2

Arrays Lists and Stack

Objectives

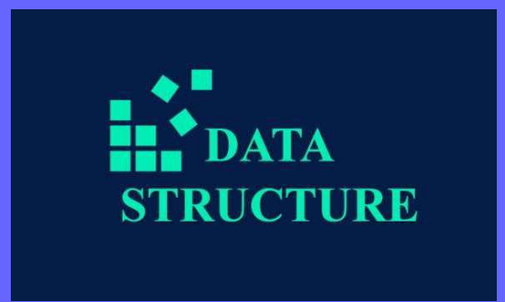
After completing this lesson, you should be able to:

- Arrays
- List
- Stack

Course Roadmap

Data Structures

- Lesson 1: Introduction to Data Structures
- **Lesson 2: Lists and Stacks** — **You are here!**
- Lesson 3: Searching
- Lesson 4: Trees and Queues
- Lesson 5: Sorting

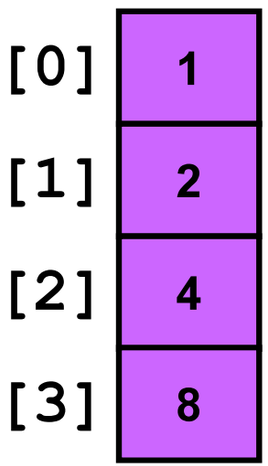


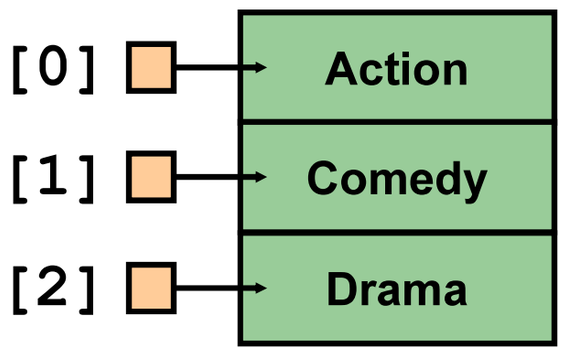
Introduction to Arrays

Arrays

An array is a collection of variables of the same type.

- Each element can hold a single item.
- Items can be primitives or object references.
- The length of the array is fixed when it is created.

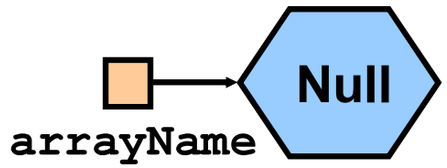




Creating an Array of Primitives

1. Declare the array.

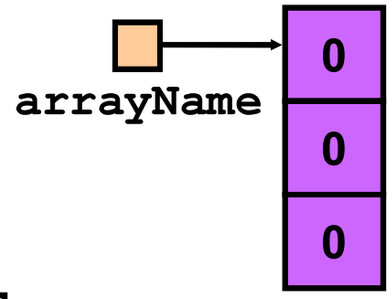
```
type[] arrayName;
… or …
type arrayName[];
```



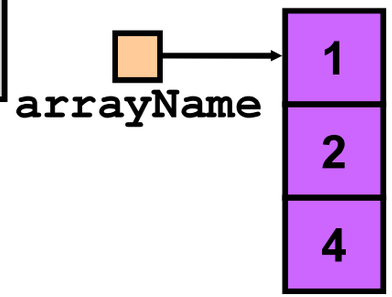
`type` is a primitive, such as `int` and so on.

2. Create the array object.

```
// Create array object syntax
arrayName = new type[size];
```



3. Initialize the array elements (optional).

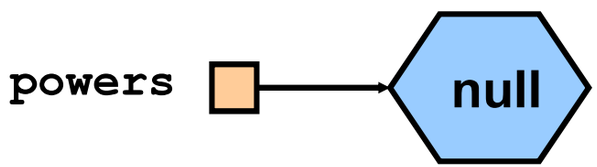


Declaring an Array

- Create a variable to reference the array object:

```
int[] powers;  // Example
```

- When an array variable is declared:
  - Its instance variable is initialized to `null` until the array object has been created



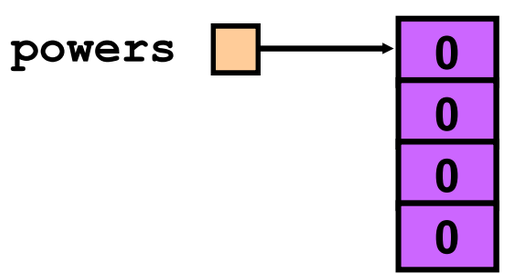
  - Its method variable is unknown until the object is created

Creating an Array Object

- Create an array of the required length and assign it to the array variable:

```
int[] powers;          // Declare array variable
powers = new int[4];   //Create array object
```

  - Create the array object by using the `new` operator.
- The contents of an array of primitives are initialized automatically.

