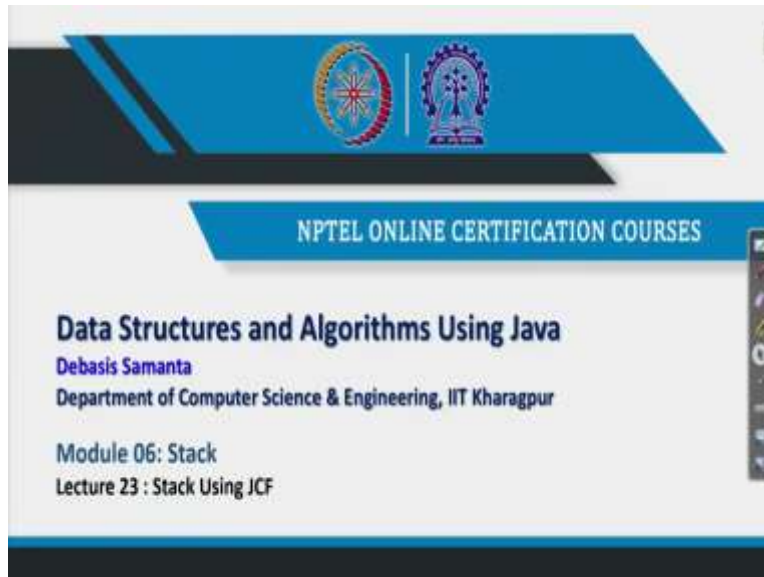


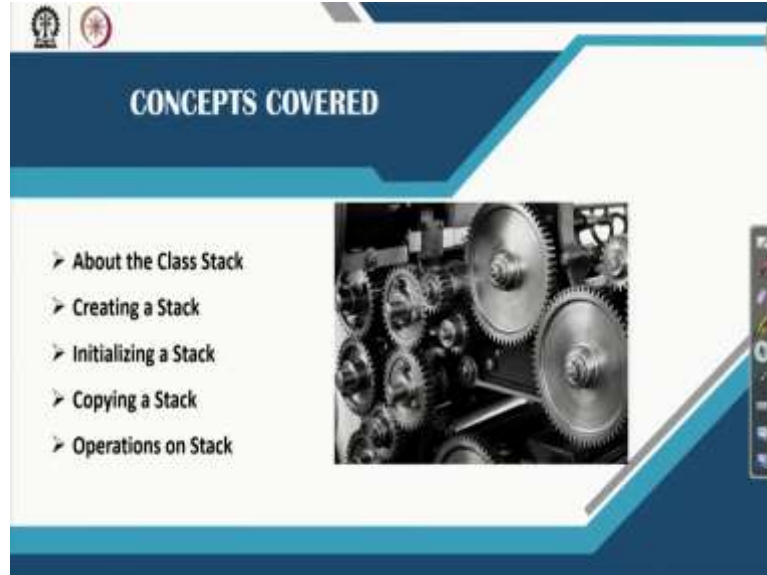
Data Structures and Algorithms using Java
Professor Debasis Samanta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Lecture 23
Stack Using JCF

(Refer Slide Time: 00:31)



We are learning stack data structure. And we have learned about the theory of stack data structure and programming of the structures using Java also we have learnt about it. Now in this video lecture, we will try to learn about how the Java can, how the stack can be realized in a program using Java collection framework.

(Refer Slide Time: 01:05)



So the topic that we will be covering here, so Java supports for the stack realization, a support regarding how a stack can be created, how a stack can be initialized and then stack related different operations like insertion, deletion, traversals and few mass operation copying a stack and status related to, what related to status of a stack and etc. So this is basically see how we can do using Java collection framework support.

(Refer Slide Time: 01:39)

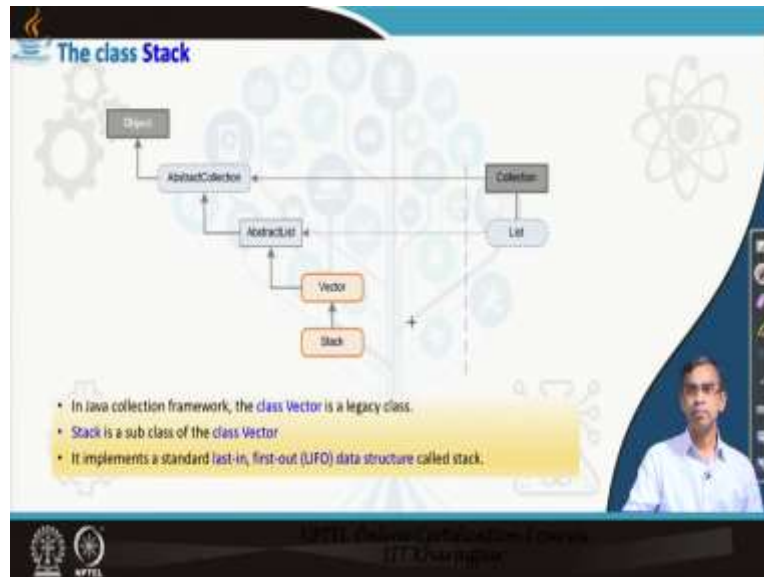


Now, here before going to having the facility of Java collection framework regarding stack related management I want to mention one thing. There is no in its present what is called the SDK, JDK 5 onwards, JDK, there is no explicitly stack class is there. Now this is interesting, why this class is there? Because stack is very important one concept. However you can recall, while I was discussing about the legacy class, there stack is there.

