**Stack Implementation Using Array.**

In this post we will implement the Stack functionality of Array data structure.

We will do this by developing a proper interface and a concrete class that implement this interface.

So in all we will have one interface and one concrete class that will implement the interface.

Also we will make this class and interface generic so we can built stack of any custom type.

Please read the comments carefully. Code is heavily commented to understand the code.

Implementation of StackInterface

**public** **interface** StackInterface<E>

**Push** Method

/\*\*

\* Push the element to the top.

\* **@throws** IllegalStateException if the corresponding

\* array representing stack is full.

\* \*/

**void** push(E item);

**Pop** Method

/\*\*

\* Pop the element from the top.

\* **@throws** NoSuchElementException if the corresponding

\* array representing stack is empty.

\* \*/

E pop();

**Peek** Method

/\*\*

\* Peek the element.

\* **@throws** NoSuchElementException if the corresponding

\* array representing stack is empty.

\* \*/

E peek();

**Is Empty** method

/\*\*

\* isEmpty

\* checks whether the corresponding array representing

\* the stack is empty or not

\* \*/

**boolean** isEmpty();

**Size** method

/\*\*

\* size

\* returns the size of the corresponding array

\* representing stack

\* \*/

**int** size();

Now we have our interface designed. Second thing to do is to construct a concrete class to implement this interface.

Let the class declaration be like following.

**public** **class** StackImpl<E> **implements** StackInterface<E>

As we are using generics, we will declare the Object[] array to store the data.

/\*\*

\* As we are using Generics we declare an Object[]

\* \*/

**private** Object[] arr;

Next, we need to maintain the size stack so we use one pointer for it.

/\*\*

\* We maintain the size counter to check the current

\* size of the stack.

\* \*/

**private** **int** size=0;

Now we declare one Parameterized constructor which takes capacity as parameter. This capacity is for Object[] which will store data.

/\*\*

\* A parameterized constructor that takes stack size

\* parameter.

\* \*/

**public** StackImpl(**final** **int** capacity) {

arr=**new** Object[capacity];

}

Now we start defining the methods of StackInterface<E>.

First we write code for **push**.

/\*\*

\* push method from StackInterface interface.

\*

\* **@throws** IllegalStateException if we try to push element

\* into array is it is full. Capacity of array is

\* determined by Parameterized constructor

\* \*/

@Override

**public** **void** push(**final** E item) {

**if**(size==arr.length){

**throw** **new** IllegalStateException("Stack full cannot add more");

}

arr[size++]=item;

}

Now we write code for **pop**.

/\*\*

\* pop method from StackInterface interface.

\*

\* **@throws** NoSuchElementException if we try to pop the element

\* from array if it is empty.

\* \*/

@SuppressWarnings("unchecked")

@Override

**public** E pop() {

**if**(size==0){

**throw** **new** NoSuchElementException("Cannot pop from empty stack");

}

Object result=arr[size-1];

arr[--size]=**null**;

**return** (E) result;

}

Next is **peek**.

/\*\*

\* peek method from StackInterface interface.

\*

\* **@throws** NoSuchElementException if we try to peek the element

\* from array if it is empty.

\* \*/

@SuppressWarnings("unchecked")

@Override

**public** E peek() {

**if** (size == 0) {

**throw** **new** NoSuchElementException("Cannot peek Stack is empty");

}

**return** (E) arr[size - 1];

}

Now **isEmpty()** method

/\*\*

\* isEmpty method from StackInterface interface.

\*

\* returns boolean

\* true if size is 0.

\* false if size > 1.

\* \*/

@Override

**public** **boolean** isEmpty() {

**return** size==0;

}

Now we write code for size.

/\*\*

\* size method from StackInterface interface.

\* returns the size of the stack.

\* \*/

@Override

**public** **int** size() {

**return** size;

}

And we are now done with entire code.

Now, I will write entire code for interface and concrete class below:

**StackInterface** is as follows:

**package** stackimpl;

/\*\*

\* We write StackInterface<E> as generic interface to

\* provide implementing class capabilities like Stack.

\* \*/

**public** **interface** StackInterface<E> {

/\*\*

\* Push the element to the top.

\* **@throws** IllegalStateException if the corresponding

\* array representing stack is full.

\* \*/

**void** push(E item);

/\*\*

\* Pop the element from the top.

\* **@throws** NoSuchElementException if the corresponding

\* array representing stack is empty.

\* \*/

E pop();

/\*\*

\* Peek the element.