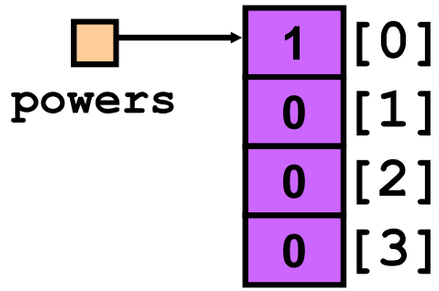


Initializing Array Elements

- Assign values to individual elements:

```
arrayName[index] = value;
```

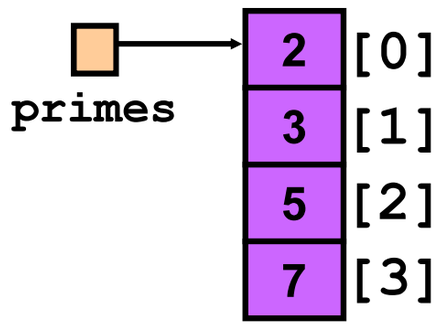
```
powers[0] = 1;
```



- Create and initialize arrays at the same time:

```
type[] arrayName = {valueList};
```

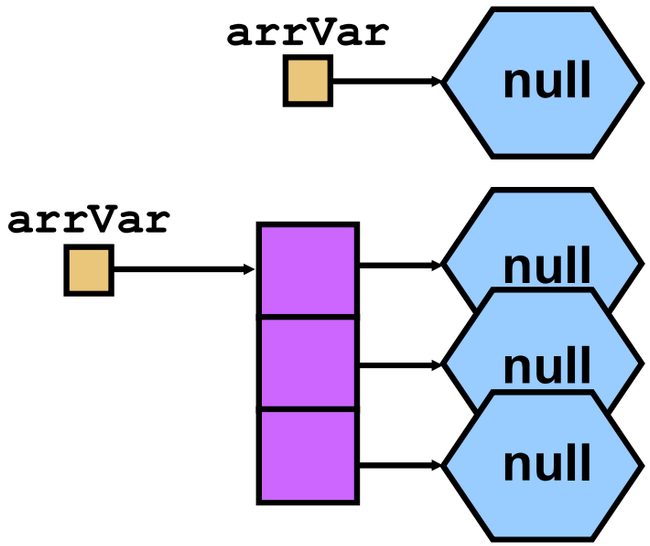
```
int[] primes = {2, 3, 5, 7};
```



Creating an Array of Object References

1. Declare the array.

```
ClassName[] arrVar;
… or …
ClassName arrVar[];
```



2. Create the array object.

```
// Create array object syntax
arrVar = new ClassName[size];
```


3. Initialize the objects in the array.

Initializing the Objects in an Array

- Assign a value to each array element:

```
// Create an array of four empty Strings
String[] arr = new String[4];
for (int i = 0; i < arr.length; i++) {
     arr[i] = new String();
}
```

- Create and initialize the array at the same time:

```
String[] categories =
           {"Action", "Comedy", "Drama"};
```

Using an Array of Object References

- Any element can be assigned to an object of the correct type:

```
String category = categories[0];
```

- Each element can be treated as an individual object:

```
System.out.println
      ("Length is " + categories[2].length());
```

- An array element can be passed to any method; array elements are passed by reference.

Going Through the Array Elements

- Use a loop to explore each element in the array:

```
for (int i = 0;i < categories.length; i++){
   System.out.println("Category: "+categories[i]);
}
```

- Java 5.0 provides this alternative enhanced syntax:

```
for (String category: categories) {
     System.out.println ("Category: " +category);
}
```

Arrays and Exceptions

- `ArrayIndexOutOfBoundsException` occurs when an array index is invalid:

```
String[] list = new String[4];
//The following throws ArrayIndexOutOfBoundsException
System.out.println(list[4]);
```

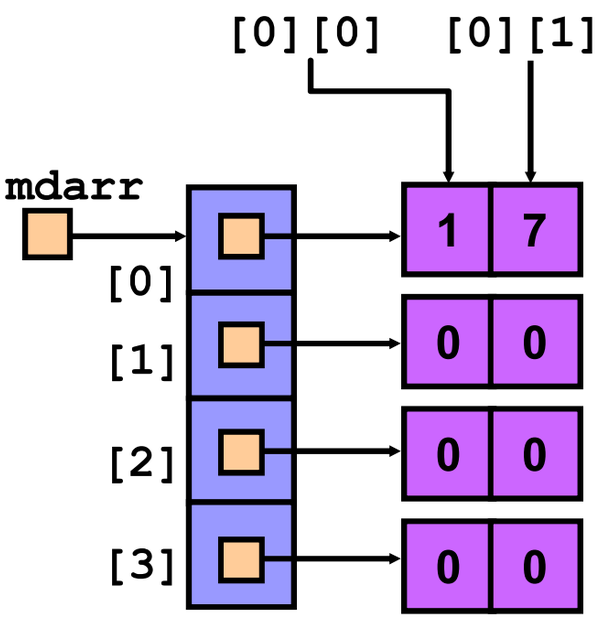
- `NullPointerException` occurs when you try to access an element that has not been initialized:

```
Movie[] movieList = new Movie[3];
// The following will throw NullPointerException
String director = movieList[0].getDirector();
```

Java supports arrays of arrays:

```
type[][] arrayname = new type[n1][n2];
```

```
int[][] mdarr = new int[4][2];
mdarr[0][0] = 1;
mdarr[0][1] = 7;
```



Passing Command-Line Parameters

- `main()` has a single parameter: `args`.
- `args` is an array of `Strings` that holds command-line parameters:

```
C:\> java SayHello Hello World
```

