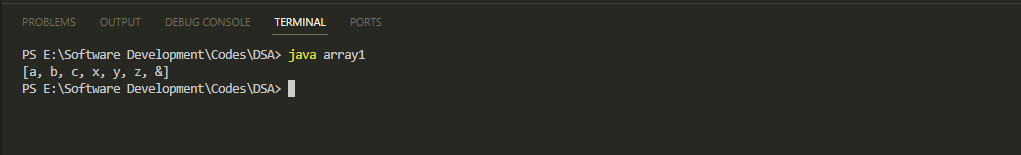
**Time Complexity For Creation Of Array**

|  |  |
| --- | --- |
| *int*[] arrayName = new *int*[4]; | O(1) |
| arrayName[0] = 5; | O(n) |
| arrayName[1] = 5; |
| arrayName[2] = 5; |
| arrayName[3] = 5; |
| System.out.println(Arrays.toString(arrayName)); | O(1) |
| Total Complexity | O(n+1+1) =O(n) |

However, if the array is initialized at once

|  |
| --- |
| import java.util.Arrays;  public class array1 {      public static *void* main(String[] *args*) {  *char*[] arrayName2 = { 'a', 'b', 'c', 'x', 'y', 'z', '&' };          System.out.println(Arrays.toString(arrayName2));      }  } |

Output



|  |  |
| --- | --- |
| *char*[] arrayName2 = { 'a', 'b', 'c', 'x', 'y', 'z', '&' }; | O(1) |
| System.out.println(Arrays.toString(arrayName)); | O(1) |
| Total Complexity | O(1+1) =O(1) |

**Insertion In Arrays**

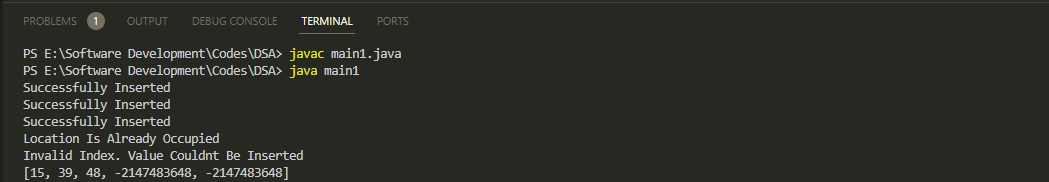
* First we will create an array class named One Dimensional Array

|  |
| --- |
| import java.util.Arrays;  public class OneDimensionalArray {  *int*[] arr = null;      public OneDimensionalArray(*int* *sizeOfArray*) {          arr = new *int*[sizeOfArray];          for (*int* i = 0; i < arr.length; i++) {              arr[i] = Integer.MIN\_VALUE;          }      }      public *void* insertValue(*int* *index*, *int* *value*) {          try {              if (arr[index] == Integer.MIN\_VALUE) {                  arr[index] = value;                  System.out.println("Successfully Inserted");              } else {                  System.out.println("Location Is Already Occupied");              }          } catch (*ArrayIndexOutOfBoundsException* *e*) {              System.out.println("Invalid Index. Value Couldnt Be Inserted");          }      }      public *void* showArray() {          System.out.println(Arrays.toString(arr));      }  } |

* Next we will add values

|  |
| --- |
| public class main1 {      public static *void* main(String[] *args*) {          OneDimensionalArray oda = new OneDimensionalArray(5);          oda.insertValue(0, 15);          oda.insertValue(1, 39);          oda.insertValue(2, 48);          oda.insertValue(1, 50);          oda.insertValue(6, 51);          oda.showArray();      }  } |

Output



**Time Complexity For Insertion In Arrays**

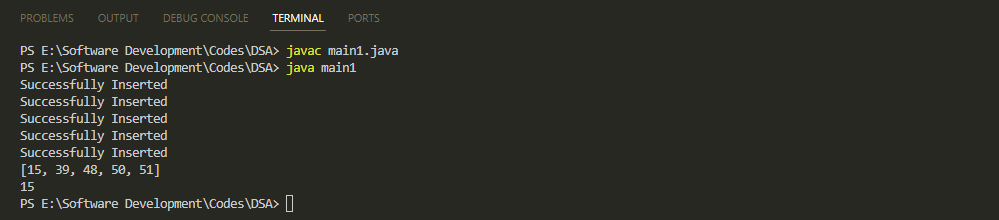
|  |  |
| --- | --- |
| *int*[] arrayName = new *int*[4]; | O(1) |
| if (arr[index] == Integer.MIN\_VALUE) { | O(1) |
| arr[index] = value; | O(1) |
| System.out.println("Successfully Inserted"); | O(1) |
| System.out.println("Location Is Already Occupied"); | O(1) |
| catch (*ArrayIndexOutOfBoundsException* *e*) | O(1) |
| System.out.println("Invalid Index. Value Couldnt Be Inserted"); | O(1) |
| Total Complexity | O(1) |

**Accessing Elements In Arrays**

* First we will insert some values in the array
* Next we will display the array
* Next we will print the first element to verify

|  |
| --- |
| public class main1 {      public static *void* main(String[] *args*) {          OneDimensionalArray oda = new OneDimensionalArray(5);          oda.insertValue(0, 15);          oda.insertValue(1, 39);          oda.insertValue(2, 48);          oda.insertValue(3, 50);          oda.insertValue(4, 51);          oda.showArray();          var firstElement = oda.arr[0];          System.out.println(firstElement);      }  } |

Output



**Time Complexity For Insertion In Arrays**

|  |  |
| --- | --- |
| var firstElement = oda.arr[0]; | O(1) |
| System.out.println(firstElement); | O(1) |
| Total Complexity | O(1) |

