**Arrays**

* An array is a data structure that stores a collection of values of the same type.

An ***array***is a flexible structure for storing a sequence of values that are all of the same type.

**Array Subscripts**

* The values stored in an array are called ***elements****.*
* The individual elements are accessed using an integer ***index****.*
* An ***index*** is an integer indicating the *position* of a particular value in a data structure.
* In Java, array indexes must always be an integral type.

This includes

1. byte,
2. short,
3. int,
4. long

An array subscript can be

1. a literal value,
2. a variable (constant or non-constant),
3. or an expression that evaluates to an integral type.

* You can insert a value in an array using an integer index (location in the array).
* You can access a value from an array using an integer index.
* For example, if a is an array of integers, then a[i] is the ith integer in the array.

**Array Declaration**

* To declare an array specify
  + the **type** of elements that will be stored in the array,
  + then **brackets ([ ])** to show that it is an array of that type,
  + then at least one space,
  + and then a **name** for the array.
* Declarations are of the following syntax:

type[] name;

* **NOTE:** You can define an array variable either as

int[] arr;

or as

int arr[];

Most Java programmers prefer the former style because it neatly separates the type int[] (integer array) from the variable name.

* Here is an example of a declaration of an array arr of integers:

int[] arr;

In this case you want an array of three double values, which you can construct as follows:

* Arrays are objects, which means that this statement only *declares* the variable arr.
* Since arrays are objects, any variable that declares an array holds a *reference* to an object.
* Simply declaring a variable isn’t enough to bring the object into existence.
* **The declarations do not initialize the array**.
* Therefore, arr not yet initialized.
* Here’s another example of more declarations in code:

Graphical user interface, text, application, email

Description automatically generated

**Array Initialization**

* Initialization of arrays are of the following syntax:

<type>[] <name> = new <type>[<length>]

* To initialize an array, use
  + the **new** keyword
  + followed by the **type**
  + followed by the **size** of the array in **square brackets**
* following the initial declaration of the array.
* You can use any type as the element type, although the left and right sides of this statement have to match.
* Here is an example initialization of an array arr of integers.
* In this case you want an array of three double values, which you can construct as follows:

double[] temperature = **new** double[3];

* This statement ***declares and initializes*** an array of 3 doubles.
* When Java executes this line of code to construct the array of temperatures, it will construct an array of three double values, and return a reference to the variable temperature. The *temperature* variable now holds a reference to the array object.

A screenshot of a computer

Description automatically generated with low confidence

* As you can see, the variable *temperature* is not itself the array.
* Instead, it stores a **reference** to the array.
* The array length does not need to be a constant: new int[n] creates an array of length n.
* The array elements are *indexed from 0 to 99* (and not 1 to 100).
* To refer to an individual element of the array, you combine the name of the variable that refers to the array (temperature) with a specific index ([0], [1], or [2]).
* So, there is an element known as temperature[0], an element known as temperature[1], and an element known as temperature[2].

**Default Initialization Values**

* When you create an array of numbers, the elements are filled with default values.

*Default Initialization in Class Fields*

* It's not always necessary to assign a value when a field is declared.
* Fields that are declared but not initialized will be set to a reasonable default by the compiler.
* Relying on such default values, however, is generally considered **bad programming style**.

|  |  |
| --- | --- |
| **Data Type** | **Default Value (for fields)** |
| byte | 0 |
| short | 0 |
| int | 0 |
| long | 0L |
| float | 0.0f |
| double | 0.0d |
| char | '\u0000' |
| String (or any object) | null |
| boolean | false |

*Default Initialization of Local Variables*

* Local variables are slightly different than class fields.
* The compiler **never assigns a default value to an uninitialized local variable**.
* If you cannot initialize your local variable where it is declared, make sure to **assign it a value before you attempt to use it**.
* Accessing an uninitialized local variable will result in a compile-time error.

*Default Value for Objects - null*

* Arrays of objects are initialized with the special value **null**, which indicates that they do not (yet) hold any objects.
* For example:

String[] names = new String[10];

creates an array of ten strings, all of which are null.

If you want the array to hold empty strings, you must supply them like follows:

for (int i = 0; i < 10; i++) names[i] = "";

**How Elements Are Stored**

* When you create an array of a **primitive type** (like int) with initial values specified, space is allocated for the specified number of items of that type and the values in the array are set to the specified values.
* When you create an array of an **object type** (like String) with initial values, space is set aside for that number of object references. The objects are created, and the object references set so that the objects can be found.

Graphical user interface, text, application

Description automatically generated

**Array Initializers and Anonymous Arrays**

* Java has a **shortcut** for creating an array object and supplying initial values at the same time called **array initializers**.
* In this case you don’t specify the size of the array, it will be determined from the number of values that you specify.
* Place every value you want in the array between a pair of curly brackets.
* Here’s an example of the syntax at work:

int[ ] highScores = { 99, 98, 98, 88, 68};

String[ ] names = {"Jamal", "Emily", "Destiny", "Mateo", "Sofia"};

* Notice that you **do not call new** when you use this syntax.
* You can even initialize an ***anonymous array***:

new int[] { 17, 19, 23, 29, 31, 37 }

* This expression allocates a new array and fills it with the values inside the braces.
* It counts the number of initial values and sets the array size accordingly. You can use this syntax to reinitialize an array without creating a new variable.
* For example,

smallPrimes = new int[] { 17, 19, 23, 29, 31, 37 };

is shorthand for

int[] anonymous = { 17, 19, 23, 29, 31, 37 };

smallPrimes = anonymous;

**All Ways to Initialize Array**

Here are all the legal (and illegal) ways to declare and initialize an array

Text

Description automatically generated with medium confidence