Stack (Implementation)

How is Stack implemented in Java? What are some of the main methods that each stack have? That is what we are going to cover in this lesson.

Introduction

Every programming language comes with the basic functionality of Stack. In Java, you can use the pre-built class of Stack by importing it to your program. However, you can manually implement a stack and extend its functionality for your use.

Stacks can either be implemented using arrays or linked lists. It has yet to be concluded which implementation is more efficient, as both data structures offer different variations of Stack. However, stacks are generally implemented using arrays because it takes less space; we don't need to store an additional pointer like in a linked list.

Implementation

As we discussed in the previous lesson, a typical Stack must contain the following standard methods:

- `push (datatype V)`
- `datatype pop()`
- `boolean isEmpty()`
- `datatype top()`

Before we take a look at these methods one by one, let's construct a class of Stack, and create an instance. This class has the following three data members:

and create an instance. This class has the following three data members:

- the array that will hold all the elements
- the size of this array
- a variable for the `top` element of Stack.

The following code shows how to construct the `Stack` class:

main.java

Stack.java

```java
public class Stack <V> {
    private int maxSize;
    private int top;
    private V arr[];

    /*
    Java does not allow generic type arrays. So we have used an
    array of Object type and type-casted it to the generic type V.
    This type-casting is unsafe and produces a warning.
    Comment out the line below and execute again to see the warning.
    */
    @SuppressWarnings("unchecked")
    public Stack(int max_size) {
        this.maxSize = max_size;
        this.top = -1; //initially when stack is empty
        arr = (V[]) new Object[max_size];//type casting Object[] to V[]
    }
    public int getCapacity() {
        return maxSize;
    }

}

```

Now before adding the `push` and `pop` methods into this code, we need to implement some helper methods to keep the code simple and reusable. Here's a list of the helper methods that we will implement in the code below:

- `isEmpty()`
- `isFull()`

- isFull()
- top()

Here is the code for stacks with the new helper methods. Try experimenting with them!

main.java

Stack.java

```java
public class Stack <V> {
    private int maxSize;
    private int top;
    private V array[];

    /*
    Java does not allow generic type arrays. So we have used an
    array of Object type and type-casted it to the generic type V.
    This type-casting is unsafe and produces a warning.
    Comment out the line below and execute again to see the warning.
    */
    @SuppressWarnings("unchecked")
    public Stack(int max_size) {
        this.maxSize = max_size;
        this.top = -1; //initially when stack is empty
        array = (V[]) new Object[max_size];//type casting Object[] to V[]
    }

    //returns the maximum size capacity
    public int getMaxSize() {
        return maxSize;
    }

    //returns true if Stack is empty
    public boolean isEmpty(){
        return top == -1;
    }

    //returns true if Stack is full
    public boolean isFull(){
        return top == maxSize -1;
    }

    //returns the value at top of Stack
    public V top(){
        if(isEmpty())
            return null;
        return array[top];
    }
}
```

If your output returns `true` for `isEmpty()` and `false` for `isFull()`, it means that these helper functions are working properly! Now, take a look at this extended code with `push` and `pop` functions added to the definition of Stack class. We will try to add and remove some elements from this stack by using these two functions. Let’s get to it!

main.java

Stack.java

```java
public class Stack <V> {
    private int maxSize;
    private int top;
    private V array[];

    /*
    Java does not allow generic type arrays. So we have used an
    array of Object type and type-casted it to the generic type V.
    This type-casting is unsafe and produces a warning.
    Comment out the line below and execute again to see the warning.
    */
    @SuppressWarnings("unchecked")
    public Stack(int max_size) {
        this.maxSize = max_size;
        this.top = -1; //initially when stack is empty
        array = (V[]) new Object[max_size];//type casting Object[] to V[]
    }

    //returns the maximum size capacity
    public int getMaxSize() {
        return maxSize;
    }

    //returns true if Stack is empty
    public boolean isEmpty(){
        return top == -1;
    }

    //returns true if Stack is full
    public boolean isFull(){
        return top == maxSize -1;
    }

    //returns the value at top of Stack
    public V top(){
        if(isEmpty())
            return null;
        return array[top];
    }

    //inserts a value to the top of Stack
    public void push(V value){
        if(isFull()) {
            System.err.println("Stack is Full!");
            return;
        }
        array[++top] = value; //increments the top and adds value to updated top
    }

    //removes a value from top of Stack and returns
    public V pop(){
        if(isEmpty())
            return null;
        return array[top--]; //returns value and top and decrements the top
    }

}

```

If you look at the output of the code, you can see that the elements popped out of the stack in the exact reverse order as they were pushed in. That means our Stack works perfectly.

Complexities of Stack Operations

Let’s look at the time complexity of each stack operation.

Operation	Time Complexity
isEmpty	O(1)
top	O(1)
size	O(1)
push	O(1)
pop	O(1)

The next data structure that we are going to cover is a **Queue**, which is almost similar to Stack with minor variations.

Back
What is a Stack?

Next
What is a Queue?

✓ Mark as Completed