**Searching Elements In Arrays**

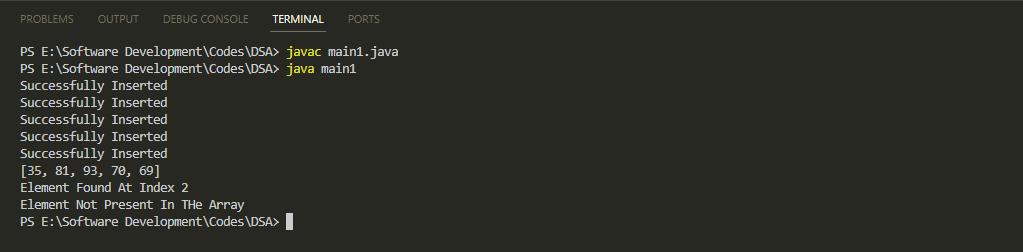
* First we will add search method in the one dimensional array class

|  |
| --- |
| public *void* searchElement(*int* *findElement*) {          for (*int* i = 0; i < arr.length; i++) {              if (arr[i] == *findElement*)                  System.out.println("Element Found At Index " + i);              else                  System.out.println("Element Not Found In The Array");          } |

* Next we will insert the elements
* Next we will search for the given elements

|  |
| --- |
| public class main1 {      public static *void* main(String[] *args*) {          OneDimensionalArray oda = new OneDimensionalArray(5);          oda.insertValue(0, 35);          oda.insertValue(1, 81);          oda.insertValue(2, 93);          oda.insertValue(3, 70);          oda.insertValue(4, 69);          oda.showArray();          oda.searchElement(93);          oda.searchElement(15);      }  } |

Output



**Time Complexity For Searching In Arrays**

|  |  |
| --- | --- |
| for (*int* i = 0; i < arr.length; i++) { | O(n) |
| if (arr[i] == *findElement*) | O(1) |
| System.out.println("Element Found At Index " + i); | O(1) |
| System.out.println("Element Not Found In The Array"); | O(1) |
| Total Complexity | O(n+1) = O(n) |

**Deleting Elements In Arrays**

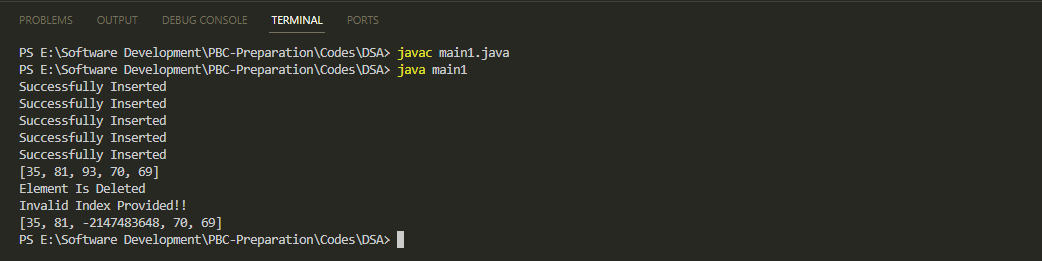
* First we will add delete method in the one dimensional array class

|  |
| --- |
| public *void* deleteElement(*int* *index*) {          try {              if (arr[*index*] != Integer.MIN\_VALUE) {                  arr[*index*] = Integer.MIN\_VALUE;                  System.out.println("Element Is Deleted");              } else                  System.out.println("Position Already Empty, Nothing To Delete!");          } catch (ArrayIndexOutOfBoundsException e) {              System.out.println("Invalid Index Provided!!");          }      } |

* Next we will provide the index from which element has to be deleted

|  |
| --- |
| public class main1 {      public static *void* main(String[] *args*) {          OneDimensionalArray oda = new OneDimensionalArray(5);          oda.insertValue(0, 35);          oda.insertValue(1, 81);          oda.insertValue(2, 93);          oda.insertValue(3, 70);          oda.insertValue(4, 69);          oda.showArray();          oda.deleteElement(2);          oda.deleteElement(6);  oda.showArray();      }  } |

Output



**Time Complexity For Deleting In Arrays**

|  |  |
| --- | --- |
| if (arr[*index*] != Integer.MIN\_VALUE) | O(1) |
| arr[*index*] = Integer.MIN\_VALUE; | O(1) |
| System.out.println("Element Is Deleted"); | O(1) |
| System.out.println("Position Already Empty, Nothing To Delete!"); | O(1) |
| catch (ArrayIndexOutOfBoundsException e) { | O(1) |
| System.out.println("Invalid Index Provided!!"); | O(1) |
| Total Complexity | O(1) |

**Time And Space Complexity For 1D Arrays**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sl No** | **Operation** | **Space Complexity** | **Time Complexity** |
| 1 | Creating an empty array | O(n) | O(1) |
| 2 | Inserting value in an array | O(1) | O(1) |
| 3 | Traversing a given array | O(1) | O(n) |
| 4 | Accessing a given cell | O(1) | O(1) |
| 5 | Searching a given value | O(1) | O(n) |
| 6 | Deleting a given value | O(1) | O(1) |

**Creation Of 2D Array**

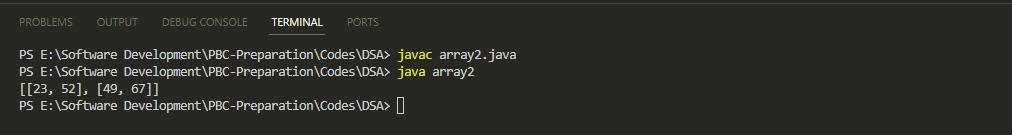
The syntax for creation of two dimensional array is as follows

|  |
| --- |
| Datatype [ ] [ ] Reference\_Name = *new* Datatype [No of rows][No of columns |

|  |
| --- |
| import java.util.Arrays;  public class array2 {      public static *void* main(String[] *args*) {          // initializing array  *int*[][] array2D = new *int*[2][2];          // assignment of values          array2D[0][0] = 23;          array2D[0][1] = 52;          array2D[1][0] = 49;          array2D[1][1] = 67;          // displaying the array          System.out.println(Arrays.deepToString(array2D));  }  } |

