Notes: Arrays

Native Arrays

* **array**: an object that stores many values of the same type
* **element**: one value in an array
* **index**: a zero-based integer to access an element from an array

| ***index*** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *value* | 12 | 49 | -2 | 26 | 5 | 17 | -6 | 84 | 72 | 3 |

* In the array above, 12 is stored as element 0, 5 is stored as element 4, and 3 is stored as element 9
* To declare and initialize an array use type[] name = new type[length];

// creates a new int array to hold 10 values; all starting at zero  
int[] numbers = new int[10];

* To access elements use name[index]
* To modify elements use name[index] = value

Details on Arrays

* Different array types have different default values; the default values for common types are shown in the table below

| **Type** | **Default Value** |
| --- | --- |
| int | 0 |
| double | 0.0 |
| boolean | false |
| String | null |

* If you know the values of an array you want to define, you can use the array initializer syntax
  + int[] numbers = {2, 3, 5, 7, 11, 13, 17};
* Arrays have **random access** which means you can access any element in the array without having to loop through the entire array
* The legal indexes of an array are between 0 and the array's length - 1. If you read or write outside of this range you will generate an ArrayIndexOutOfBoundsException
* Arrays and for-loops are best friends
* An array's length field stores the number of elements, name.length, notice that there are no parens after length because it is a field and not a method

Printing Arrays

* Arrays are objects, they behave different from primitive types (e.g., int, char, boolean)
* Every object in Java has a special method named toString() that defines how that object should be converted into a String
* The default toString() method for arrays is to print the memory address of the array, so you should need to use a for-loop to print the contents of an array

// print out the elements of an array

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

System.out.print(numbers[i] + " ");

}

System.out.println();

* **Instead**, you can use the toString method of the Arrays class to print your array

System.out.println(Arrays.toString(name));

where *name* is the name of the array you are printing

Code Examples

// print out the elements of an array

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

System.out.print(numbers[i] + " ");

}

System.out.println();

// multiply every value in an array by 2

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

numbers[i] = 2 \* i;

}

// count how many values are above 5

int count = 0;

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

if (numbers[i] > 5) {

count++;

}

}

// using an array to count the number of each digit in a number

int num = 229231007;

int[] counts = new int[10];

while (num > 0) {

int digit = num % 10;

counts[digit]++;

num = num / 10;

}

// print a histogram for the values in an array

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

System.out.print(i + ": ");

for (int j = 0; j < counts[i]; j++) {

System.out.print("\*");

}

System.out.println();

}

Arrays Class

The Arrays class (java.util.Arrays) has useful methods for manipulating arrays:

Methods of the Arrays class

| **Method Name** | **Description** |
| --- | --- |
| equals(array1, array2) | returns true if the two arrays contain the same elements in the same order |
| fill(array, value) | sets every element in the array to have the given value |
| sort(array) | arranges the elements in the array into ascending order |
| toString(array) | returns a String representing the array |

Code Examples

// create an array called numbers that stores 100 int's

int[] numbers = new int[100];

// use the Arrays class to fill all 100 spots in numbers to be -5

Arrays.fill(numbers, -5);

// use the Arrays class to print all the values in numbers

System.out.println(Arrays.toString(numbers));

Notes: More on Arrays

Value Semantics

* Value semantics are used with primitive types in Java (e.g., int, char, boolean)
* If you set a variable equal to another it takes on that new value, but does not constantly update so those variables always have the same values

int x = 3;

int y = 4;

int z = x;

z = 42; // z is set to 42, but x does not update to also be 42 (x is still 3)

* Because of this, if a primitive variable is change inside a method, the variable in the main is not updated (unless used in conjunction with a return statement)

int x = 10;

int y = 20;

add(x,y); // after this call x will still be 10, y will still be 20

y = add(x,y); // after this call x will still be 10, y will be 30

public static int add(int x, int y) {

// this changes the value of the local var x (not the one in the main)

x += y;

// this will print the updated values of x and y,

//even though these changed values may not persist to the main

System.out.println(x + " " + y);

// this returns the local var x's value to where the method was called

return x;

}

Reference Semantics

* Arrays use reference semantics (to contrast, the primitive types use value semantics)
* Arrays use references to where it is stored in memory, not the actual values

int[] values = {1, 2, 3};

int[] values2 = values; // values and values2 point to the same array in memory

values[0] = 7; // changes both the first value of values, and the first value of values2 to 7

* Because of this, you do not need to return an array from a method to get its updated values, since both arrays refer to the same array in memory there is no reason to return the array

int[] values = {4, 5, 6};

triple(values); // after this call, values will contain 12, 15, 18

public static void triple(int [] values) {

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

values[i] = values[i] \* 3;

}

}

Array Mystery Problems

* Pay attention to the loop bounds, often they will not go until the length of an array
* You should reference the updated values of the array when evaluating the answer instead of the original array
* Note that a[i] and i are different! a[i] refers to the element at index i, whereas i is just an index

More Code Examples

// return the sum of all elements in an array of integers

public static int sum(int[] values) {

int sum = 0;

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

sum += values[i];

}

return sum;

}

// print the values of an array

int[] primes = {2, 3, 5, 7, 11, 13, 17};

System.out.println(Arrays.toString(primes));

Solving an array problem "in place" refers to solving an array problem without creating a new array; you are able to rearrange the array's elements in the original array.

// reverse the elements of an array

public static void reverse(int[] values) {

for (int i = 0; i < values.length / 2; i++) {

int temp = values[i];

values[i] = values[values.length - 1 - i];

values[values.length - 1 - i] = temp;

}

}

// apply Math.abs to all elements of an array

public static void makePositive(int[] values) {

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

values[i] = Math.abs(values[i]);

}

}