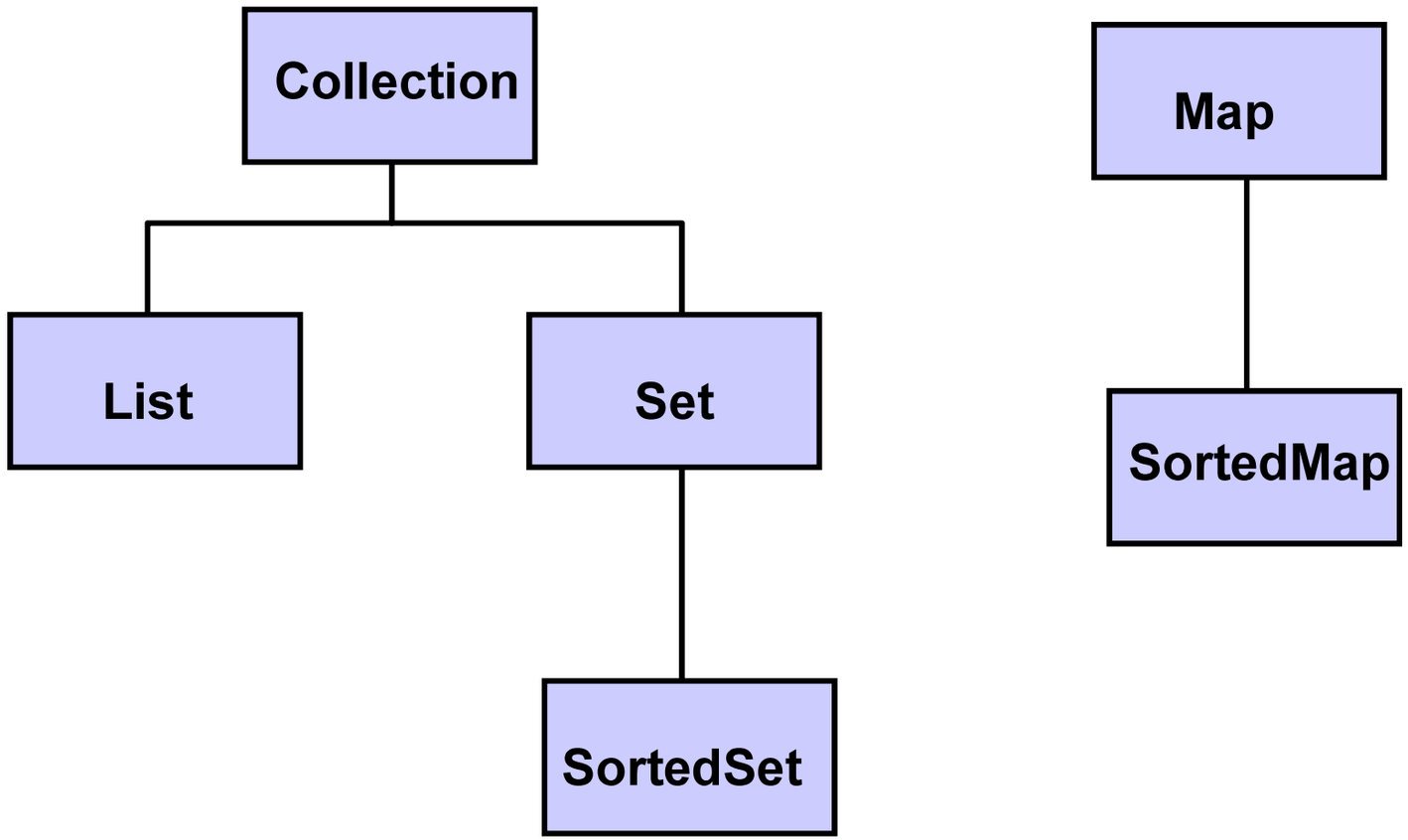
```
public class SayHello {
  public static void main(String[] args) {
    if (args.length != 2)
     System.out.println("Specify 2 arguments");
    else
      System.out.println(args[0]+" "+args[1]);
  } …
```