To check if the array is created or not, we can use the deepToString method from the Arrays Library. For that we need to import the Arrays Library (java.util.Arrays)

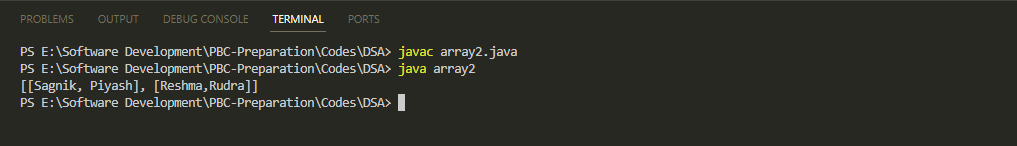
Output



2D arrays can be instantiated in the following manner as well

|  |
| --- |
| import java.util.Arrays;  public class array2 {      public static *void* main(String[] *args*) {          // initializing array          String[][] array2DStr = { { "Sagnik", "Piyash" }, { "Reshma,Rudra" } };          // displaying the array          System.out.println(Arrays.deepToString(array2DStr));      }  } |

Output



**Time Complexity For Creation Of 2D Arrays**

|  |  |
| --- | --- |
| *int*[][] array2D = new *int*[2][2]; | O(1) |
| array2D[0][0] = 23; | O(mn) |
| array2D[0][1] = 52; |
| array2D[1][0] = 49; |
| array2D[1][1] = 67; |
| System.out.println(Arrays.deepToString(array2D)); | O(1) |
| Total Complexity | O(mn+1+1) =O(mn) |

**Insertion In Arrays**

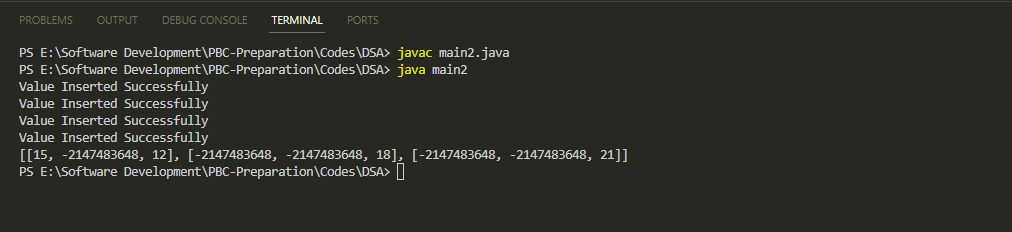
* First we will create an array class named Two Dimensional Array

|  |
| --- |
| import java.util.Arrays;  public class TwoDimensionalArray {  *int*[][] intArray2D = null;      public TwoDimensionalArray(*int* *row*, *int* *col*) {          this.intArray2D = new *int*[*row*][*col*];          for (*int* i = 0; i < *row*; i++) {              for (*int* j = 0; j < *col*; j++) {                  intArray2D[i][j] = Integer.MIN\_VALUE;              }          }      }      public *void* insertValue(*int* *row*, *int* *col*, *int* *val*) {          try {              if (intArray2D[*row*][*col*] == Integer.MIN\_VALUE) {                  intArray2D[*row*][*col*] = *val*;                  System.out.println("Value Inserted Successfully");              } else                  System.out.println("Position is Already Occupied");          } catch (ArrayIndexOutOfBoundsException e) {              System.out.println("Invalid Index");          }      }      public *void* showArray() {          System.out.println(Arrays.deepToString(intArray2D));      }  } |

* Next we will add values

|  |
| --- |
| public class main2 {      public static *void* main(String[] *args*) {          TwoDimensionalArray tda = new TwoDimensionalArray(3, 3);          tda.insertValue(0, 0, 15);          tda.insertValue(0, 2, 12);          tda.insertValue(1, 2, 18);          tda.insertValue(2, 2, 21);          tda.showArray();      }  } |

Output



**Time Complexity For Insertion In Arrays**

|  |  |
| --- | --- |
| if (intArray2D[*row*][*col*] == Integer.MIN\_VALUE) | O(1) |
| intArray2D[*row*][*col*] = *val*; | O(1) |
| System.out.println("Value Inserted Successfully"); | O(1) |
| System.out.println("Successfully Inserted"); | O(1) |
| System.out.println("Position is Already Occupied"); | O(1) |
| catch (ArrayIndexOutOfBoundsException e) | O(1) |
| System.out.println("Invalid Index") | O(1) |
| Total Complexity | O(1) |

**Accessing Elements In Arrays**

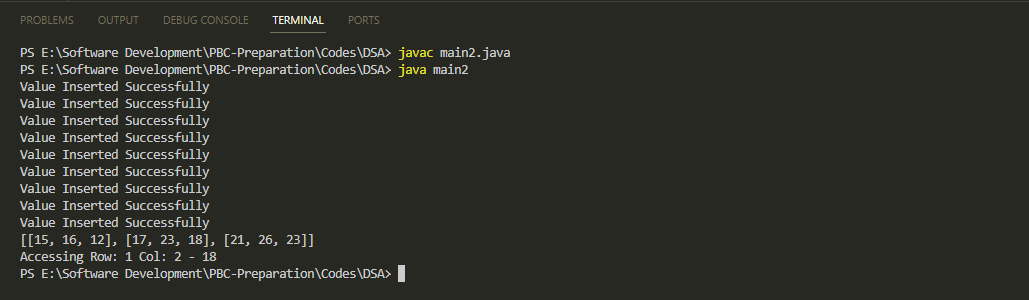
* First we will create a method for accessing elements

|  |
| --- |
| public *void* accessArrayElement(*int* *row*, *int* *col*) {          try {              System.out.println("Accessing" + " Row: " + *row* + " Col: " + *col* + " - " + intArray2D[*row*][*col*]);          } catch (ArrayIndexOutOfBoundsException e) {              System.out.println("Invalid Indexes");          }      } |