So class stack was earlier known. But later on, Java developer has not included this class more. Now the reason here is that, the idea of the stack can be realized with the idea of the linked list itself that mean if you know the linked list, you know stack actually or if you have the benefits of linked list stack, you basically able to maintain the linked list, the stack. And further again I want to mention here is that, in Java a stack is maintained not by using an array, rather using a linked list.

More specifically it is a double linked list, because linked list in Java is by virtue of double linked list itself. Now, but stack is there, in Java earlier it was there, but recently stack class has been defecated, but still if you write a code, if you use the stack class, the legacy class, then you can have the many facilities about it. So, in this lecture actually we will see, how in your code using the legacy class stack, we can maintain the stack in our things there.

(Refer Slide Time: 03:35)



Now so far the hierarchy of this class is concerned, as we see, stack is a legacy class which is basically is an interface initially. This interface is basically extends, this interface is a sub-class of vectors. So that means vector is basically is a array. So using this array stack, that is why the legacy class basically follow stack as an array, not that linked list. But again you can recall, the vector can grow dynamically, because that vector is designed that way.

So this way stack is automatically grow and that is why it is actually not a static rather dynamic. Now this vector and static and everything, they basically implements static list where static list is basically is an implementation of list. And then this is the abstract collection final is the collection. So this is basically hierarchy of the stack.

So now stack is basically available to you as a interface, but it has been implemented as a legacy class. The class is called the stack actually.

(Refer Slide Time: 04:42)

Methods	Description
<code>boolean empty()</code>	Returns true if the stack is empty, and returns false if the stack contains elements.
<code>E peek()</code>	Returns the element on the top of the stack, but does not remove it.
<code>E pop()</code>	Returns the element on the top of the stack, removing it in the process.
<code>E push(E element)</code>	Pushes element onto the stack, element is also returned.
<code>int search(Object element)</code>	Searches for element in the stack, if found, its offset from the top of the stack is returned. Otherwise, -1 is returned.

Note:

- The methods which are available for the `Vector` class is also available to the `Stack` class, in addition, it has its own methods.

Now let us see, what are the methods are there so for the creation of stack using this class stack is concerned. Now we are again repeating, I am telling again that this stack class is a legacy class. The method in this legacy stack class is listed here, few methods which are very important, as the name implies, you can understand about it. So this method if you call for a stack object, it will tell whether stack is empty or not. Now peek is basically is the method just like a pop.

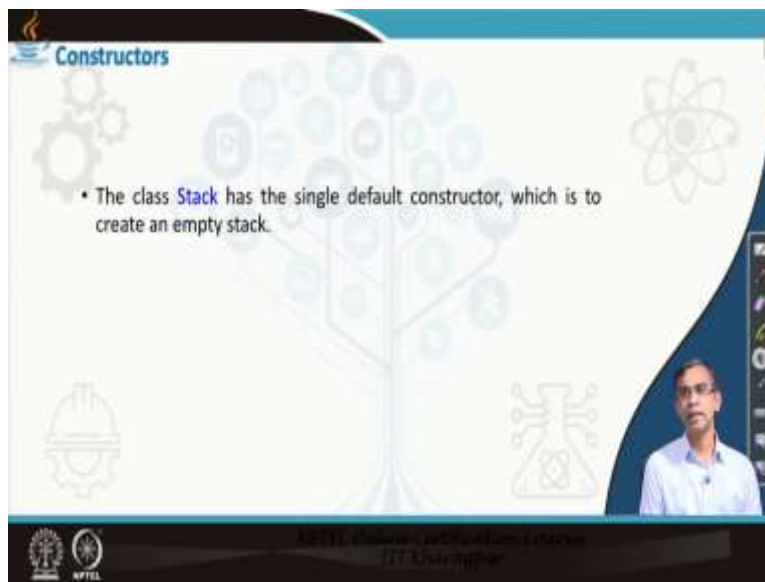
It basically returns the element which is at the top of the stack. But there is another operation also pop. Pop does the same thing. But there are two different things, the operations are there. One thing that peek will check the first element or the top most element but not remove it. But pop operation will check the top most element and remove it after checking it also. So this pop is basically remove operation we can say.

And then there is another push operation, this is the user insertion basically. If an element, if you want to add into a stack, then you can call the push operation for the stack object. Now there is again search operation just like contents. If you want to search an item if it is there in the stack or not, then you can call this method. So these are basically the different methods are there, which are defined in stack class. Actually you can note it all these methods are defined in vector class.

