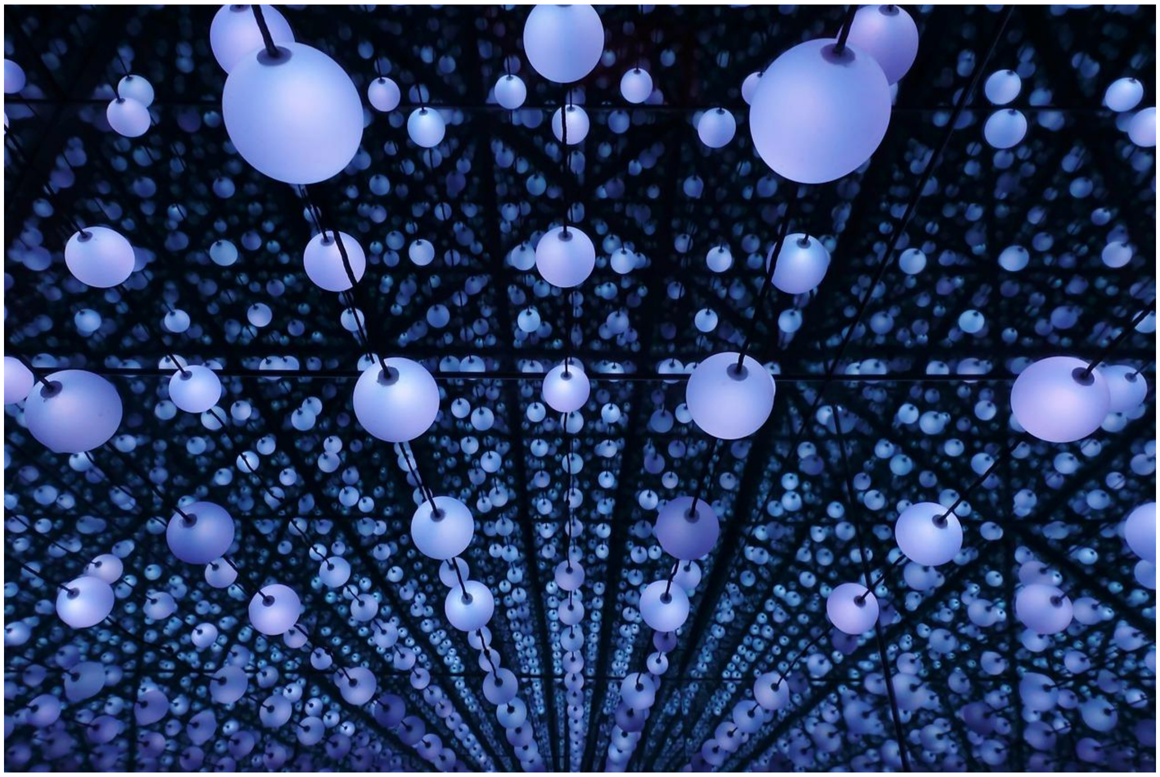

ARRAYS AND COLLECTIONS

OBJECT ORIENTED PROGRAMMING I

Sercan Külcü | Object Oriented Programming I | 10.01.2023

Contents

Introduction

Arrays and collections are essential data structures used in Java programming. Arrays are a simple and efficient way to store and access data, while collections provide more advanced features such as dynamic sizing, sorting, and searching. In this chapter, we will delve into the fundamental concepts of arrays and collections in Java and how to use them effectively in your programs.

Array

In Java, an array is a data structure that can store a fixed number of elements of the same data type. Arrays are declared using the [] syntax, and the elements of an array are accessed using their index. For example, the following code declares an array of integers and stores the values 1, 2, and 3 in it:

```
int[] myArray = new int[]{1, 2, 3};
```

To access the element at index 0, we would use the following code:

```
int element = myArray[0];
```

Arrays are a very powerful tool in Java, and they can be used to store a wide variety of data. They are often used to store collections of data, such as lists of strings or numbers. Arrays can also be used to store more complex data structures, such as graphs and trees.