* Next we will insert some values in the array
* Next we will display the array
* Next we call the access method and verify if proper element is displayed

|  |
| --- |
| public class main2 {      public static *void* main(String[] *args*) {          TwoDimensionalArray tda = new TwoDimensionalArray(3, 3);          tda.insertValue(0, 0, 15);          tda.insertValue(0, 1, 16);          tda.insertValue(0, 2, 12);          tda.insertValue(1, 0, 17);          tda.insertValue(1, 1, 23);          tda.insertValue(1, 2, 18);          tda.insertValue(2, 0, 21);          tda.insertValue(2, 1, 26);          tda.insertValue(2, 2, 23);          tda.showArray();          tda.accessArrayElement(1, 2);      }  } |

Output



**Time Complexity For Accessing Elements In Arrays**

|  |  |
| --- | --- |
| System.out.println("Accessing" + " Row: " + *row* + " Col: " + *col* + " - " + intArray2D[*row*][*col*]); | O(1) |
| catch (ArrayIndexOutOfBoundsException e) | O(1) |
| System.out.println("Invalid Indexes"); | O(1) |
| Total Complexity | O(1) |

**Traversing In 2D Arrays**

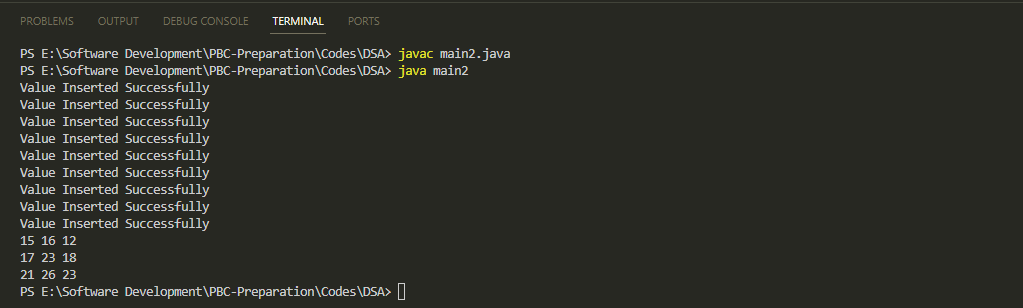
* First we will add traverse method in the two dimensional array class

|  |
| --- |
| public *void* traverseArray() {          for (*int* i = 0; i < intArray2D.length; i++) {              for (*int* j = 0; j < intArray2D[0].length; j++) {                  System.out.print(intArray2D[i][j] + " ");              }              System.out.println();          }      } |

* Next we will insert the elements
* Next we will traverse the array row by row

|  |
| --- |
| public class main2 {      public static *void* main(String[] *args*) {          TwoDimensionalArray tda = new TwoDimensionalArray(3, 3);          tda.insertValue(0, 0, 15);          tda.insertValue(0, 1, 16);          tda.insertValue(0, 2, 12);          tda.insertValue(1, 0, 17);          tda.insertValue(1, 1, 23);          tda.insertValue(1, 2, 18);          tda.insertValue(2, 0, 21);          tda.insertValue(2, 1, 26);          tda.insertValue(2, 2, 23);          tda.traverseArray();      }  } |

Output



**Time Complexity For Traversing In 2D Arrays**

|  |  |
| --- | --- |
| for (*int* i = 0; i < intArray2D.length; i++) {              for (*int* j = 0; j < intArray2D[0].length; j++) | O(mn) |
| System.out.print(intArray2D[i][j] + " ") | O(1) |
| System.out.println() | O(1) |
| Total Complexity | O(mn+1+1) = O(mn) |

**Searching Elements In 2D Arrays**

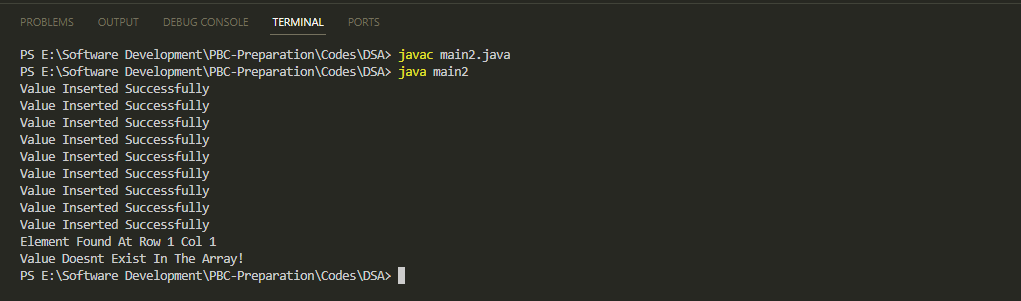
* First we will add search method in the two dimensional array class

|  |
| --- |
| public *void* searchArrayElement(*int* *value*) {          for (*int* i = 0; i < intArray2D.length; i++) {              for (*int* j = 0; j < intArray2D[0].length; j++) {                  if (*value* == intArray2D[i][j]) {                      System.out.println("Element Found At Row " + i + " Col " + j);                      return;                  }              }          }          System.out.println("Value Doesnt Exist In The Array!");      } |

