# Unit 5: Arrays and ArrayLists

## Arrays

### Syntax

Declaration: type[] name;

Initialization: name = new type[size];

Declaration and Initialization: type[] name = new type[size];

type[] name = {item, item, …, item};

When you initialize an array with new, it constructs an array with space for size items of the given type. Then, all of the elements will be initialized to zero or the equivalent of zero.

Examples:

|  |  |
| --- | --- |
| int[] a = new int[3]; | 0  0  0  a |
| double[] a = new double[3]; | 0.0  0.0  0.0  a |
| boolean[] a = new boolean[3]; | false  false  false  a |
| String[] a = new String[3]; | null  null  null  a |

#### A note about null

null is a reference to an object that doesn’t exist.

null cannot be dereferenced through . notation.

For example,

String a = null; //ok

System.out.println(a.length()); //throws a NullPointerException at runtime

Since a is referencing null (no object) it has no length, so the program complains when it is run.

### Array Accessing

An *index* is a number between 0 and the length of the array – 1 (inclusive) that is used to access a particular element of an array.

Accessing: arr[index]

Assignment: arr[index] = value;

Size: arr.length

The first element of an array is always arr[0] and the last element of an array is always arr[arr.length - 1].

Warning! Any attempt to access an element that is not a valid index will cause an ArrayOutOfBoundsException.

For example:

arr[arr.length] //error since the last element is at arr.length - 1.

|  |  |
| --- | --- |
| double[] temps = new double[3] ;  temps[0] = 13.5 ;  temps[1] = -3.14;  System.out.println(temps[0]);  System.out.println(temps[1]);  System.out.println(temps[2]); | Output:  13.5  -3.14  0.0 |

### For Loops

It is common to want to loop over all the elements in an array, either to print the array out, find the largest value, or determine if a value is contained in the array. That makes arrays and for loops the best of friends.

Process an array from front-to-back:

for (int i = 0; i < arr.length; i++) {

//do something with arr[i]

}

Process an array from back-to-front:

for (int i = arr.length - 1; i >= 0; i--) {

//do something with arr[i]

}

## 2D Arrays

### Syntax

Declaration: type[][] name;

Initialization: name = new type[int][int];

Declaration and Initialization: type[][] name = new type[int][int]; or

type[][] name = {{val,…},{val,…},…,{val,…}};

Number of rows: name.length

Number of columns: name[0].length

### Accessing 2D arrays

It is easiest to think of a 2D array as a matrix of rows and columns, where the first number is the row number and the second number is the column number. For example:

int[][] matrix = new int[3][3];

matrix[0][0] = 4;

matrix[1][1] = 5;

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 | 4 | 0 | 0 |
| 1 | 0 | 5 | 0 |
| 2 | 3 | 0 | 0 |

matrix[2][0] = 3

Could be thought of as the grid:

### 2D Arrays as Arrays of Arrays

It is important to know that 2D arrays are in fact arrays of arrays. For example:

matrix[0] points to the row array with elements {4, 0, 0}

matrix[1] points to the row array with elements {0, 5, 0} and

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 | 4 | 2 | 0 |
| 1 | 0 | 5 | 0 |
| 2 | 3 | 0 | 0 |

matrix[2] points to the row array with elements {3, 0, 0}.

Therefore if the following code were executed, matrix would look like this:

int[] a = matrix[0];

a[1] = 2;

### Nested For-Loops

To iterate over (look at) all of the elements of a 2D array, we typically use nested for loops.

Processes the 2D array in row-major order (row by row)

for (int row = 0; row < matrix.length; row++) {

for (int col = 0; col < matrix[0].length; col++) {

//do something with matrix[row][col]

}

}

Processes the 2D array in column-major order (column by column)

for (int col = 0; col < matrix[0].length; col++) {

for (int row = 0; row < matrix.length; row++) {

//do something with matrix[row][col]

}

}

## ArrayLists

ArrayLists differ from Arrays in that they are dynamically resizeable. When arrays are initialized, their size must be explicitly given. ArrayLists can be initialized empty and then filled with the add method.

Note about the <> usage. <Type> specifies the Class of the object that the ArrayList will contain.

### Syntax

Declaration: ArrayList<Type> name;

Initialization: name = new ArrayList<Type>();

Declaration and Initialization: ArrayList<Type> name = new ArrayList<Type>(); or

List<Type> name = new ArrayList<Type>();

### Useful Methods

|  |  |  |
| --- | --- | --- |
| ArrayList | Array | Function |
| Type list.get(int index) | arr[index] | Returns the object at index |
| void list.set(int index, Type val) | arr[index] = val | Sets the object at index to val |
| int list.size() | arr.length | Returns the number of elements in list |
| void list.add(Type val) | N/A | Adds the object val to the end of the list, size is incremented |
| void list.add(int index, Type val) | N/A | Inserts the object val at index, all objects to the right of val have their indexes shifted up one and size is incremented |
| Type list.remove(int index) | N/A | Removes and returns the object at index, all objects to the right of index have their indexes shifted down one and size is decremented |

Warning! Modifying an ArrayList while looping over the elements can cause unwanted behavior.

