Chapter 3: Container Types

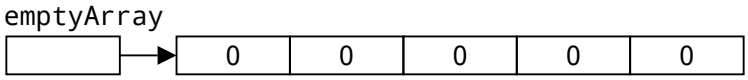
Topic 3.1: Container Types in Java

Arrays in Java

An array in Java is a contiguous block of memory statically allocated to store a set number of primitive types or object references. For example may declare an array of integers and allocate memory to store five 32-bit integer values with the code:

```
int[] emptyArray = new int[5];
```

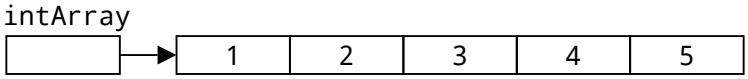
This results in a memory data structure represented by the following diagram. The variable `emptyArray` holds a reference to the location of the array, while each element in the array holds a 32-bit integer value. Note that Java automatically initializes array element values to a default value, which happens to be zero for integer types.



Java also allows us to declare an array of integers and initialize the values with code such as:

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

A diagram of the resulting memory data structure show in the diagram below.

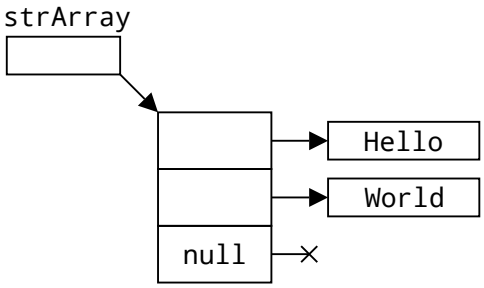


When allocating an array of Java objects, each element stores a reference to an object. Examine the following code segment:

```
String[] strArray = new String[3];
strArray[0] = "Hello";
strArray[1] = "World";
```

The first line of the above code allocates a block of memory to store three object references. When allocating an array of object references using the `new` keyword, Java initializes each array element to `null`.

In this example, the variable `strArray` holds the reference to the array. The next two lines of code set the first two references in the array to point to two `String` objects. The final memory structure is shown diagrammatically here:



The String objects in the diagram above have been simplified.

Collection Hierarchy in Java

Arrays are the only language-level constructs that implement a container in Java. A number of other containers are implemented in classes in the standard library, sometimes using an array as the building block. Many of the most common container data structures belong to the `Collection` interface. (An `interface` in Java is a contract that specifies a set of methods that a class must implement, without providing any implementation details itself.) The Collection hierarchy includes these top-level interfaces:

- `interface List<E>`
  - an ordered collection of elements; it is basically a wrapper around an array that allows automatic resizing with insertion and deletion of array elements.
- `interface Queue<E>`
  - a collection that is intended to hold elements so that they come out in a set order, perhaps first-in-first-out (FIFO), last-in-first-out (LIFO, or also called a stack), or a priority queue.
- `interface Set<E>`
  - a collection that contains no duplicate elements and where the order of elements does not matter. If code attempts to add an element that is already in the set, it will not be added.

Each `interface` of the Collection hierarchy has one or more `class` that implements it, and it is up to the programmer to choose the implementation will be most efficient for their use case.

As you will learn, Python implements lists and sets at the language level. Queues are implemented in the Python standard library.

The Java Map Class: Key-Value Pairs

It is very common to want to associate a data value with a “key” to tag it with a meaning. A real-world analogies of this would be a phone book, where each phone number is associated with a name as the key, or a dictionary, where each definition is tagged with the word it defines as the key. To find the phone number, we search for the name; to find the definition, we search for the word.

The standard library of Java contains `interface Map<K, V>`, and a number of classes that implement the interface.

The equivalent to a map in Java is called a *dictionary* in Python, and it is implemented at the language level.