* Next we will provide the value of the element which is to be found out

|  |
| --- |
| public class main2 {      public static *void* main(String[] *args*) {          TwoDimensionalArray tda = new TwoDimensionalArray(3, 3);          tda.insertValue(0, 0, 15);          tda.insertValue(0, 1, 16);          tda.insertValue(0, 2, 12);          tda.insertValue(1, 0, 17);          tda.insertValue(1, 1, 23);          tda.insertValue(1, 2, 18);          tda.insertValue(2, 0, 21);          tda.insertValue(2, 1, 26);          tda.insertValue(2, 2, 23);          tda.searchArrayElement(23);          tda.searchArrayElement(56);      }  } |

Output



**Time Complexity For Searching In 2D Arrays**

|  |  |
| --- | --- |
| for (*int* i = 0; i < intArray2D.length; i++)  for (*int* j = 0; j < intArray2D[0].length; j++) | O(mn) |
| if (*value* == intArray2D[i][j]) | O(1) |
| System.out.println("Element Found At Row " + i + " Col " + j) | O(1) |
| System.out.println("Value Doesnt Exist In The Array!"); | O(1) |
| Total Complexity | O(mn+1+1+1) = O(mn) |

**Deleting Elements In 2D Arrays**

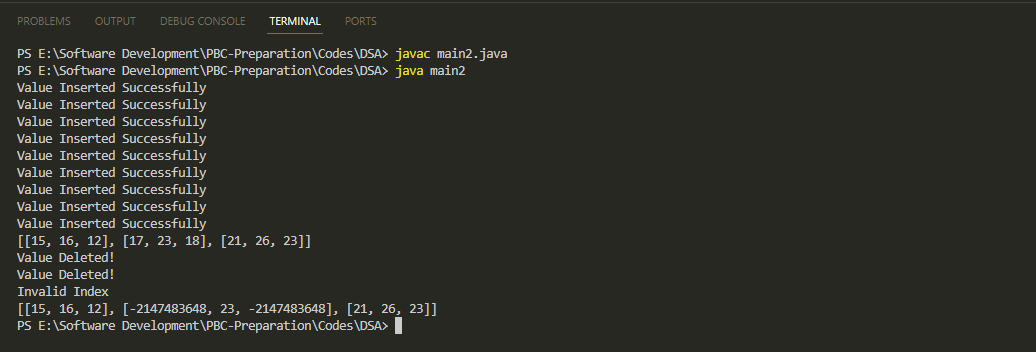
* First we will add delete method in the one dimensional array class

|  |
| --- |
| public *void* deleteArrayElement(*int* *row*, *int* *col*) {          try {              if (intArray2D[*row*][*col*] != Integer.MIN\_VALUE) {                  intArray2D[*row*][*col*] = Integer.MIN\_VALUE;                  System.out.println("Value Deleted!");              } else {                  System.out.println("Nothing To Delete!");              }          } catch (ArrayIndexOutOfBoundsException e) {              System.out.println("Invalid Index");          }      } |

* Next we will add element into the array
* Display the array
* Next we will provide the index from which element has to be deleted
* Display the array to verify

|  |
| --- |
| public class main2 {      public static *void* main(String[] *args*) {          TwoDimensionalArray tda = new TwoDimensionalArray(3, 3);          tda.insertValue(0, 0, 15);          tda.insertValue(0, 1, 16);          tda.insertValue(0, 2, 12);          tda.insertValue(1, 0, 17);          tda.insertValue(1, 1, 23);          tda.insertValue(1, 2, 18);          tda.insertValue(2, 0, 21);          tda.insertValue(2, 1, 26);          tda.insertValue(2, 2, 23);          tda.showArray();          tda.deleteArrayElement(1, 0);          tda.deleteArrayElement(1, 2);          tda.deleteArrayElement(3, 1);          tda.showArray();      }  } |

Output



**Time And Space Complexity For 2D Arrays**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sl No** | **Operation** | **Space Complexity** | **Time Complexity** |
| 1 | Creating an empty array | O(mn) | O(1) |
| 2 | Inserting value in an array | O(1) | O(1) |
| 3 | Traversing a given array | O(1) | O(mn) |
| 4 | Accessing a given cell | O(1) | O(1) |
| 5 | Searching a given value | O(1) | O(mn) |
| 6 | Deleting a given value | O(1) | O(1) |
| Introduction To Arraylist | | | | | |

**What is an ArrayList ?**

* An arraylist is a dynamic and resizable data structure which is a part of the java collection framework
* Arraylists can grow or shrink automatically with addition or deletion of elements
* An arraylist internally uses an array to store the elements but automatically handles resizing and
* Arraylist comes with many pre built useful methods for data manipulation
* Arraylists can contain data with multiple datatypes.
* Arraylists are slower than arrays

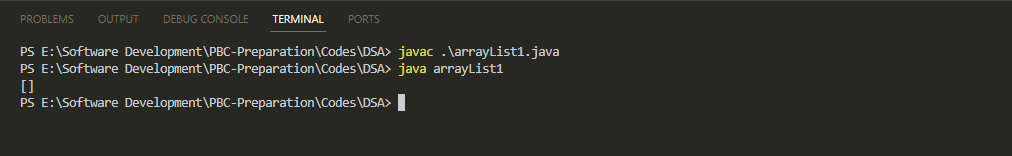
**Creating an ArrayList**

The syntax for creation of arraylist is as follows

|  |
| --- |
| ArrayList <*Generics*> arrayListName = new ArrayList( ) |

|  |
| --- |
| import java.util.ArrayList;  public class arrayList1 {      public static *void* main(String[] *args*) {          ArrayList<Integer> a1 = new ArrayList<>(30);          System.out.println(a1);      }  } |