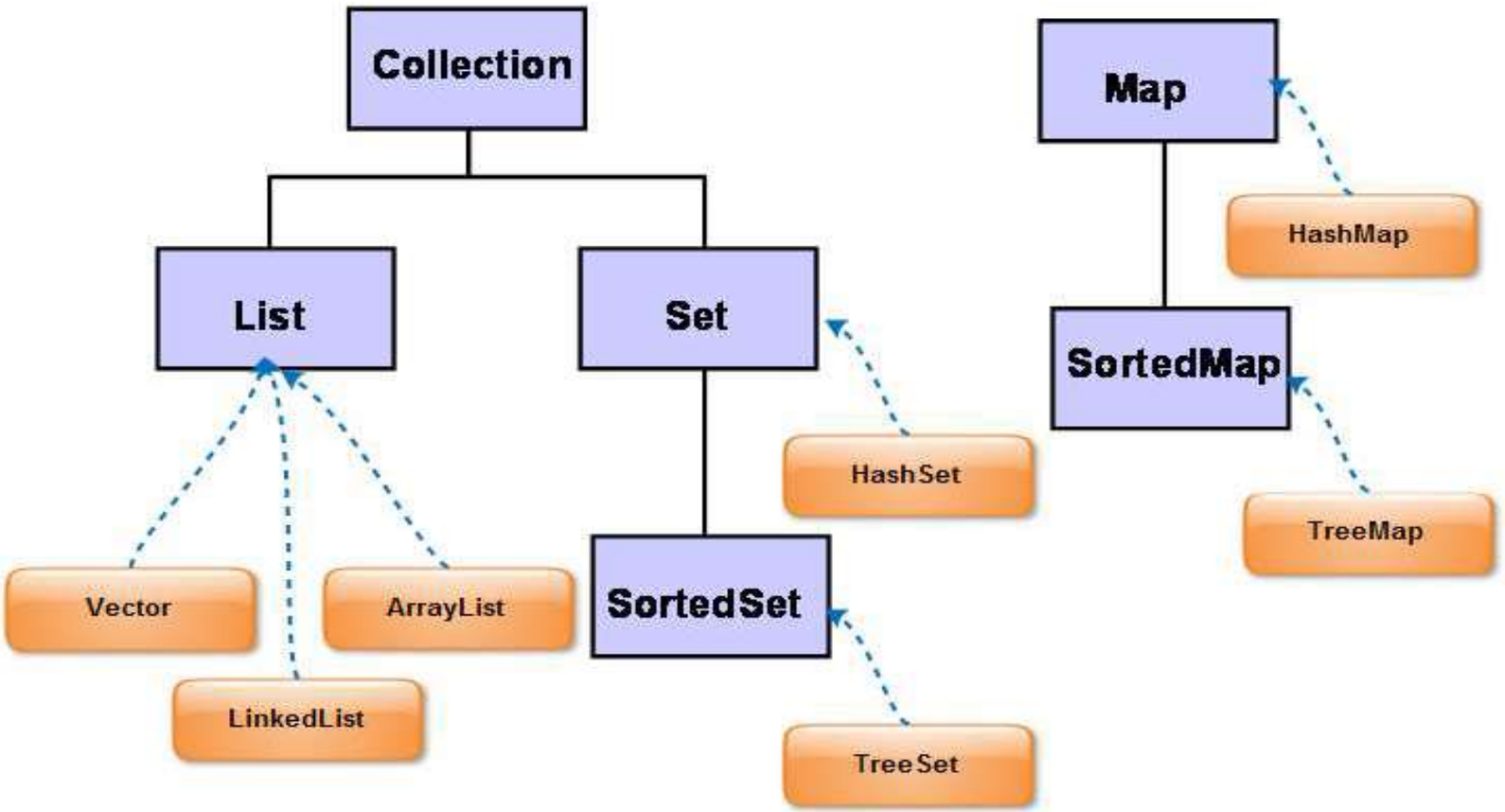
Java Collections Framework

Java Collections Framework is an API architecture for managing a group of objects that can be manipulated independently of their internal implementation. It is:

- Found in the `java.util` package
- Defined by six core interfaces and many implementation classes:
  - `Collection` interface: Generic group of elements
  - `Set` interface: Group of unique elements
  - `List` interface: Ordered group of elements
  - `Map` interface: Group of unique keys and their values
  - `SortedSet` and `SortedMap` for a sorted `Set` and `Map`

Framework Interface Hierarchy




Collection
List
Set
Map
HashMap
SortedMap
HashSet
TreeMap
Vector
ArrayList
SortedSet
LinkedList
TreeSet

Collections Framework Components

Collections Framework is a set of interfaces and classes used to store and manipulate groups of data as a single unit.

- Core interfaces are the interfaces used to manipulate collections and to pass them from one method to another.
- Implementations are the actual data objects used to store collections, which implement the core collection interface.
- Algorithms are pieces of reusable functionality provided by the JDK.

`List` Interface

- `List` defines generic list behavior.
  - Is an ordered collection of elements
- `List` behaviors include:
  - Adding elements at a specific index
  - Getting an element based on an index
  - Removing an element based on an index
  - Overwriting an element based on an index
  - Getting the size of the list
- `List` allows duplicate elements.

- Is an implementation of the `List` interface
  - The list automatically grows if elements exceed initial size.
- Has a numeric index
  - Elements are accessed by index.
  - Elements can be inserted based on index.
  - Elements can be overwritten.
- Allows duplicate items

```
List<Integer> partList = new ArrayList<>(3);
    partList.add(new Integer(1111));
    partList.add(new Integer(2222));
    partList.add(new Integer(3333));
    partList.add(new Integer(4444)); // ArrayList auto grows
    System.out.println("First Part: " + partList.get(0)); // First item
    partList.add(0, new Integer(5555)); // Insert an item by index
```

The `ArrayList` class:

- Is a resizable implementation of the `List` interface
- Allows manipulation of the array size
- Has capacity that grows as elements are added to the list
- Creating an empty ArrayList:

```
ArrayList members = new ArrayList();
```

- Creating an ArrayList with an initial size:

```
// Create an ArrayList with 10 elements.
ArrayList members = new ArrayList(10);
```

Modifying an ArrayList

- Add an element to the end of the ArrayList:

```
String name = MyMovie.getNextName();
members.add(name);
```

- Add an element at a specific position:

```
// Insert a string at the beginning
members.add(0, name);
```

- Remove the element at a specific index:

```
// Remove the first element
members.remove(0);
```