\* **@throws** NoSuchElementException if the corresponding

\* array representing stack is empty.

\* \*/

E peek();

/\*\*

\* isEmpty

\* checks whether the corresponding array representing

\* the stack is empty or not

\* \*/

**boolean** isEmpty();

/\*\*

\* size

\* returns the size of the corresponding array

\* representing stack

\* \*/

**int** size();

}

Concrete implementation is as follows:

**package** stackimpl;

**import** java.util.NoSuchElementException;

/\*\*

\* StackImpl is the generic class which implement the StackInterface

\* interface.

\*

\* All the operations are performed in O(1) time and O(1) space.

\* \*/

**public** **class** StackImpl<E> **implements** StackInterface<E> {

/\*\*

\* As we are using Generics we declare an Object[]

\* \*/

**private** Object[] arr;

/\*\*

\* We maintain the size counter to check the current

\* size of the stack.

\* \*/

**private** **int** size=0;

/\*\*

\* A parameterized constructor that takes stack size

\* parameter.

\* \*/

**public** StackImpl(**final** **int** capacity) {

arr=**new** Object[capacity];

}

/\*\*

\* push method from StackInterface interface.

\*

\* **@throws** IllegalStateException if we try to push element

\* into array is it is full. Capacity of array is

\* determined by Parameterized constructor

\* \*/

@Override

**public** **void** push(**final** E item) {

**if**(size==arr.length){

**throw** **new** IllegalStateException("Stack full cannot add more");

}

arr[size++]=item;

}

/\*\*

\* pop method from StackInterface interface.

\*

\* **@throws** NoSuchElementException if we try to pop the element

\* from array if it is empty.

\* \*/

@SuppressWarnings("unchecked")

@Override

**public** E pop() {

**if**(size==0){

**throw** **new** NoSuchElementException("Cannot pop from empty stack");

}

Object result=arr[size-1];

arr[--size]=**null**;

**return** (E) result;

}

/\*\*

\* peek method from StackInterface interface.

\*

\* **@throws** NoSuchElementException if we try to peek the element

\* from array if it is empty.

\* \*/

@SuppressWarnings("unchecked")

@Override

**public** E peek() {

**if** (size == 0) {

**throw** **new** NoSuchElementException("Cannot peek Stack is empty");

}

**return** (E) arr[size - 1];

}

/\*\*

\* isEmpty method from StackInterface interface.

\*

\* returns boolean

\* true if size is 0.

\* false if size > 1.

\* \*/

@Override

**public** **boolean** isEmpty() {

**return** size==0;

}

/\*\*

\* size method from StackInterface interface.

\* returns the size of the stack.

\* \*/

@Override

**public** **int** size() {

**return** size;

}

**public** **static** **void** main(String[] args) {

StackImpl<Integer> s=**new** StackImpl<>(5);

s.push(1);s.push(2);

Integer t=s.pop();

System.***out***.println(t);

}

}

Now let us check whether overflow and underflow conditions are working properly or not.

For that we design two test cases for it.

In case of Capacity 5 we will insert 6 elements which should throw an exception.

**int**[] a = { 6, 4, 3, 2, 3, 2 };

@Test(expected = IllegalStateException.**class**)

**public** **void** testPushOverFlow() {

**for** (**int** i = 0; i < a.length; i++) {

stack.push(a[i]);

}

}

Now after pushing 5 element we try to pop 6 elements which should throw exception.

@Test(expected = NoSuchElementException.**class**)

**public** **void** testPopUnderFlow() {

**for** (**int** i = 0; i < a.length; i++) {

System.***out***.println(stack.pop());

}

}

Following is code for both the test cases.

**package** stackimpl;

**import** java.util.NoSuchElementException;

**import** org.junit.Test;

**public** **class** StackImplTest {

**final** **int** CAPACITY = 5;

**int**[] a = { 6, 4, 3, 2, 3, 2 };

**int**[] aTest = { 3, 2, 3, 4, 6 };

StackImpl<Integer> stack = **new** StackImpl<Integer>(CAPACITY);

@Test(expected = IllegalStateException.**class**)

**public** **void** testPushOverFlow() {

**for** (**int** i = 0; i < a.length; i++) {

stack.push(a[i]);

}

}

@Test(expected = NoSuchElementException.**class**)

**public** **void** testPopUnderFlow() {

**for** (**int** i = 0; i < a.length; i++) {

System.***out***.println(stack.pop());

}

}

}

Below is the screen shot of both the test cases.