Output

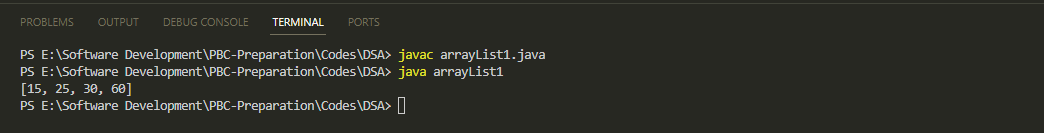


**Insertion in an ArrayList**

For adding an element in the arraylist we are going to use the add method. Add method appends an element in the end of the arraylist

|  |
| --- |
| import java.util.ArrayList;  public class arrayList1 {      public static *void* main(String[] *args*) {  ArrayList<Integer> a1 = new ArrayList<>(30);          a1.add(15);          a1.add(25);          a1.add(30);          a1.add(60);        System.out.println(a1);      }  } |

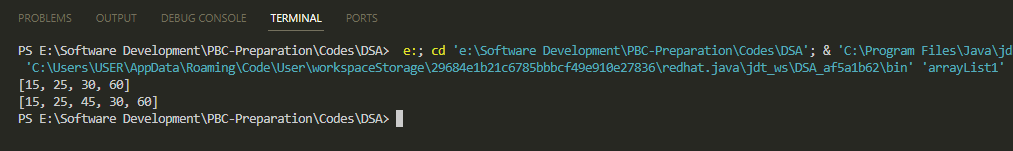
Output



Add method can also be used to insert an element at a specific position. Let us add 45 at 2nd index

|  |
| --- |
| import java.util.ArrayList;  public class arrayList1 {      public static *void* main(String[] *args*) {          ArrayList<Integer> a1 = new ArrayList<>(30);          a1.add(15);          a1.add(25);          a1.add(30);          a1.add(60);          System.out.println(a1);          a1.add(2, 45);          System.out.println(a1);      }  } |

Output



Time complexity of adding element at the end of the arraylist is Amortized O(1). For inserting the element at a specific index in the arraylist the time complexity will be O(n)

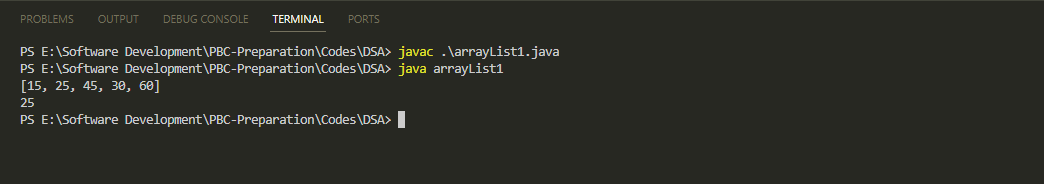
**Accessing Elements in an ArrayList**

In ArrayList, we can access a specific element by using the get method.

|  |
| --- |
| import java.util.ArrayList;  public class arrayList1 {      public static *void* main(String[] *args*) {          ArrayList<Integer> a1 = new ArrayList<>(30);          a1.add(15);          a1.add(25);          a1.add(30);          a1.add(60);          a1.add(2, 45);          System.out.println(a1);          System.out.println(a1.get(1));      }  } |

s

Output



Time complexity of accessing element in ArrayList using get method is O(1).

Space complexity of accessing element in ArrayList using get method is O(1).

**Traversal in an ArrayList**

Traversal in arraylist can happen using three methods

Traversal In ArrayList

Using For Loop

Using Iterator

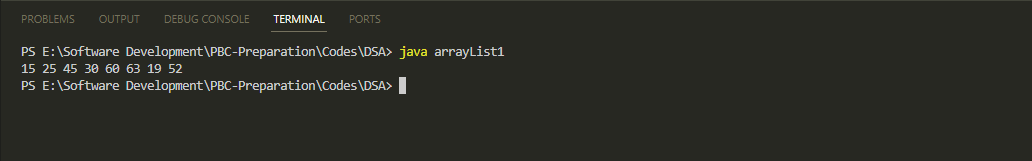
Using For Each Loop

**Using For Loop**

Let us see an example

|  |
| --- |
| import java.util.ArrayList;  public class arrayList1 {      public static *void* main(String[] *args*) {          ArrayList<Integer> a1 = new ArrayList<>(30);          a1.add(15);          a1.add(25);          a1.add(30);          a1.add(60);          a1.add(2, 45);          a1.add(63);          a1.add(19);          a1.add(52);          for (*int* i = 0; i < a1.size(); i++) {              System.out.print(a1.get(i) + " ");          }      }  } |

Output



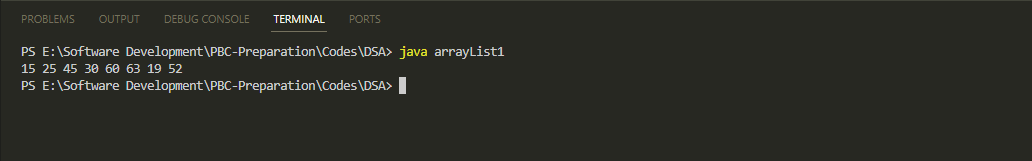
The time complexity of traversing an ArrayList using a for loop and get method is O(n)

**Using For Each Loop**

Let us see an example

|  |
| --- |
| import java.util.ArrayList;  public class arrayList1 {      public static *void* main(String[] *args*) {          ArrayList<Integer> a1 = new ArrayList<>(30);          a1.add(15);          a1.add(25);          a1.add(30);          a1.add(60);          a1.add(2, 45);          a1.add(63);          a1.add(19);          a1.add(52);          for (*int* i = 0; i < a1.size(); i++) {              System.out.print(a1.get(i) + " ");          }      }  } |