Accessing an ArrayList

- Get the first element:

```
String s = members.get(0);
```

- Get an element at a specific position:

```
String s = members.get(2);
```

- Find an object in an ArrayList:

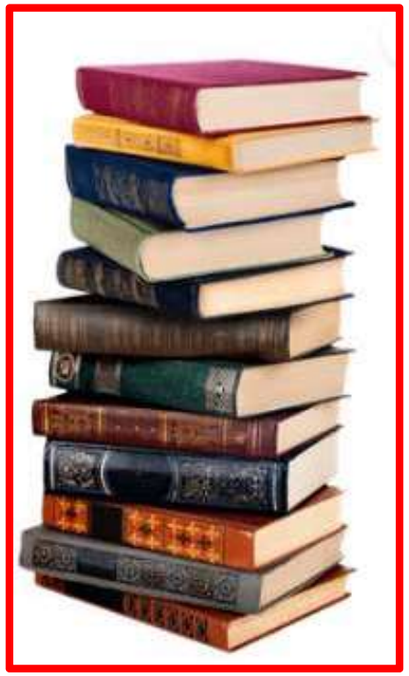
```
int position = members.indexOf(name);
```

- Get the size of an ArrayList :

```
int size = members.size();
```

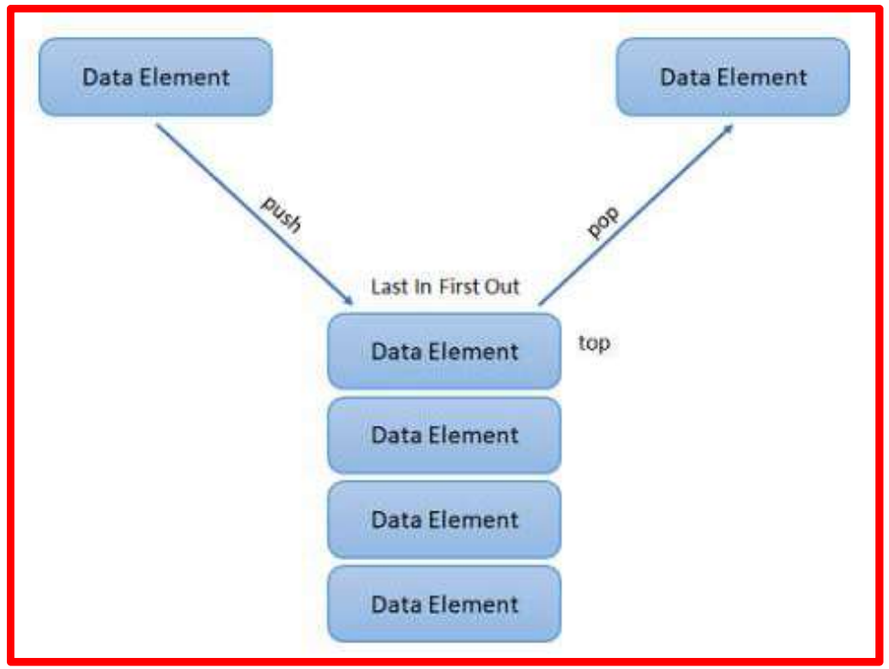
3. Stacks

- A **Stack** is a Linear Data Structure that follows the **LIFO** (Last In, First Out) principle that allows operations like insertion and deletion from one end of the Stack, i.e., Top.
- Stacks can be implemented with the help of contiguous memory, an Array, and non-contiguous memory, a Linked List. Real-life examples of Stacks are piles of books, a deck of cards, piles of money, and many more.



The primary operations in the Stack are as follows:

- **Push:** Operation to insert a new element in the Stack is termed as Push Operation.
- **Pop:** Operation to remove or delete elements from the Stack is termed as Pop Operation.

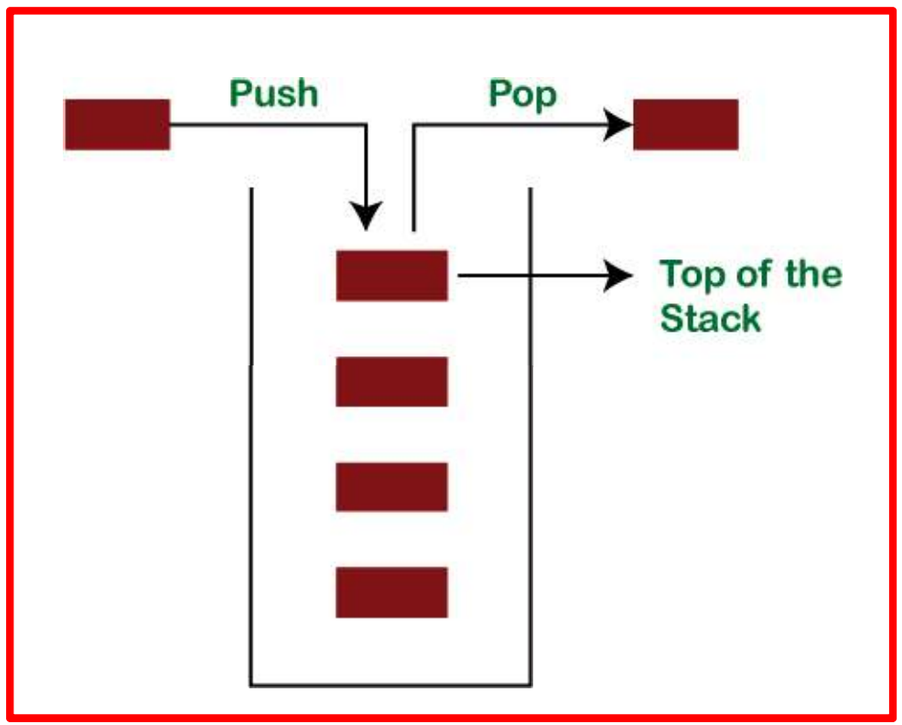


Some Applications of Stacks:

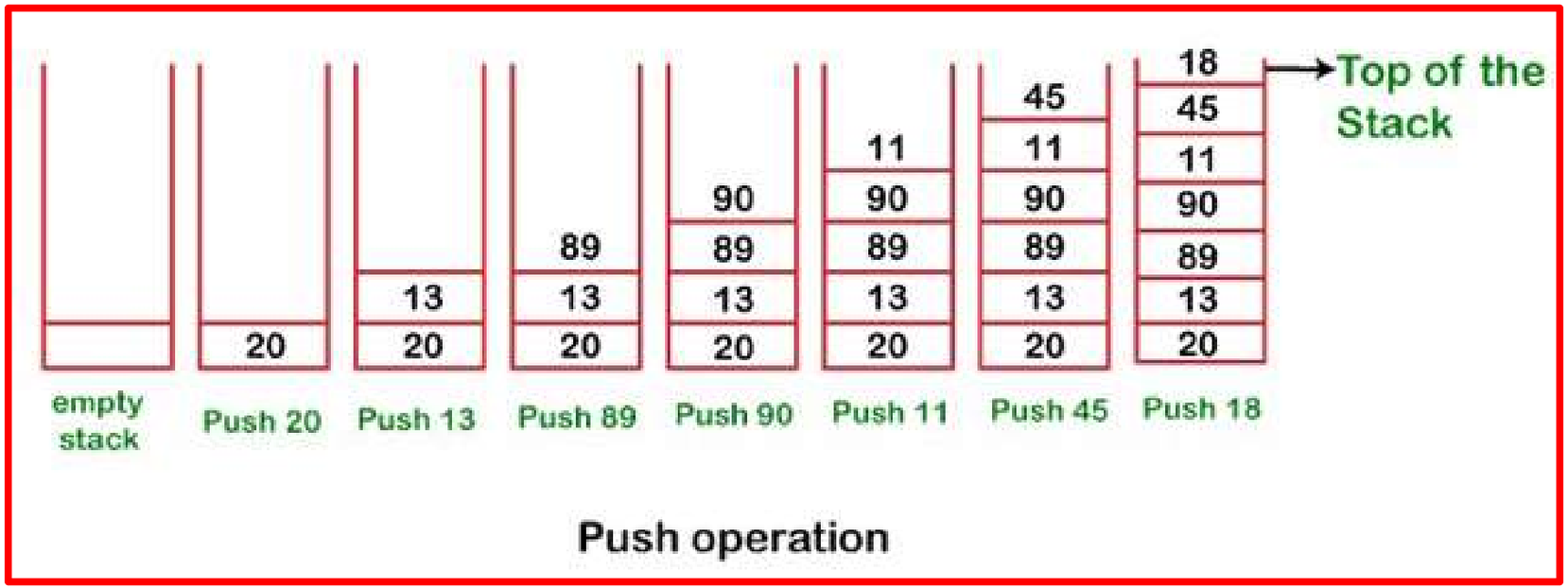
1. The Stack is used as a Temporary Storage Structure for recursive operations.
2. Stack is also utilized as Auxiliary Storage Structure for function calls, nested operations, and deferred/postponed functions.
3. We can manage function calls using Stacks.
4. Stacks are also utilized to evaluate the arithmetic expressions in different programming languages.
5. Stacks are also helpful in converting infix expressions to postfix expressions.
6. Stacks allow us to check the expression's syntax in the programming environment.
7. We can match parenthesis using Stacks.

8. Stacks can be used to reverse a String.
9. Stacks are helpful in solving problems based on backtracking.
10. We can use Stacks in depth-first search in graph and tree traversal.
11. Stacks are also used in Operating System functions.
12. Stacks are also used in UNDO and REDO functions in an edit.