As the vector class and stack is a sub-class of vector, so all the methods, those are there in the vector class also come into the stack class also. But hardly we use all the methods those are there

in the vector in the sub-class, I have listed, rather I have summarized shortlisted few methods those are important. These are the methods actually, so far the stack related operation is concerned, will be used, so that is why I have listed here.

(Refer Slide Time: 06:48)



Now let us have some programming exercise by which we can call the stack objects, create a stack objects using class stack, which is defining Java dot util dot stack. Now first we will discuss how stack can be created using stack class. Now in this class stack, there is only one

constructor that is also default constructor. That means if you want to create a stack, only you have to rely on default constructor.

Default constructor does not require any argument to be passed to the constructor, while it is, while you create an object of this class actually.

(Refer Slide Time: 07:23)



```
// This program illustrates how to declare a stack as a collection
import java.util.Stack;

public class StackCreateDemo {
    public static void main(String a[]){
        // Creating a stack of integers
        Stack<Integer> stack = new Stack<Integer>();
        System.out.println("Stack contains : " + stack); // Printing the stack
        System.out.println("Is stack empty? : " + stack.empty());
    }
}
```

Now here is a simple example, just I advise you to follow it. This example is basically Stack Create Demo. We want to create a stack and for, okay assume that we want to create a stack of numbers. So that is why we said we should, we shall call this method, this is the stack is a constructor is called for integer, because we want to store integers.

And the name of the stack is stack. So stack is the stack object we can say, which basically store integers as an item into it. Now here is a few method we have call. You can simply print LN method you can call, for the stack, so passing an argument stack it will basically print if elements is there. So in this case stack is just created and it is essentially an empty stack now. So it will print nothing.

Now here again, this is the one method you can call stack dot empty for this. So you can understand what it will return, it will return false. So if you print, it will be print false. So this method is very quick demonstration to understand that how a stack can be created without any

programming headache that we have already done in case of, so you do not have to write any code. And then you just simply plug and play sort of things. So this use, we use this one, Java dot util dot stack, it will work for you. So this is the idea that how a stack can be created.

And here again the stack object that you can create is of generic type. That means you can add objects of any type, however it is homogeneous. So if you define an integer, then you should add only integer item, or remove only integer item, but you cannot do any insertion or removing of other item also. Anyway, so this idea about the stack creation is learnt.

(Refer Slide Time: 09:24)



Basic methods related to handle a stack

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

Now let us come to the discussion about basic stack operations. Few operation that you can apply to the stack object is listed here to check whether a stack is empty or not. Now peek is basically, as I told you, it will just check it, but will not remove it. And pop operation is basically remove the item, push basically insertion, okay and search is there.

So here basically so far the stack related operation, these are the methods that is basically useful. Now using these method let us have some example or illustration that how you can write a program, where you create a stack, and some element can be added into the stack and can be removed. And then check the different operation call for there.

(Refer Slide Time: 10:11)

```
// This program illustrates some basic operations on a stack

import java.util.Stack;

public class StackOperationsDemo {
    public static void main(String a[]){
        // Creating a stack of integers
        Stack<Integer> stack = new Stack<Integer>();
        System.out.println("Empty stack : " + stack);
        System.out.println("Is empty stack : " + stack.empty());

        /* Following statement will throw an exception if you pop an
        item from an empty stack. */
        System.out.println("Empty stack : Pop operation : " + stack.pop());
    }
} // Continued to next..
```

```
// Continued on..
// Inserting new data into the stack created
stack.push(1);
stack.push(1);
stack.push(1);
stack.push(1);

// Print the entire stack now
System.out.println("Data in the stack : " + stack);
System.out.println("Pop operation : " + stack.pop());
System.out.println("After pop operation : " + stack);
System.out.println("The element at the top : " + stack.peek());
System.out.println("After peek operation : " + stack);
System.out.println("Search operation : " + stack.search(1));
System.out.println("Is stack empty? " + stack.empty());
}
```

Now this is an example. As you can check this example, it is a very simple. We can give the name of the example is stack operations demo. So first we have to create a stack using constructor you can do it. So this is a stack of integers and the name of the stack is stack. So this is the stack object. And then these are the things we have already learned about it. Now, so here stack dot pop.

Now here you can see, if you want to remove or pop, you have to perform pop operation for an empty stack, it usually throw IO exception. So you have to catch it and then accordingly you can

write code into the catch block that it has happened like this one. If we do not catch it using try catch block, then definitely it gives an error.

So if you write to pop operation like this one, so I should advise you to enclose the code within the try catch block and write the catch block code there, if some exception occur, what to do it. So I have not listed here try catch block, but you should do it while you are running.

Sometimes you will not be able to compile the code without I catch block, as you all know that so exception handling is a must routine for some code. There should be exception handling try catch block to be included. Anyway, so you can add the try catch block, particularly for the pop operation or push operation.

Better to write a throw exception here, also that will serve your purpose. Anyway, these are the different issue that you should take care while writing the robust program. Okay fine, we have discussed about creating a stack and then how the different operations are there. Let us continue this program to have the few more statements, the codes are there.