Here are some of the advantages of using arrays in Java:

- Arrays are efficient data structures. They can be accessed quickly and easily, and they require very little memory.
- Arrays are easy to use. They can be declared and initialized in a single line of code, and they can be accessed using simple index notation.
- Arrays are versatile. They can be used to store a wide variety of data, including primitive types, objects, and even other arrays.

Here are some of the disadvantages of using arrays in Java:

- Arrays are fixed-size. Once an array is created, its size cannot be changed.
- Arrays can be slow to iterate over. If you need to access all of the elements in an array, you will need to use a for loop.
- Arrays can be difficult to debug. If there is an error in an array, it can be difficult to find the source of the error.

Overall, arrays are a powerful and versatile data structure that can be used to store a wide variety of data in Java. However, it is important to be aware of their limitations, such as their fixed size and their potential for slow performance.

Collections Framework

The Java Collections Framework provides a wide variety of collection classes, each with its own strengths and weaknesses. Here is a list of the most commonly used collection classes:

- List - A list is an ordered collection of objects. Lists can be accessed by their index, and they can be added to, removed from, or replaced at any time.
- Set - A set is an unordered collection of objects. Sets do not allow duplicate elements, and they can be accessed using iterator objects.
- Map - A map is a collection of key-value pairs. Maps can be used to store data in a way that is easy to retrieve.
- Queue - A queue is a first-in, first-out (FIFO) collection of objects. Objects are added to the end of the queue, and they are removed from the beginning of the queue.
- Deque - A deque is a double-ended queue. Objects can be added to or removed from the beginning or end of the deque.

In addition to these core collection classes, the Java Collections Framework also provides a number of utility classes, such as:

- Collections - This class provides a number of static methods for working with collections, such as sorting, searching, and copying.
- Arrays - This class provides a number of methods for working with arrays, such as sorting, searching, and copying.

The Java Collections Framework is a powerful and versatile tool for storing and manipulating data. By understanding the different types of collections and the strengths and weaknesses of each, you can choose the right collection for your needs.

Collections

Here is a list of all the collections classes in Java:

List - An ordered collection of objects.

- ArrayList - An array-based list.
- LinkedList - A linked list.
- Vector - A synchronized list.

Set - An unordered collection of objects that does not allow duplicate elements.

- HashSet - A hash-based set.
- LinkedHashSet - A hash-based set that maintains the insertion order of elements.
- TreeSet - A tree-based set that maintains the elements in sorted order.

Map - A collection of key-value pairs.

- HashMap - A hash-based map.
- LinkedHashMap - A hash-based map that maintains the insertion order of elements.
- TreeMap - A tree-based map that maintains the elements in sorted order.

Queue - A first-in, first-out (FIFO) collection of objects.

- ArrayDeque - An array-based deque.
- LinkedList - A linked list.
- PriorityQueue - A priority queue.

Deque - A double-ended queue.

- ArrayDeque - An array-based deque.
- LinkedList - A linked list.

These are just the most common collection classes in Java. There are many other collection classes available in the Java Collections Framework.

ArrayList

In Java, ArrayList is a class that implements the List interface. It is a resizable array, which means that its size can be changed dynamically. ArrayList is one of the most commonly used collection classes in Java.

ArrayLists are created using the new ArrayList() constructor. The constructor takes an optional argument that specifies the initial capacity of the array. If the initial capacity is not specified, the default capacity is 10.

To add an element to an ArrayList, use the add() method. The add() method takes an object as its argument. The object is added to the end of the array.

To remove an element from an ArrayList, use the remove() method. The remove() method takes an object as its argument. The object is removed from the array.

To get the element at a specific index in an ArrayList, use the get() method. The get() method takes an integer as its argument. The integer specifies the index of the element to be returned.

To set the element at a specific index in an ArrayList, use the set() method. The set() method takes two arguments: the index of the element to be set and the new value for the element.

To iterate over the elements of an ArrayList, use a for loop. The for loop can be used to access the elements of the array in order.

To sort the elements of an ArrayList, use the sort() method. The sort() method uses the natural ordering of the elements to sort the array.

To reverse the order of the elements in an ArrayList, use the reverse() method.

To clear the contents of an ArrayList, use the clear() method.