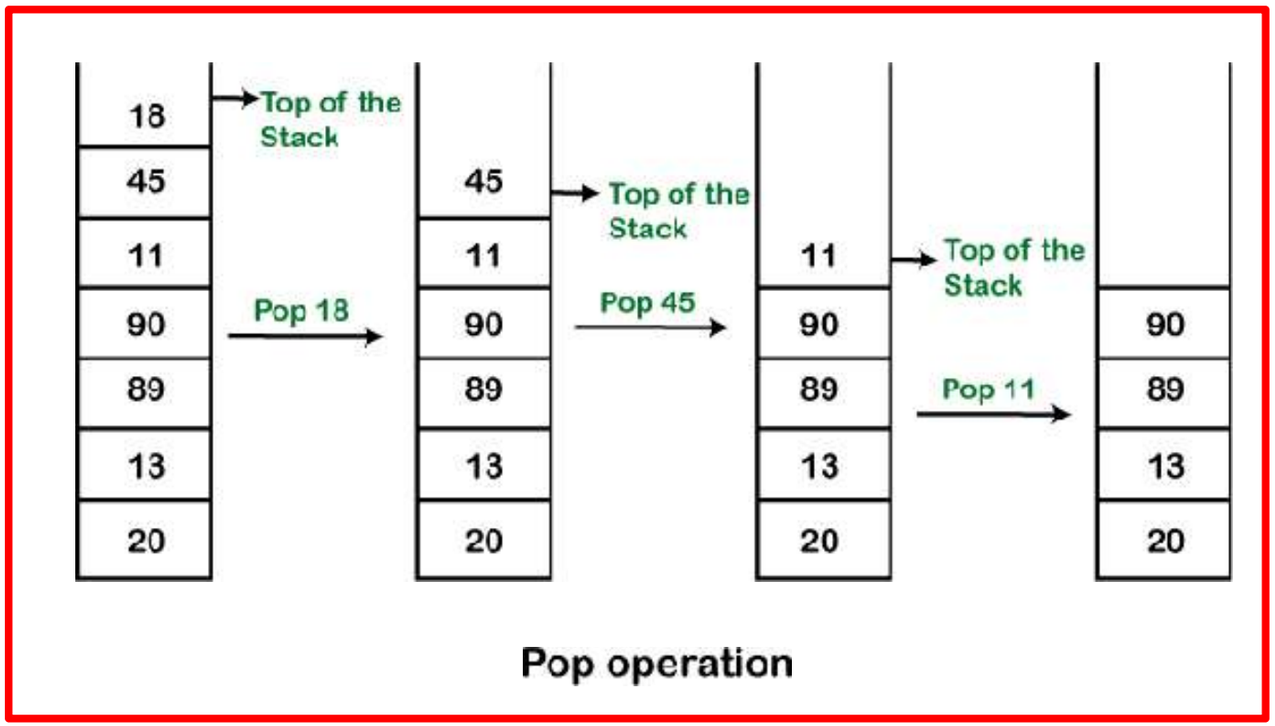
- The stack data structure has the two most important operations that are **push** and **pop**.
- The push operation inserts an element into the stack and pop operation removes an element from the top of the stack.



- Let's push 20, 13, 89, 90, 11, 45, 18, respectively into the stack.



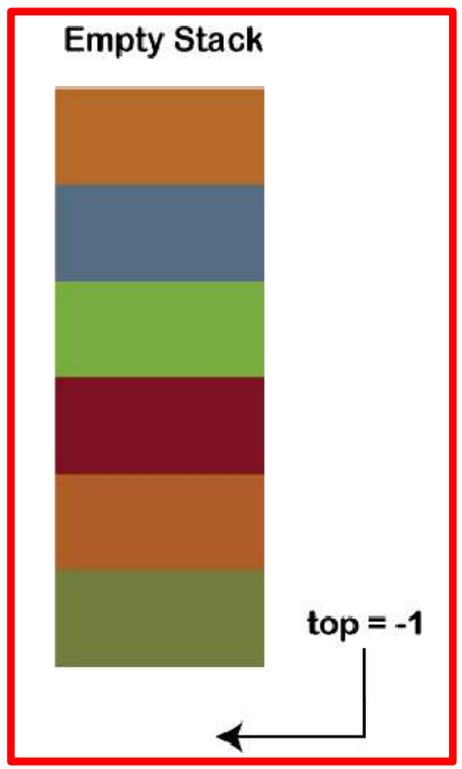
- Let's remove (pop) 18, 45, and 11 from the stack.



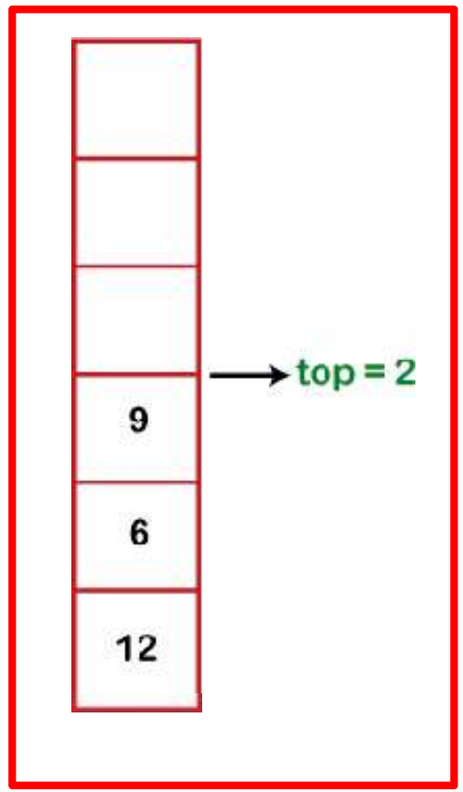
Pop operation

Empty Stack

- **Empty Stack:** If the stack has no element is known as an **empty stack**. When the stack is empty the value of the top variable is -1.