Now in continuation of this program, I can add few statement like, okay these are the obvious push operation you can understand. So basically go on adding numbers, so as integers. We cannot add any name here, because it is a stack of integers only. Now here you can perform several operations. So printing the entire stack using print LN.

Again using iterator also you can do, because it is also a collection. Iterator method, printing a collection is iterator method that you can use it, but I did not use it here, simply print LN is fine. Now stack dot pop you can understand. Now it is possible if we do it for in this case obviously, what is the last element inserted, so 444. So this pop will basically return this value.

Now after pop operation, if you print it, you can see that it will print this stack this one. Because pop operation will remove this element permanently. Then stack peek as you can see will return this one, peep is a top most element again. And if you print this again you will see the stack remains same. So peep will just simply read the top most element but not remove it. Then stack search 22, it basically check whether 22 is there in the stack or not.

So it will return true, if it presents, otherwise it false. So you can see in this case, it will return true, hence you would. Now after doing this things stack dot empty, but at the moment after following this operation stack is really not empty because it contains one element. So it is basically return false. So these are the few methods we have given for an illustration to understand that how the stack class can be utilized in your program to facilitate the stack related operations are there.

And once those operations are there, you can again come back to the application, either in fixed to post-fix expression, or expression evaluation, following these are the concepts. So basic idea is that those applications are basically, when to push, when to pop. And push and pop are the fundamental operation. That is all, so this way you can do it.

how the elements from all the array can be added into the stack in the same order, the order of the elements in the array.

Now again as the array content duplicates an element, so a stack also can contain duplicate elements as I want to say. And vector also can include duplicate element as you know, so all these things are there. But this is not an issue. Because whether duplicate elements are there or now all are unique element that is hardly not matters. So only matters that operation, push and pop, that is all.

Now here, let us consider this example. First we create an array. Now here we create a string array. String array includes, these are the basically you know the string array includes an expression. So here a plus b star c minus 5, this kind of things. So if you want to store an expression before in fix to post fix calculation, so obviously you have to store the expression.

How you can store the expression? Usually an expression is stored as a string, and then you have to pass the string to getting the token. So token is basically all symbols. Symbols are, means either these are the operand, or operator, or parenthesis, these are the symbol. So read symbol method you can understand why the algorithm I was giving. So read symbol means read a symbol in the string actually, is basically a token that it can return.

Anyway, so I have just a, I mean I have here in this example, maintained one array and that name of the array is called express, expression array. So this basically storing an expression actually arithmetic expression. And what I want to do is that, I want to load the stack using these expression array into the stack actually. So loading a stack using this array of expression, rather we can say is a string actually.

Now here you can see, what I can do, I can declare a stack and this stack is of type string, because I want to load all these value as a string symbol. So I should declare a stack of type string only. So this is the stack declaration, using the stack class. So stack is the name of the stack object in this case. Now here, I can add 1 I can at a time using calling push operation. So I can do it for string as S. This is a for loop.

So for each string S in expression array, we just push the element into the array. So this is a for loop, for each loop rather. So using for each loop, for each element in the array, we can there. Now you can recall, there is no any other constructor like other linked list or linked list for example linked list loading and everything by which a collection can be given this one.

But here you have to go one by one pushing item into this. And whatever with this elements or input that you want to push option of problem, because stack will automatically grow, because stack is a derived class of vector, and vector also grow automatically. So this is the idea about how a stack can be loaded using an array of elements.

Otherwise you can read each element from the keyboard, there is a standard input, and go on adding into this also, that is the one way. That also you can write a code, instead of this one, you can do it. It is the same. It is basically input is taking from the array or input taking from the keyboard. That is the concept of input string from a reading item or reading data actually.

Anyway, so this is the idea about. And finally this will print the stack that you have added into their whole the stack elements. This will print same as express. So if you write system dot out dot print ln expArray, and system dot out dot print ln stack, it will print the same thing actually, because they are basically same set of collections, only one is made duplicator or copy it into others.

Okay, so this is the idea about how the stack can be loaded, either using an array of elements or it is from within keyboard, from the console based input, whatever it is there.

(Refer Slide Time: 19:08)

```
//To create a Stack object with an array of integers
import java.util.Stack;

public class ArrayToStackExample2 {
    public static void main(String a[]) {
        Integer[] intArr = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
        Stack<Integer> stack = new Stack<>();
        for (Integer i : intArr) {
            stack.push(i);
        }
        System.out.println("The stack : " + stack);
        stack.push(104);
        stackSearch();
        System.out.println("Verify the results : " + stack);
    }
}
```

Now, so this is another example, this example also tell you how an array to stack can be loaded. But in this case, here you can see, we create an integer array. So these are the integer numbers we can store. You can define long also, what a, so capital long also, because these are the little bit lengthy number and integer may not, but so integer will can be there.

Anyway, so whether it is a long or integer also no problem, so you can create an array first and this array is created with initialized if these are the whole numbers. And then again using for loop, for each element in this array, we can go on printing. And finally we will print it, it is there. Now you can see, so this, this array, the stack actually it contains all the elements.