ArrayLists are a powerful and versatile tool for storing and manipulating data. They are easy to use and they offer a wide range of features.

LinkedList

In Java, a linked list is a data structure that consists of a series of nodes, each of which contains a value and a reference to the next node in the list. Linked lists are a type of dynamic data structure, which means that they can be resized at runtime.

Linked lists are created using the LinkedList class. The LinkedList class implements the List interface, which means that it can be used in place of any other list implementation.

To add an element to a linked list, use the add() method. The add() method takes an object as its argument. The object is added to the end of the list.

To remove an element from a linked list, use the remove() method. The remove() method takes an object as its argument. The object is removed from the list.

To get the element at a specific index in a linked list, use the get() method. The get() method takes an integer as its argument. The integer specifies the index of the element to be returned.

To set the element at a specific index in a linked list, use the set() method. The set() method takes two arguments: the index of the element to be set and the new value for the element.

To iterate over the elements of a linked list, use a for loop. The for loop can be used to access the elements of the list in order.

To sort the elements of a linked list, use the sort() method. The sort() method uses the natural ordering of the elements to sort the list.

To reverse the order of the elements in a linked list, use the reverse() method.

To clear the contents of a linked list, use the clear() method.

Linked lists are a powerful and versatile tool for storing and manipulating data. They are easy to use and they offer a wide range of features.

Here are some of the advantages of using linked lists in Java:

- Linked lists are dynamic data structures, which means that they can be resized at runtime.
- Linked lists are easy to implement and use.
- Linked lists are efficient for operations such as insertion and deletion.
- Linked lists are versatile and can be used to store a wide variety of data.

Here are some of the disadvantages of using linked lists in Java:

- Linked lists are not as efficient for operations such as accessing elements by index.
- Linked lists can be more memory-intensive than other data structures, such as arrays.
- Linked lists can be more difficult to debug than other data structures.

Overall, linked lists are a powerful and versatile data structure that can be used to store and manipulate data in Java. However, it is important to be aware of their limitations, such as their memory usage and their potential for difficulty in debugging.

Vector

In Java, a Vector is a data structure that implements the List interface. It is a resizable array, which means that its size can be changed dynamically. Vectors are synchronized, which means that they are thread-safe. This means that multiple threads can access a Vector at the same time without causing any problems.

Vectors are created using the new Vector() constructor. The constructor takes an optional argument that specifies the initial capacity of the array. If the initial capacity is not specified, the default capacity is 10.

To add an element to a Vector, use the add() method. The add() method takes an object as its argument. The object is added to the end of the array.

To remove an element from a Vector, use the remove() method. The remove() method takes an object as its argument. The object is removed from the array.

To get the element at a specific index in a Vector, use the get() method. The get() method takes an integer as its argument. The integer specifies the index of the element to be returned.

To set the element at a specific index in a Vector, use the set() method. The set() method takes two arguments: the index of the element to be set and the new value for the element.

To iterate over the elements of a Vector, use a for loop. The for loop can be used to access the elements of the array in order.

To sort the elements of a Vector, use the sort() method. The sort() method uses the natural ordering of the elements to sort the array.

To reverse the order of the elements in a Vector, use the reverse() method.

To clear the contents of a Vector, use the clear() method.

Vectors are a powerful and versatile tool for storing and manipulating data. They are easy to use and they offer a wide range of features.

Here are some of the advantages of using Vectors in Java:

- Vectors are dynamic data structures, which means that they can be resized at runtime.
- Vectors are synchronized, which means that they are thread-safe.
- Vectors are easy to implement and use.
- Vectors are efficient for operations such as insertion and deletion.
- Vectors are versatile and can be used to store a wide variety of data.

Here are some of the disadvantages of using Vectors in Java:

- Vectors can be more memory-intensive than other data structures, such as ArrayLists.
- Vectors can be more difficult to debug than other data structures.

Overall, Vectors are a powerful and versatile data structure that can be used to store and manipulate data in Java. However, it is important to be aware of their limitations, such as their memory usage and their potential for difficulty in debugging.

