Lecture

NOTE: FOR FURTHER DETAILS AND MORE COMPREHENSIVE STUDY, PLEASE SEE RECOMMENDED BOOKS OR INTERNET.

Stack

A ***Stack*** is a collection of objects inserted and removed according to the Last In First Out (***LIFO***) principle.

Stack is data structure used to store the data in such a way that first element inserted into the stack will be removed at last. Just take real time example, suppose we have created stack of the book like this shown in the following figure:

Stack Concept

Stack Concept

How Books are arranged in Stack?

1. Books are kept one above the other.
2. Book which is inserted first is taken out at last. **(Brown)**
3. Book which is inserted lastl is served first. **(Light Green)**

Common Example:

Suppose at your home you have multiple chairs then you put them together to form a vertical pile. From that vertical pile the chair which is placed last is always removed first. Chair which was placed first is removed last. In this way we can see how stack is related to us.

Stack ADT

The abstract data type (ADT) is special kind of data type, whose behavior is defined by a ***set of values*** or ***set of operations*** and sometimes both. The keyword ***"Abstract"*** is used as we can perform different operations. But how those operations are working that is totally hidden from the user. The ADT is made with primitive data types but their primitive operation's logics are dependent on user implementation.

Primitive Operations

The primitive operations for a stack are defined as:

Operation	Description
`isEmpty()`	Determine whether the stack is empty, if stack is empty returns ***true*** else returns ***false***
`isFull()`	Determine whether the stack is full, if stack is full returns ***true*** else returns ***false***
`push()`	Add an item to the stack, the item that is added will stay at the top.
`pop()`	Remove and return the item from the stack, the item that is return is the item that is on the top or most recently added item.
`peek()`	Retrieve or returns the item most recently added or in other words the item that is on the top.
`size()`	Returns the number of items that are available in the stack.

Representation of Stack using Array

The code below represents a stack and the implementation of primitive operations.

```
public class MyStack
{
    private static int nSize = 5;
    int arr[] = new int[nSize];
    int top = -1;

    public boolean isEmpty()
    {
        if(top <= -1)
            return true;
        else
            return false;
    }

    public boolean isFull()
    {
        if(top >= nSize-1)
            return true;
        else
            return false;
    }
```

```
    public void push(int value)
    {
        if (isFull())
            System.out.println("Stack Overflow");
        else
        {
            top++;
            arr[top] = value;
            System.out.println(value + " is pushed into Stack");
        }
    }

    public int pop()
    {
        if(isEmpty())
        {
            System.out.println("Stack Underflow");
            return -9999;
        }
        else
        {
            System.out.println(arr[top] + " is Popped from Stack");
            return arr[top--];
        }
    }

    public void peek()
    {
        if(isEmpty())
            System.out.println("Stack Underflow");
        else
            System.out.println(arr[top]+ " is at the top of Stack");
    }

    public int size()
    {
        return (top+1);
    }
}
```

Code above is the code for stack class. However, to test the MyStack class we will use the following code in the main class.

```
public class StackDemo
{

    public static void main(String[] args)
    {
        MyStack stack = new MyStack();
        System.out.println("Is Stack Empty: " + stack.isEmpty());
        int x = stack.pop();
```

```
        stack.push(23);
        stack.push(2);
        stack.push(73);
        stack.peek();
        stack.push(21);
        stack.push(109);
        System.out.println("Stack Size is " + stack.size());
        System.out.println("Is Stack Full: " + stack.isFull());
        stack.push(13);
        x = stack.pop();
        System.out.println("Popped Value Received is " + x);
        x = stack.pop();
        x = stack.pop();
        x = stack.pop();
        x = stack.pop();
        System.out.println("Popped Value Received is " + x);
        x = stack.pop();
    }
}
```

Output:

```
Is Stack Empty: true
Stack Underflow
23 is pushed into Stack
2 is pushed into Stack
73 is pushed into Stack
73 is at the top of Stack
21 is pushed into Stack
109 is pushed into Stack
Stack Size is 5
Is Stack Full: true
Stack Overflow
109 is Popped from Stack
Popped Value Received is 109
21 is Popped from Stack
73 is Popped from Stack
2 is Popped from Stack
23 is Popped from Stack
Popped Value Received is 23
Stack Underflow
```

The Built-in Stack Class *(For Your Study Assignment)*

Stack is a subclass of Vector that implements a standard last-in, first-out stack. Stack only defines the default constructor, which creates an empty stack. Stack includes all the methods defined by Vector, and adds several of its own.