If you go on pushing another element, it will not be a problem because it allow duplicate element to be added there. Now here again search. Now you can see, search if it is performed here with, so here basically the element that is top, the element that is in the top is 104. And if you give the push 1, say this will be there. So search actually will search the top most element first. So that is why the first element it is there.

Now you can check it in which order the search works for you if the many elements are there, you can see it anyway. So these are the example that you can see how the elements can be collected from some source, maybe array or somewhere others, and then they can be add in the

stack. And then finally the stack is available to you to perform, to help you to perform many other operations on it.

Okay, so this is the usual operation that is usually you have to follow while you are, you have to use array in your, solving your problem.

(Refer Slide Time: 21:08)

```
Example 23.5: ArrayList to stack loading

//To initialize a stack with a given list of integers stored in a collection object
of type ArrayList.*/

import java.util.*;
public class ArrayListToStackExample {
    public static void main(String a[])
    // create a collection object of type ArrayList
    ArrayList<Integer> list = new ArrayList<Integer>();
    list.add(1);
    list.add(2);
    list.add(3);
    Stack<Integer> stack = new Stack<Integer>(); // Declare a Stack object
    stack.addAll(list); // loading the stack with the collection items
    // loading the first item in the stack +
    System.out.println("The item in the stack : " + stack.peek());
}
```

There are few more things also that I can add here. Now this a very interesting on program you can see, array list. Array list is also a collection. Now here is an array list. The name of the collection is list here. It is a type of integer. We can create an array list, it is like this. Then stack object is created. Now here stack add all, add all is the one method which is defined in the class vector. Because all the method that we have listed earlier, they are not there.

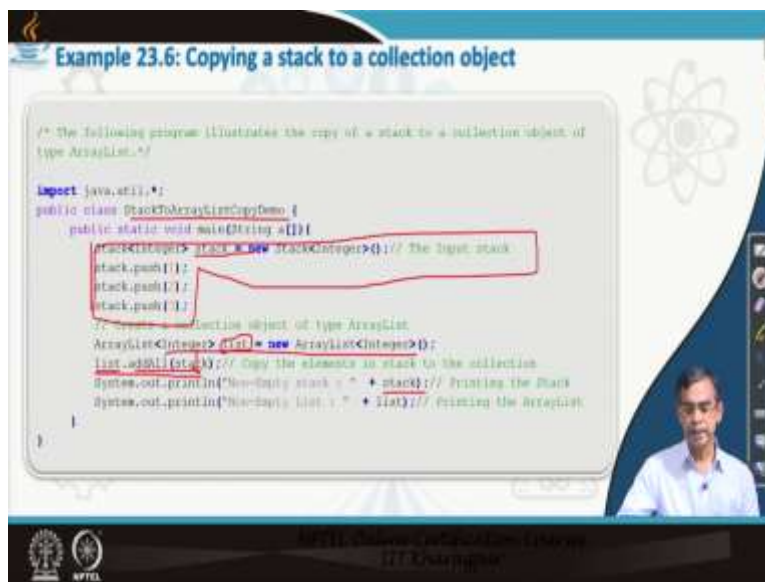
But add all method is you can recall, it is defined in vector class. Now stack it is a derived class, of the vector class, so we can call add all method for this object actually. Now here you can see, for the stack object that we have created earlier, for integer, right? You have to create a stack object first ofcourse. So it a stack object is created which is initially empty in this case.

Now initially empty stack ad if we add all passing this argument is a list, then what will happen is that, all the elements which are there will be added into the stack in the same order 1, 2, 3 in that way. So this is the one idea about stack. And then if you go to the peep, so which is the top

element, it will be shown to you. So third is the in this case the top element because it is added in that order.

Anyway so here this example is an interesting example that how the bulk operation that means as a collection you can add into a stack. And many other methods, those are defined again for the vector also you can apply here also. That is left as an exercise for you, you can just practice.

(Refer Slide Time: 22:50)



Now, opposite operations also can be there. Now in the last example we have learned about how a collection can be added into stack. Opposite operation is basically how a stack can be copied into a collection. So let us see, some example for this. Now here is an example that you can follow. So this example give stack to array list copy. That means, given a stack, we want to copy all the elements which are preset in the stack to an collection, the collection is array list.

So what actually we require, first we have to have a stack. So let us, this is the stack and it has some element in it. So this basically having the stack at the moment. Create a stack and load the

stack with 3 elements. Now we want to copy this stack into an array list collection, so for which we should create a array list object. So list is the array list object. And it is should be the same type as integer. So we create an integer objects like. Now here, you see this important data.

Now list is an array list. In the list class, in the array list class, there is a method add all. Actually add all method is defined in collection, and all these methods are implemented in all respective sub-class of the collection that is why add all method is also available in the array list. Now so if we can call add all for the objective list, that means the collection, and passing an stack, stack is also another collection. So it is basically copy a collection into another collection.