HashSet

In Java, the HashSet class is a data structure that implements the Set interface. It is a collection of objects that does not allow duplicate elements. HashSets are created using the new HashSet() constructor. The constructor takes an optional argument that specifies the initial capacity of the set. If the initial capacity is not specified, the default capacity is 16.

To add an element to a HashSet, use the add() method. The add() method takes an object as its argument. The object is added to the set.

To remove an element from a HashSet, use the remove() method. The remove() method takes an object as its argument. The object is removed from the set.

To check if an element is present in a HashSet, use the contains() method. The contains() method takes an object as its argument. The method returns true if the object is present in the set, and false otherwise.

To iterate over the elements of a HashSet, use a for loop. The for loop can be used to access the elements of the set in an undefined order.

To get the size of a HashSet, use the size() method. The size() method returns the number of elements in the set.

To clear the contents of a HashSet, use the clear() method.

HashSets are a powerful and versatile tool for storing and manipulating data. They are easy to use and they offer a wide range of features.

Here are some of the advantages of using HashSets in Java:

- HashSets are efficient for operations such as adding, removing, and checking for the presence of elements.
- HashSets are easy to implement and use.

- HashSets are versatile and can be used to store a wide variety of data.

Here are some of the disadvantages of using HashSets in Java:

- HashSets do not maintain the order of elements.
- HashSets are not synchronized, which means that they are not thread-safe.

Overall, HashSets are a powerful and versatile data structure that can be used to store and manipulate data in Java. However, it is important to be aware of their limitations, such as their lack of ordering and their lack of synchronization.

HashMap

A HashMap is a data structure that stores key-value pairs. It is similar to a HashSet, but it also stores the value associated with each key. HashMaps are created using the new HashMap() constructor. The constructor takes an optional argument that specifies the initial capacity of the map. If the initial capacity is not specified, the default capacity is 16.

To add an element to a HashMap, use the put() method. The put() method takes two arguments: the key and the value. The key is used to retrieve the value later.

To remove an element from a HashMap, use the remove() method. The remove() method takes a key as its argument. The key is used to identify the element to be removed.

To get the value associated with a key in a HashMap, use the get() method. The get() method takes a key as its argument. The method returns the value associated with the key, or null if the key is not present in the map.

To iterate over the elements of a HashMap, use a for loop. The for loop can be used to access the elements of the map in an undefined order.

To get the size of a HashMap, use the size() method. The size() method returns the number of elements in the map.

To clear the contents of a HashMap, use the clear() method.

HashMaps are a powerful and versatile tool for storing and manipulating data. They are easy to use and they offer a wide range of features.

Here are some of the advantages of using HashMaps in Java:

- HashMaps are efficient for operations such as adding, removing, and checking for the presence of elements.

- HashMaps are easy to implement and use.
- HashMaps are versatile and can be used to store a wide variety of data.

Here are some of the disadvantages of using HashMaps in Java:

- HashMaps do not maintain the order of elements.
- HashMaps are not synchronized, which means that they are not thread-safe.

Overall, HashMaps are a powerful and versatile data structure that can be used to store and manipulate data in Java. However, it is important to be aware of their limitations, such as their lack of ordering and their lack of synchronization.

ArrayDeque

In Java, an ArrayDeque is a data structure that implements the Deque interface. It is a double-ended queue, which means that elements can be added or removed from either the front or the back of the queue. ArrayDeques are created using the new ArrayDeque() constructor. The constructor takes an optional argument that specifies the initial capacity of the deque. If the initial capacity is not specified, the default capacity is 16.

To add an element to the front of an ArrayDeque, use the addFirst() method. The addFirst() method takes an object as its argument. The object is added to the front of the deque.

To add an element to the back of an ArrayDeque, use the addLast() method. The addLast() method takes an object as its argument. The object is added to the back of the deque.

To remove an element from the front of an ArrayDeque, use the removeFirst() method. The removeFirst() method does not return anything.

To remove an element from the back of an ArrayDeque, use the removeLast() method. The removeLast() method does not return anything.

To get the element at the front of an ArrayDeque, use the getFirst() method. The getFirst() method returns the object at the front of the deque.

