To implement the clone() method for an ArrayStack class, we need to ensure that we create a copy of the stack, including all its elements. A clone() method for a stack typically involves copying the stack's internal array (or backing array), as well as copying the current size (i.e., how many elements are in the stack).

Here is an example of how to implement the clone() method for an ArrayStack class:

**Example Java Code for ArrayStack with clone() method:**

import java.util.Arrays;

public class ArrayStack<T> implements Cloneable {

private T[] stackArray;

private int size;

private static final int INITIAL\_CAPACITY = 10;

// Constructor to initialize the stack with a default capacity

public ArrayStack() {

stackArray = (T[]) new Object[INITIAL\_CAPACITY];

size = 0;

}

// Push method to add an element to the stack

public void push(T item) {

if (size == stackArray.length) {

resize(stackArray.length \* 2);

}

stackArray[size++] = item;

}

// Pop method to remove and return the top element of the stack

public T pop() {

if (isEmpty()) throw new IllegalStateException("Stack is empty");

T item = stackArray[--size];

stackArray[size] = null; // Avoid loitering (help GC)

return item;

}

// Peek method to return the top element without removing it

public T peek() {

if (isEmpty()) throw new IllegalStateException("Stack is empty");

return stackArray[size - 1];

}

// Method to check if the stack is empty

public boolean isEmpty() {

return size == 0;

}

// Method to get the current size of the stack

public int size() {

return size;

}

// Method to resize the stack's backing array

private void resize(int capacity) {

stackArray = Arrays.copyOf(stackArray, capacity);

}

// Clone method to create a copy of the stack

@Override

public ArrayStack<T> clone() {

try {

// Call Object's clone method to create a shallow copy

ArrayStack<T> clone = (ArrayStack<T>) super.clone();

// Deep copy the stack's array and set the size

clone.stackArray = Arrays.copyOf(this.stackArray, this.stackArray.length);

clone.size = this.size;

return clone;

} catch (CloneNotSupportedException e) {

throw new RuntimeException("Cloning not supported", e);

}

}

// Main method to test cloning

public static void main(String[] args) {

ArrayStack<Integer> originalStack = new ArrayStack<>();

originalStack.push(1);

originalStack.push(2);

originalStack.push(3);

System.out.println("Original Stack: ");

System.out.println("Top element: " + originalStack.peek());

// Clone the stack

ArrayStack<Integer> clonedStack = originalStack.clone();

// Display the cloned stack's content

System.out.println("Cloned Stack: ");

System.out.println("Top element: " + clonedStack.peek());

// Modify the original stack to verify the cloned stack remains unaffected

originalStack.pop();

System.out.println("Original Stack after pop: ");

System.out.println("Top element: " + originalStack.peek());

System.out.println("Cloned Stack after original stack pop: ");

System.out.println("Top element: " + clonedStack.peek());

}

}

### Explanation:

1. **ArrayStack Class**:
   * This class implements a basic stack with methods like push, pop, peek, and isEmpty.
   * It uses an array (stackArray) to store the elements and has a size variable to track the number of elements in the stack.
   * The resize() method is used to dynamically resize the stack when the array is full.
2. **clone() method**:
   * The clone() method is overridden from the Object class and uses super.clone() to create a shallow copy of the ArrayStack object.
   * It then deep copies the stackArray using Arrays.copyOf(), which ensures that the clone gets a new array that is separate from the original stack.
   * The size field is also copied directly to maintain the correct number of elements in the cloned stack.
3. **Testing**:
   * In the main method, the stack is populated with a few elements.
   * A clone of the stack is created using the clone() method, and both the original and cloned stacks are tested to ensure they behave as expected.
   * The original stack is modified with a pop() operation, and the cloned stack remains unaffected, showing that they are independent.

Original Stack:

Top element: 3

Cloned Stack:

Top element: 3

Original Stack after pop:

Top element: 2

Cloned Stack after original stack pop:

Top element: 3

**Key Points:**

* **Shallow Copy**: When you call super.clone(), it creates a shallow copy of the object, meaning it copies the reference to the stackArray, not the array itself.
* **Deep Copy**: We create a deep copy of the stackArray by using Arrays.copyOf() to ensure that the cloned stack has its own separate array.
* This approach ensures that modifying the original stack does not affect the cloned stack, and vice versa.