So this way, we can achieve how the entire stack can be copied into a correction, in this case array list. Now we can create another vector collection that also you can do. So collection of stack into vector or any other arrays, or any other collection type actually. And then finally this is printing the stack and printing list.

You can see both, both printing will print all the same items in them. So this is the idea, it shows that how the bulk operation and then copying stack can be done, and in many occurrence, many situation, you have to copy from here to there and there to here, and that is why those are the useful, what is called the discussion in the context of stack related application.

(Refer Slide Time: 25:22)



Example 23.7: Basic operations like push, pop and peek

```

/**to create a stack and performing basic operations like push, pop and peek*/
import java.util.*;

public class StackExample {
    public static void main(String[] args) {
        // creating a stack
        Stack<String> stackOfCards = new Stack<String>();
        // Pushing new items to the Stack
        stackOfCards.push("Jack");
        stackOfCards.push("Queen");
        stackOfCards.push("King");
        stackOfCards.push("Ace");
        System.out.println("Current Stack => " + stackOfCards);
        System.out.println();
    }
}
// Continued to next...

```

Example 23.7: Basic operations like push, pop and peek

```

// Continued to next...

// Popping items from the Stack
String cardAtTop = stackOfCards.pop();
// These lines are commented if the stack is empty
System.out.println("Stack.pop() => " + cardAtTop);
System.out.println("Current Stack => " + stackOfCards);
System.out.println();
// Get the item at the top of the stack without removing it
cardAtTop = stackOfCards.peek();
System.out.println("Stack.peek() => " + cardAtTop);
System.out.println("Current Stack => " + stackOfCards);

```

Now I just before concluding to have some few more operations on the stack, that is searching, then capacity availability, and all these things are there. Even you can call the clean method to delete the entire elements on the stack and everything. Now so there are few example I want to discuss about it, using a stack class again. So these are stack example, Java dot util dot stack or star, whatever it is there. If you want to use any other class, that is belongs to util dot things also you can have it.

So this is the basically declaration of a stack. The name of the stack object is stack of cards. You know, if you want to have a game, software development, building, then playing cards, like

Solitaire or whatever it is there. So this actually you have to maintain many cards or whatever it is there, graphics and then other audio video can be also be included there, animation. Anyway, so what I want to say is that stack is required there whenever you want to play a game, or you want to design a game.

So obviously this example is not up to that kind of details that okay we are playing, designing a game software here. But anyway, I am telling you, if you want to plan search that kind of application software, you may have to use the stack for your purpose. Anyway, now let us see, we have declare a stack, stack of type string. And these are the few add operation basically different elements are added into the stack. Then it is a user printing operation, printing.

So what, in this part of the code, what we have done, we have created a stack of string objects and added some objects into this stack, and then print, we print the stack object to see that in which order they have added. It is the, already you have learned about these things. Now let us continue this program to learn few more, what is called the code in it, to give it little bit shape, like adding, removing, all these things are there.

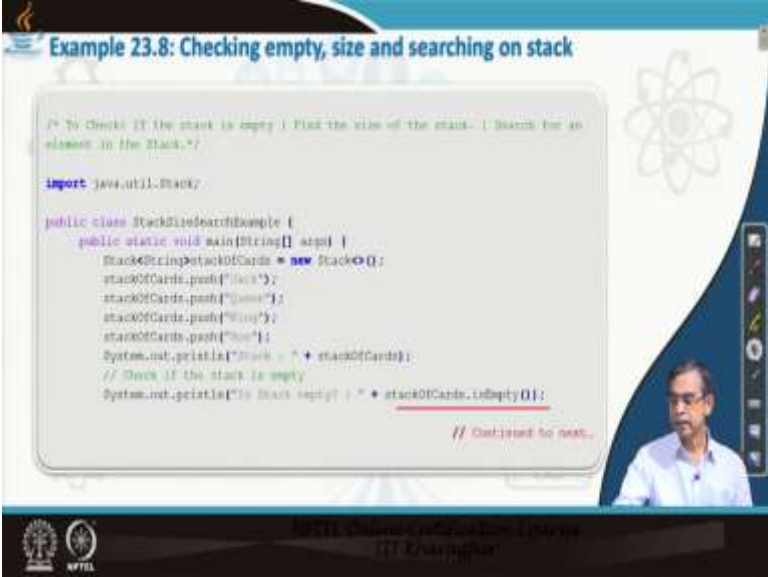
So here I see stack of cards pop, you can understand what it will do. It is the top most element will be deleted. And then, after deleting it is printing. And then, that it will print, card at top also, it will see pop. Okay, so card at top is basically here pop operation will return card at top actually and printing this one. The same thing that you can do it here. And then here current stack is basically after deletion, what is the remaining elements are there, you can print it.

Now here if you see, again you can apply the peek operation and you can identify what is the difference between peek and pop by executing these are the two statements, after peek and after pop. You can understand that one remove, one does not remove. It kind of things are there. Anyway, yes. Okay, so these are the basic operation we are already familiar which I have just repeated it for your understanding again or exercising. My advice is that you can run all the course that is, that is presented here, and so that you can have the flavor about it.