To get the element at the back of an ArrayDeque, use the getLast() method. The getLast() method returns the object at the back of the deque.

To iterate over the elements of an ArrayDeque, use a for loop. The for loop can be used to access the elements of the deque in either order.

To get the size of an ArrayDeque, use the size() method. The size() method returns the number of elements in the deque.

To clear the contents of an ArrayDeque, use the clear() method.

ArrayDeques are a powerful and versatile tool for storing and manipulating data. They are easy to use and they offer a wide range of features.

Here are some of the advantages of using ArrayDeques in Java:

- ArrayDeques are efficient for operations such as adding, removing, and getting elements.
- ArrayDeques are easy to implement and use.
- ArrayDeques are versatile and can be used to store a wide variety of data.

Here are some of the disadvantages of using ArrayDeques in Java:

- ArrayDeques are not synchronized, which means that they are not thread-safe.
- ArrayDeques can be more memory-intensive than other data structures, such as ArrayLists.
- ArrayDeques can be more difficult to debug than other data structures.

Overall, ArrayDeques are a powerful and versatile data structure that can be used to store and manipulate data in Java. However, it is important to be aware of their limitations, such as their lack of synchronization and their potential for difficulty in debugging.

PriorityQueue

A PriorityQueue is a data structure in Java that implements the Queue interface. It is a collection of objects that are sorted according to their priority. The priority of an object is determined by the Comparable or Comparator object that is passed to the PriorityQueue constructor.

PriorityQueues are created using the new PriorityQueue() constructor. The constructor takes an optional argument that specifies the initial capacity of the queue. If the initial capacity is not specified, the default capacity is 11.

To add an element to a PriorityQueue, use the add() method. The add() method takes an object as its argument. The object is added to the queue in sorted order, according to its priority.

To remove an element from a PriorityQueue, use the remove() method. The remove() method does not return anything.

To get the element at the head of a PriorityQueue, use the peek() method. The peek() method returns the object at the head of the queue, without removing it.

To remove and get the element at the head of a PriorityQueue, use the poll() method. The poll() method returns the object at the head of the queue and removes it from the queue.

To iterate over the elements of a PriorityQueue, use a for loop. The for loop can be used to access the elements of the queue in sorted order, according to their priority.

To get the size of a PriorityQueue, use the size() method. The size() method returns the number of elements in the queue.

To clear the contents of a PriorityQueue, use the clear() method.

PriorityQueues are a powerful and versatile tool for storing and manipulating data. They are easy to use and they offer a wide range of features.

Here are some of the advantages of using PriorityQueues in Java:

- PriorityQueues are efficient for operations such as adding, removing, and getting elements.
- PriorityQueues are easy to implement and use.
- PriorityQueues are versatile and can be used to store a wide variety of data.

Here are some of the disadvantages of using PriorityQueues in Java:

- PriorityQueues are not synchronized, which means that they are not thread-safe.
- PriorityQueues can be more memory-intensive than other data structures, such as ArrayLists.
- PriorityQueues can be more difficult to debug than other data structures.

Overall, PriorityQueues are a powerful and versatile data structure that can be used to store and manipulate data in Java. However, it is important to be aware of their limitations, such as their lack of synchronization and their potential for difficulty in debugging.

Stack

A stack is a linear data structure that follows the Last In First Out (LIFO) principle. This means that the last element added to the stack is the first element to be removed. Stacks are often used in recursion and backtracking.

In Java, the Stack class is used to represent a stack. The Stack class is a subclass of the Vector class. The Stack class has the following methods:

- push(object): Adds an object to the top of the stack.
- pop(): Removes the object from the top of the stack and returns it.
- peek(): Returns the object at the top of the stack without removing it.
- isEmpty(): Returns true if the stack is empty.
- size(): Returns the number of elements in the stack.

Conclusion

Arrays and collections are fundamental data structures in Java that are essential for developing complex software systems. In this chapter, we have covered the fundamental concepts of arrays and collections in Java, including the differences between arrays and collections, the most commonly used collection classes in Java, and their features. Understanding these concepts is crucial for effectively using arrays and collections in your Java programs.