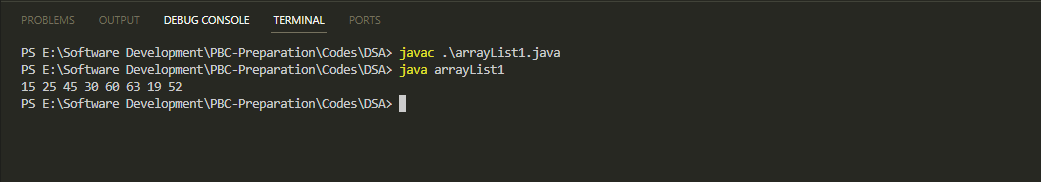
Output



The time complexity of traversing an ArrayList using a for each loop is O(n)

|  |
| --- |
| import java.util.ArrayList;  public class arrayList1 {      public static *void* main(String[] *args*) {          ArrayList<Integer> a1 = new ArrayList<>(30);          a1.add(15);          a1.add(25);          a1.add(30);          a1.add(60);          a1.add(2, 45);          a1.add(63);          a1.add(19);          a1.add(52);          for (*int* i : a1) {              System.out.print(i + " ");          }      }  } |

Output

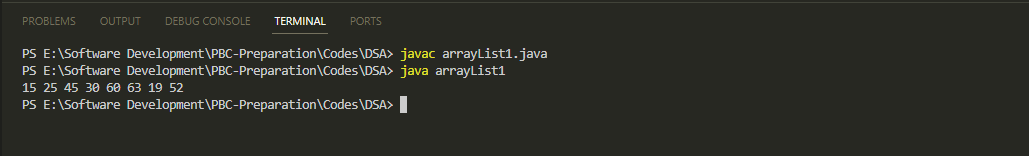


**Using Iterator**

Let us see an example

|  |
| --- |
| import java.util.ArrayList;  import java.util.Iterator;  public class arrayList1 {      public static *void* main(String[] *args*) {          ArrayList<Integer> a1 = new ArrayList<>(30);          a1.add(15);          a1.add(25);          a1.add(30);          a1.add(60);          a1.add(2,45);          a1.add(63);          a1.add(19);          a1.add(52);          Iterator<Integer> i = a1.iterator();          while (i.hasNext()) {              System.out.println(i.next());          }      }  } |

Output



The time complexity of traversing an ArrayList using a iterator is O(n)

**Searching Elements in an ArrayList**

In ArrayList, we can search an element by three methods

Searching In ArrayList

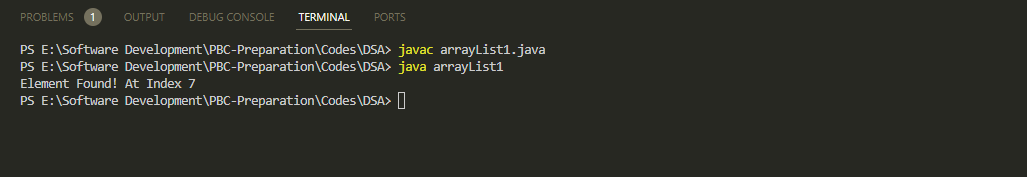
Using For Loop

Using indexOf

Using For Loop

|  |
| --- |
| import java.util.ArrayList;  import java.util.Iterator;  public class arrayList1 {      public static *void* main(String[] *args*) {          ArrayList<Integer> a1 = new ArrayList<>(30);          a1.add(15);          a1.add(25);          a1.add(30);          a1.add(60);          a1.add(2, 45);          a1.add(63);          a1.add(19);          a1.add(52);  *int* element = 52;          for (*int* i = 0; i < a1.size(); i++) {              if (a1.get(i) == element) {                  System.out.println("Element Found! At Index " + i);                  break;              }          }      }  } |

Output

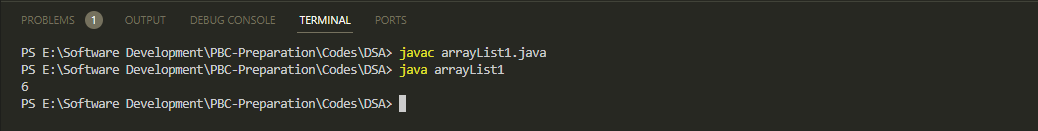


Time complexity of searching element in ArrayList using get method is O(n). Space complexity of accessing element in ArrayList using get method is O(1).

Using Index Of

|  |
| --- |
| import java.util.ArrayList;  import java.util.Iterator;  public class arrayList1 {      public static *void* main(String[] *args*) {          ArrayList<Integer> a1 = new ArrayList<>(30);          a1.add(15);          a1.add(25);          a1.add(30);          a1.add(60);          a1.add(2, 45);          a1.add(63);          a1.add(19);          a1.add(52);  *int* element = 19;          System.out.println(a1.indexOf(element));      }  } |

Output



**Deleting Elements in an ArrayList**

In ArrayList, we can delete an element by remove method. There are two of them

Remove In ArrayList

Remove (Using Parameter of Index)

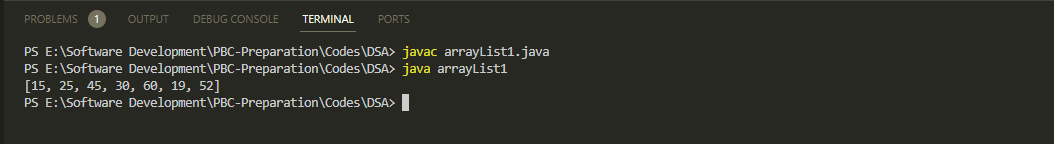
Remove (Using Parameter of Object)