And you can also add some statement into the code, those code, so that rewrite the code program and then you can learn it. So you can do lot of experiments about it. And then you can think some gaming, game player, game software, and then you can see using stack how it can

implement. So this is the innovative idea that you can think and you can learn, so that will help you to, I mean gain the confidence about Java programming and particularly stack application in it.

(Refer Slide Time: 29:02)



The slide displays a Java code snippet for a stack application. The code includes an import statement for `java.util.Stack`, a class definition `StackDemoSearchExample`, and a `main` method. The `main` method creates a `Stack` object, pushes several elements, prints the stack, and checks if it is empty. A comment indicates that the code is cut off to save space.

```
/* To Check if the stack is empty | Find the size of the stack | Search for an element in the Stack.*/  
  
import java.util.Stack;  
  
public class StackDemoSearchExample {  
    public static void main(String[] args) {  
        Stack<String> stackOfCards = new Stack<>();  
        stackOfCards.push("Jack");  
        stackOfCards.push("Queen");  
        stackOfCards.push("King");  
        stackOfCards.push("Ace");  
        System.out.println("Stack = " + stackOfCards);  
        // Check if the stack is empty  
        System.out.println("Is Stack empty? : " + stackOfCards.isEmpty());  
        // Cutted to next...  
    }  
}
```

Now this is another okay, I have just okay, expanding the example again. And this basically the same program that we have discussed in the last, okay 23 point 7. This program here is basically same. But here we are see using `isEmpty`, right? After performing that operation, is empty. Is empty, already you are familiar. So it is no more necessary to discuss in details.

(Refer Slide Time: 29:35)

The slide displays the following Java code with annotations:

```
// Continued to test...

// Find the size of Stack
System.out.println("Size of Stack = " + stackOfCards.size());
// Search for an element.
// The search() method returns the
// 1-based position of the element from the top of the stack
// It returns -1 if the element was not found in the stack.
int position = stackOfCards.search("Queen");
if(position != -1) {
    System.out.println("Found element 'Queen' at position = " + position);
} else {
    System.out.println("Element not found");
}
```

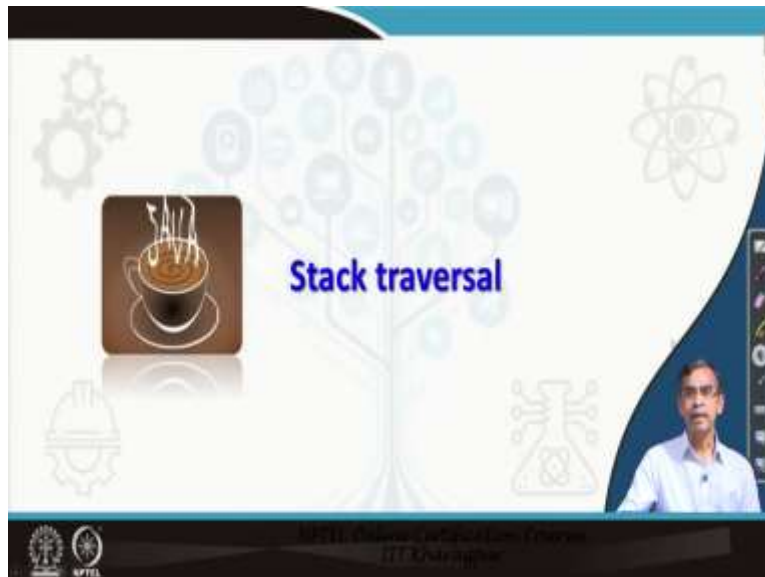
Red circles highlight `stackOfCards.size()` and `stackOfCards.search("Queen")`. A red line underlines the `if` statement. A small inset video of a presenter is visible in the bottom right corner of the slide.

Now okay, there are few more operation that you can think about it. Here, I have mentioned here. So searching, this size is also there. That means, what is the number of, size is basically how many elements are present at the moment in the stack. So the size can be, size method can be called for a stack object to have the size.

Now, here the position search Queen. That means, if you pass an object and then you can search again. The method it is there defined in vector class again, for this stack object. So it basically search and if it found, it will give the index at which location it is found. So index actually, if it does not found any location, it return minus 1. And then you can just see whether the element present or not, you can check it.

So this is the method by which the capacity of a stack at any moment can be understood, and a search can, and a stack can be searched for an item, whether it presents at there at not. And so these are the pretty simple things, those things are okay you can have it readily available in your program, I mean to be used in your program.

(Refer Slide Time: 31:01)



Now traversing a stack, there is a different ways, all the items, because stack is a collection. So all traversals, those are applicable to collection is also applicable to here, like say iterator, and then for each, and simple print. But we will discuss about few operations like here.

(Refer Slide Time: 31:20)



```
// Continued on ...
System.out.println("Iterate over a Stack using Iterator()");
Iterator<String> iterator = stackOfPlates.iterator();
while (iterator.hasNext()) {
    String plate = iterator.next();
    System.out.println(plate);
}
// Continued to next...
```

So first we have to create, here we want to give a demo about how iterator can be applied to stack object to traverse the full stack. Now you can see, push and pop operation is that the permissible operation. But here the stack class can allow you to traverse from top to bottom, whatever the elements are there.