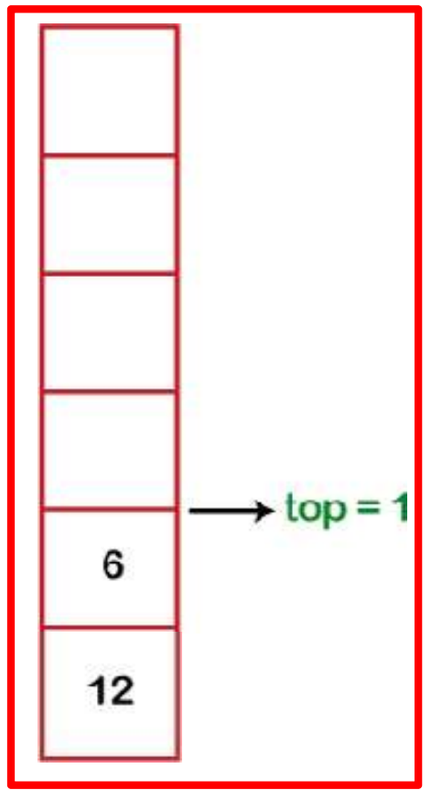


- When we push an element into the stack the top is **increased by 1**. In the following figure,
  - Push 12, top=0
  - Push 6, top=1
  - Push 9, top=2

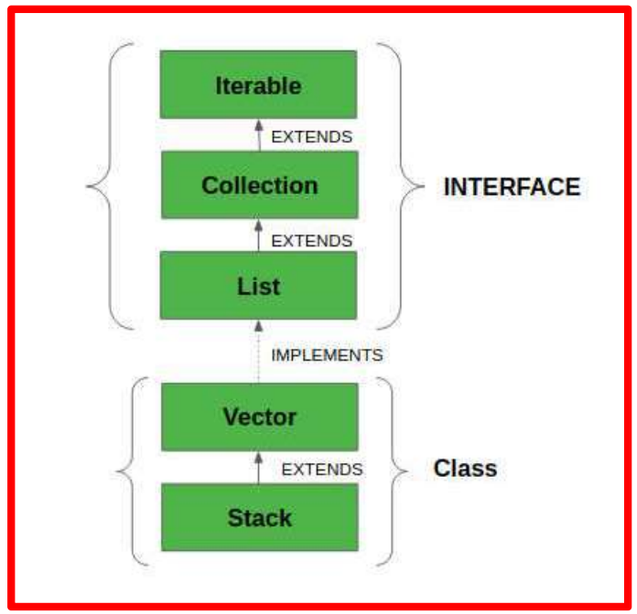


- When we pop an element from the stack the value of top is **decreased by 1**. In the following figure, we have popped 9.

Top value	Meaning
-1	It shows the stack is empty.
0	The stack has only an element.
N-1	The stack is full.
N	The stack is overflow.



- Java Collection framework provides a Stack class that models and implements a Stack data structure. The class is based on the basic principle of last-in-first-out. In addition to the basic push and pop operations, the class provides three more functions of empty, search, and peek.
- The class can also be said to extend Vector and treats the class as a stack with the five mentioned functions. The class can also be referred to as the subclass of Vector.



How to Create a Stack?

- In order to create a stack, we must import **java.util.stack** package and use the Stack() constructor of this class.

```
Stack<E> stack = new Stack<E>();
```

- Here E is the type of Object.

Methods of the Stack Class

Method	Modifier and Type	Method Description
empty()	boolean	The method checks the stack is empty or not.
push(E item)	E	The method pushes (insert) an element onto the top of the stack.
pop()	E	The method removes an element from the top of the stack and returns the same element as the value of that function.
peek()	E	The method looks at the top element of the stack without removing it.
search(Object o)	int	The method searches the specified object and returns the position of the object.

Sample

```
 */
package training.iqgateway;
import java.util.*;

/**
 *
 * @author Sai Baba
 */
public class StackExample {

    public static void main(String[] args) {

        Stack<Integer> integerStackRef = new Stack<>();
        boolean isEmpty = integerStackRef.empty();
        System.out.println("Is Stack Empty " + isEmpty );
```

```
// Lets Add Elements to the Stack
integerStackRef.push(10);
integerStackRef.push(20);
integerStackRef.push(30);
integerStackRef.push(40);
integerStackRef.push(50);

// Get the Content From Stack
System.out.println("Elements Part of the Stack : " + integerStackRef);
boolean isEmpty1 = integerStackRef.empty();
System.out.println("Is Stack Empty " + isEmpty1 );

// Removing Elements
int removedElement = integerStackRef.pop();
System.out.println("Elements Part of the Stack : " + integerStackRef);
System.out.println("Removed Element is : " + removedElement);
```

```
        int removedElementAtIndex = integerStackRef.remove(2);
        System.out.println("Elements Part of the Stack : " + integerStackRef);
        System.out.println("Removed Element is : " + removedElementAtIndex);

        int topElement = integerStackRef.peek();
        System.out.println("Elements Part of the Stack : " + integerStackRef);
        System.out.println("Top Element is : " + topElement);

        // Searching
        int position = integerStackRef.search(20);
        System.out.println("Positio of Element is : " + position);

        // Iterate The Stack [ Loops Using Enumeration /Iterator ]
        // But We Can Even Use forEach [ Lambdas ]
        integerStackRef.forEach(data -> System.out.println(data));
    }
}
```

Summary

In this lesson, you should have learned that:

- Arrays
- List
- Stack