For example:

Intent: Remove all even integers.

for (int i = 0; i < list.size(); i++) {

if (list.get(i) % 2 == 0) {

list.remove(i);

}

}

Problem: If two adjacent integers are even, then the second integer will be skipped.

Solution: Process list from back-to-front or decrement i whenever remove is called.

|  |  |
| --- | --- |
| int start = list.size() - 1;  int end = 0;  for (int i = start; i >= end; i--) {  if (list.get(i) % 2 == 0) {  list.remove(i);  }  } | int start = 0;  int end = list.size() - 1;  for (int i = start; i < end; i++) {  if (list.get(i) % 2 == 0) {  list.remove(i);  i--;  }  } |

### Enhanced For Loops

To loop over every item in a List or array, you can use an enhanced for loop.

|  |  |  |
| --- | --- | --- |
| for (type name : list) {  //do something with name } | for (int i = 0; i < list.size(); i++) {  //do something with list.get(i)  } | This accesses each item in list but doesn’t allow you to modify the item (unless it is an object that you call a mutator method on). |

Enhanced for loops are read-only for all primitives.

# Unit 5: Arrays and ArrayLists - Problems

1. Write a static method called minIndex that returns the first index of the smallest element in an array of integers.

For example:

minIndex(new int[]{3, -1, 2, 4, -1}) would return 1.

public static int minIndex(int[] a) {

}

1. Write a static method that returns an int[] array of the first n multiples of 3 or 5.

For example:

threesAndFive(7) would return [3, 5, 6, 9, 10, 12, 15]

public static int[] threesAndFives(int n) {

}

1. Write a static method called swapPairs that takes an array of Strings as a parameter and swaps each pair of Strings. If the array of Strings has an odd number of Strings, then the last String is not moved.

For example:

String[] fruits = {“apple”, “pear”, “banana”, “mango”, “grape”};

swapPairs(fruits); //fruits is now {“pear”, “apple”, “mango”, “banana”, “grape”}

public static void swapPairs(String[] s) {

}

1. Consider the following method.

public ArrayList<Integer> mystery(int n) {

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

for (int k = 1; k <= n; k++) {

seq.add(new Integer(k \* k + 3));

}

return seq;

}

Which of the following is printed as a result of executing the following statement?

System.out.println(mystery(6));

1. [3, 4, 7, 12, 19, 28]
2. [3, 4, 7, 12, 19, 28, 39]
3. [4, 7, 12, 19, 28, 39]
4. [39, 28, 19, 12, 7, 4]
5. [39, 28, 19, 12, 7, 4, 3]
6. Consider the following method that is intended to return the sum of the elements in the array key.

public static int sumArray(int[] key) {

int sum = 0;

for (int i = 1; i <= key.length; i++) {

/\* missing code \*/

}

return sum;

}

Which of the following statements should be used to replace /\* missing code \*/ so that sumArray will work as intended?

1. sum = key[i];
2. sum += key[i -1];
3. sum += key[i];
4. sum += sum + key[i - 1];
5. sum += sum + key[i];
6. Consider the following method.

public static void arrayMethod(int[] nums) {

int j = 0;

int k = nums.length - 1;

while (j < k) {

int x = nums[j];

nums[j] = nums[k];

nums[k] = x;

j++;

k--;

}

}

Which of the following describes what the method arrayMethod does to the array nums?

1. The array nums is unchanged.
2. The first value in nums is copied to every location in the array.
3. The last value in nums is copied to every location in the array.
4. The method generates an ArrayIndexOutOfBoundsException.
5. The contents of the array nums are reversed.
6. Consider the following code segment.

int[] oldArray = {1, 2, 3, 4, 5, 6, 7, 8, 9};

int[][] newArray = new int[3][3];

int row = 0;

int col = 0;

for (int value : oldArray) {

newArray[row][col] = value;

row++;

if ((row % 3) == 0) {

col++;

row = 0;

}

}

System.out.println(newArray[0][2]);

What is printed as a result of executing the code segment?

1. 3
2. 4
3. 5
4. 7
5. 8
6. Write a method called removeDuplicates that takes an ArrayList of Strings as a parameter and removes all duplicates from the list.

For example: Calling removeDuplicates on the list [“fuji”, “fuji”, “cameo”, “braeburn", “cameo”, “fuji”]

would turn the list into [“fuji”, “cameo”, “braeburn"].

public static void removeDuplicates(ArrayList<String> list) {

}