**Remove (Using Parameter of Index)**

Remove method (parameter of index) removes that particular element and shifts all the elements to the left by one slot

|  |
| --- |
| import java.util.ArrayList;  import java.util.Iterator;  public class arrayList1 {      public static *void* main(String[] *args*) {          ArrayList<Integer> a1 = new ArrayList<>(30);          a1.add(15);          a1.add(25);          a1.add(30);          a1.add(60);          a1.add(2, 45);          a1.add(63);          a1.add(19);          a1.add(52);          a1.remove(5);          System.out.println(a1);      }  } |

Output

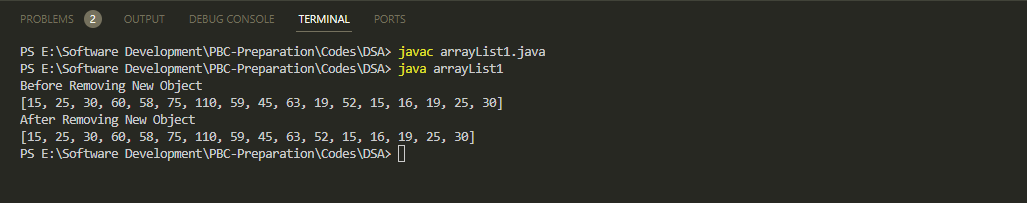


**Remove (Using object)**

Remove method (object) removes that particular element and shifts all the elements to the left by one slot

|  |
| --- |
| import java.lang.reflect.Array;  import java.util.ArrayList;  import java.util.Arrays;  import java.util.Iterator;  public class arrayList1 {      public static *void* main(String[] *args*) {          ArrayList<Integer> a1 = new ArrayList<>(30);          a1.add(15);          a1.add(25);          a1.add(30);          a1.add(60);          a1.add(58);          a1.add(75);          a1.add(110);          a1.add(59);          a1.add(45);          a1.add(63);          a1.add(19);          a1.add(52);          ArrayList<Integer> b1 = new ArrayList<>(Arrays.asList(15, 16, 19, 25, 30));          a1.addAll(b1);          System.out.println("Before Removing New Object ");          System.out.println(a1);          a1.remove(b1.get(2));          System.out.println("After Removing New Object ");          System.out.println(a1);      }  } |

Output



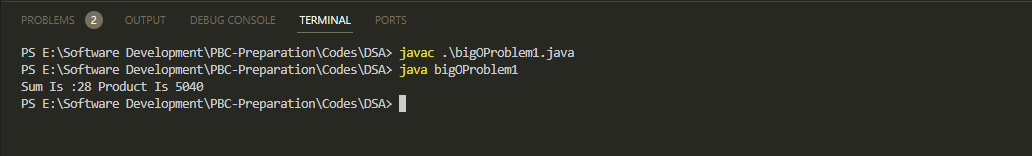
Time complexity of removing element in ArrayList is O(n). Space complexity of removing element in ArrayList is O(1)

|  |
| --- |
| Big O Problems :Arrays |

**Sum And Product Of Arrays**

|  |
| --- |
| class sumproductofArray {      public *void* arraySumAndProd(*int*[] *array*) {  *int* sum = 0;  *int* prod = 1;          for (*int* i = 0; i < *array*.length; i++) {              sum += *array*[i];          }          for (*int* i = 0; i < *array*.length; i++) {              prod \*= *array*[i];          }          System.out.println("Sum Is :" + sum + " Product Is " + prod);      }  }  public class bigOProblem1 {      public static *void* main(String[] *args*) {          sumproductofArray spa = new sumproductofArray();  *int*[] array = { 1, 2, 3, 4, 5, 6, 7 };          spa.arraySumAndProd(array);      }  } |

Output

****

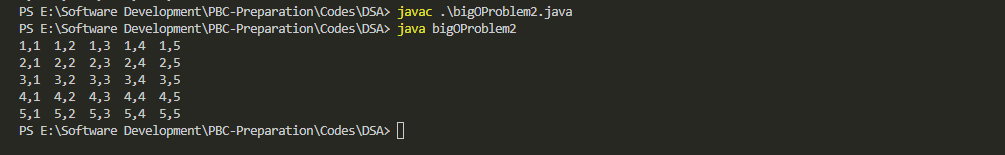
**Time Complexity**

|  |  |
| --- | --- |
| *int* sum = 0; | O(1) |
| *int* prod = 1; | O(1) |
| for (*int* i = 0; i < *array*.length; i++) | O(n) |
| sum += *array*[i]; | O(1) |
|  | O(1) |
| for (*int* i = 0; i < *array*.length; i++) | O(n) |
| prod \*= *array*[i]; | O(1) |
| System.out.println("Sum Is :" + sum + " Product Is " + prod); | O(1) |
| **Total Complexity** | O(n) |

**Print Pairs**

|  |
| --- |
| class printPairs {      public *void* printPairsFromArray(*int*[] *array*) {          for (*int* i = 0; i < *array*.length; i++) {              for (*int* j = 0; j < *array*.length; j++)                  System.out.print(*array*[i] + "," + *array*[j] + "  ");              System.out.println();          }      }  }  public class bigOProblem2 {      public static *void* main(String[] *args*) {  *int*[] arrayOfNum = { 1, 2, 3, 4, 5 };          printPairs pp = new printPairs();          pp.printPairsFromArray(arrayOfNum);      }  } |

Output

****

**Time Complexity**

|  |  |
| --- | --- |
| for (*int* i = 0; i < *array*.length; i++) {      for (*int* j = 0; j < *array*.length; j++) | O(n2) |
| System.out.print(*array*[i] + "," + *array*[j] + "  "); | O(1) |
| System.out.println(); | O(n) |
| **Total Complexity** | O(n2) |