Apart from the methods inherited from its parent class Vector, Stack defines following methods:

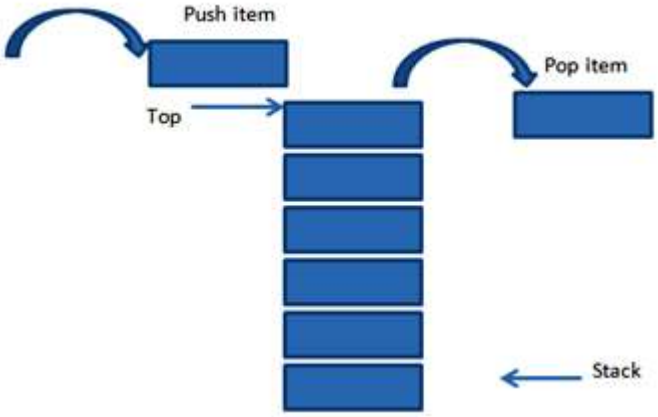
Method	Description
`boolean empty()`	Tests if this stack is empty. Returns true if the stack is empty, and returns false if the stack contains elements.
`Object peek( )`	Returns the element on the top of the stack, but does not remove it.
`Object pop( )`	Returns the element on the top of the stack, removing it in the process.
`Object push(Object element)`	Pushes element onto the stack. Element is also returned.
`int search(Object element)`	Searches for element in the stack. If found, its offset from the top of the stack is returned. Otherwise, .1 is returned.

Lecture

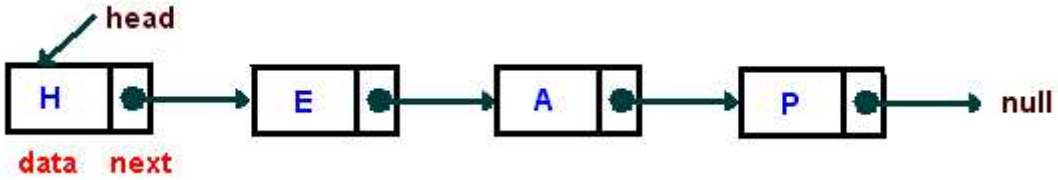
NOTE: FOR FURTHER DETAILS AND MORE COMPREHENSIVE STUDY, PLEASE SEE RECOMMENDED BOOKS OR INTERNET.

Stack Using Linked List (Dynamic Stack)

A ***Stack*** is a collection of objects inserted and removed according to the ***Last In First Out*** (***LIFO***) principle. Stack is data structure used to store the data in such a way that first element inserted into the stack will be removed at last.



On the other hand, a linked list is a linear data structure where each element is a separate object or ***node***.



We have already discussed both the data structures in detail in our previous lectures. Today we will use the dynamic property (***variable size***) of ***linked list*** to create the ***dynamic stack*** unlike stack we created using ***array***.

Code for Dynamic Stack

The code created for ***dynamic stack*** class is also ***generic*** so you can utilize this stack for any type of data you want. The code is given below:

```
public class DyStack <T>
{
    class Node
    {
        T data;
        Node link;
    }
```

```
Node top = null;

public boolean isEmpty()
{
    if(top == null)
        return true;
    else
        return false;
}

public void push(T value)
{
    Node n = new Node();
    n.data = value;
    n.link = top;
    top = n;
}

public T pop()
{
    if(isEmpty())
    {
        System.out.println("Stack Underflow");
        return null;
    }
    else
    {
        T value = top.data;
        top = top.link;
        return value;
    }
}

public void peek()
{
    if(!isEmpty())
    {
        System.out.println("Next Value = " +top.data);
    }
}

public void print()
{
    Node temp = top;
    System.out.println("\n      STATUS     ");
    System.out.println("|_________________|");
    while(temp != null)
    {
        System.out.println("|\t" +temp.data +"\t|");
        System.out.println("|_________________|");
        temp = temp.link;
    }
```

```
    }
}
```

Code above is the code for Dynamic Stack. However, to demonstrate the working of dynamic stack class we will use the following code in the main class.

```
public class DyStackDemo
{
    public static void main(String[] args)
    {
        DyStack<Integer> st1 = new DyStack<>();
        DyStack<Character> st2 = new DyStack<>();

        st1.pop();
        st1.push(10);
        st1.push(20);
        st1.push(30);
        st1.peek();
        st1.print();
        System.out.println("Popped: "+ st1.pop());
        System.out.println("Popped: "+ st1.pop());
        st1.print();

        st2.pop();
        st2.push('A');
        st2.push('B');
        st2.push('C');
        st2.peek();
        st2.print();
        System.out.println("Popped: "+ st2.pop());
        System.out.println("Popped: "+ st2.pop());
        st2.print();
    }
}
```

Output:

```
Stack Underflow
Next Value = 30

       STATUS
|_________________|
|        30       |
|_________________|
|        20       |
|_________________|
|        10       |
|_________________|
Popped: 30
Popped: 20
```

```
      STATUS
|_________________|
|        10       |
|_________________|
Stack Underflow
Next Value = C

      STATUS
|_________________|
|        C        |
|_________________|
|        B        |
|_________________|
|        A        |
|_________________|
Popped: C
Popped: B

      STATUS
|_________________|
|        A        |
|_________________|
```