Now let us have an example, so this example, say the iterator over the stack. Now first we have to create a stack, the name of the object that we have created here, stack of plates. Then we just place the element into the stack, plate 1, plate 2, plate 3, plate 4 is a string. Then here, there is a traversal, it is called the 4, which is a, this is called the lambda expression.

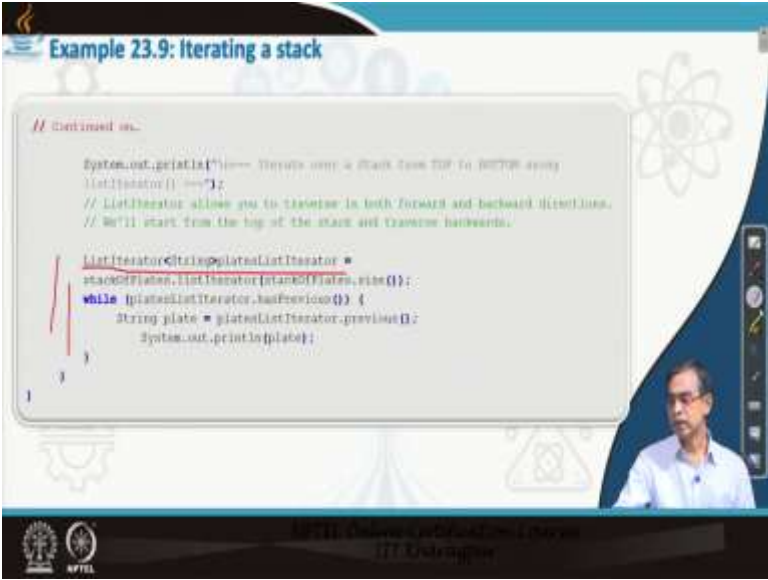
Now regarding this lambda expression, we will discuss in details not now, because it is different topics to be discussed. So we just simply, keep on holding right now, but lambda expression is look like this, so for each then plate, if plate is present there then you can just print it. So it is the idea about printing the element there, and you see the, within this bracket this is there. So it is the concept.

So plate and then pin the plate. If plate is present there, in the for each, I mean in the stack of plates, it is the concept. So this basically the idea about how a stack can be printed using lambda expression. Now other than lambda expression, there is another user operator of iterator also, those we are already familiar to, can also be exercised here.

And here as you see, you use the iterator for this object stack plates. And this is basically iterator object. And this iterator objects of type string, that mean it iterates over the string objects. And for this operator has next and next. Has next check that whether still there element is there or not. If element is there, it basically return and then return can be done by means of next method, and print the method it is there.

So these are, iterator you are already familiar to. You have used in many occasions within the use of linked list and other also, array list, arrays and everything. So this is the way that array list, a stack also can be, be traversed by using this concept. So what I want to emphasis again, a stack is nothing but a collection. But only the stack whenever you have to apply it for certain application, restricting the operation, insertion and deletion only at one end. So that those things are can be taken place here also.

(Refer Slide Time: 34:13)



```
// Continued on...

System.out.println("====> Iterating over a Stack from TOP to BOTTOM using
ListIterator() ==>");
// ListIterator allows you to traverse in both forward and backward directions.
// We'll start from the top of the stack and traverse backwards.

ListIterator<String> platesListIterator =
stackOfPlates.listIterator(stackOfPlates.size());
while (platesListIterator.hasPrevious()) {
    String plate = platesListIterator.previous();
    System.out.println(plate);
}
```

So here is another way, now a stack can be also traversed either from top to bottom and bottom to top also. Now that also can be done by many ways. Descending order traversals or this order also. But here we can do it in a little bit different way. You can check the code here, this basically, again list iterator. See, earlier iterator is a simple iterator.

A list iterator is there, because array, a stack is also in, is defined by list also. Because it implements the list class there. Now so list has the iterator called list iterator. So list iterator

basically allow you to list in both direction, from left to right and from right to right, because list is basically a double linked list sort of thing.

So using this list iterator, you can traverse the stack in a opposite order. So it is from bottom to top like. So this code you can check and it will say, it will give you to traverse the stack in opposite order also. So stack it is like this, so whether it can be in order reverse order or in order, that you can decide and accordingly you can apply it.

But for the illustration I have mentioned that how the stack can be traversed in a different way. So these are various ways by which a stack can be visited and its element can be added.

(Refer Slide Time: 35:39)



Now for your further study and everything, I should suggest you to go to the book. There you can find a lot of discussion regarding legacy class and the link, the second link is basically the link of my course page, where you can find all course and detail discussion about stack class also there.

And then this is the tutorial material regarding details about the many methods and operation, those are there in the stack class. Okay, with these things let us conclude the stack concept here. We will move to another module discussing queue structure in the next class onwards. Thank you very much.