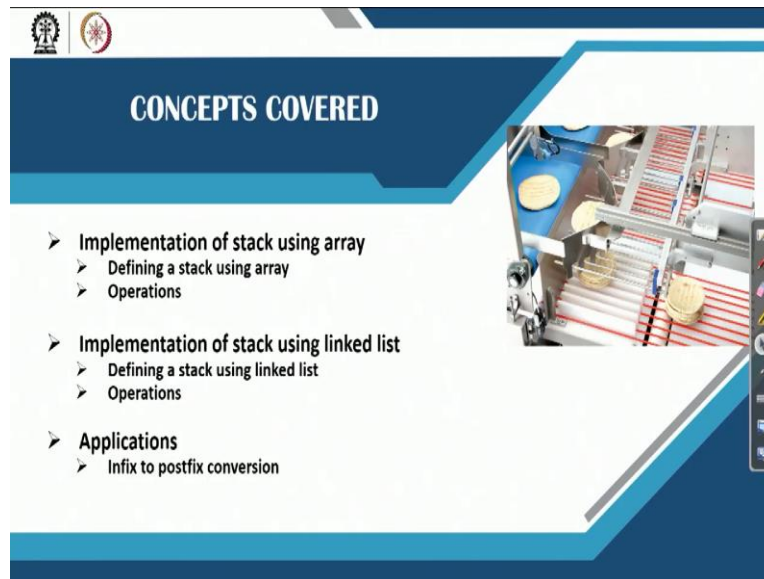


**Data Structures and Algorithms using Java**  
**Professor. Debasis Samanta**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**  
**Lecture 22**  
**Programming for stack**

We have learned about Stack data structures. Now let us see how a Stack structures can be Programmed using Java.

(Refer Slide Time: 00:40)



So, in today's video lectures we will exactly how a stack can be realized or a stack can be, mean programming can for stack can be done using the concept of array. So here basically, we have to define, different operations related to array, but all those operations in the view of that array is maintained using an array.

The same thing again we can do but using a linked list and finally, I will demonstrate one application of a stack, this application is basically how an arithmetic expression can be evaluated. Now regarding this array implementation for example say you want to, stack implementation say suppose you want to implement a stack using linked list. You do not have to define all linked list related operations and it's definition.

Whatever the linked list programs that we have done, while we are discussing about programming for linked list all the codes that is those are created there you can import into your program here in while we are programming using stack. I will discuss this things let us proceed.

(Refer Slide Time: 02:05)

**Example 22.1: Defining the class stack using array**

```
// This program shows how a stack can be defined using an array

/* import array package;
   This program uses array related implementation; so, include the directory,
   where all those programs are defined. */

class Stack<T> {
    T[] data;
    int length;
    int top;
    Stack(T[] a) {
        data = a;
        length = a.length;
        top = -1;
    }

    /* The methods push(), pop(), isEmpty(), printStack() are to be defined here.
       See the next slides. */
}
```

NPTEL Online Certification Courses  
IIT Kharagpur

First, let us see how a stack can be programmed using an array. Now here is a basic definition that you can consider. So, first we have to declare that stack is an object, so to declare this things we can define a class. The name of the class is stack a means that stack using array and we want to make the stack is a generic that means this stack can store any type of data, integer, long, float, string or any abstract data type.

So that is why we declared this class as a generic class. So, once we declare this class as a generic, then we have to include the fill and the different methods in it. The first fill that we have to declare that length it will store what is the current length of the stack and top is a pointer it indicates that what is the top pointer of the stack. Because the stack is characterized with a top pointer.

So, top will basically says that index. So basically, it index varies from minus 1 to length minus 1, minus 1 indicates that stack is empty and top equals to 0 indicate that just only one item there in the stack. So, in that case length is 1 and that is why length fills is length minus 1 is the maximum. So, if the stack of size n, so length is equal to n actually but the index varies from 0 to n minus 1 that you know and here, a stack essentially is a number of elements to be stored, so declare it is an array, the name of the array is data here.

So, that is why this is the stack a declaration using array. So, this array data will store the elements in it and the type of the element that this store as we have declared is a generic. So, this

the concept and so there is constructor, this constructor for any generic class you know, for every generic class you have to define a constructors. So, this is the only constructor that is there, so in this constructor we can pass an array is a reference to an array objects. So, this array objects will be initialized to the data. So, if you can create a stack initially passing an array, so initial array is there.

That array can be null in that case, it will be null that means we do not any elements in it and length equals to a dot length. So, a is if it is null then a dot length is 0, so it is 0 and top equals to minus 1 initially the array has the pointer, top pointer is minus 1 because it is empty, stack is empty. So this basically, declares the basic thing, basic elements of the array if few fills and the contain, container which basically contains the element into the stack and then, our next task is to add the different methods which is require to manipulate stack.

It is just like a linked list programing that we have note, we have to create the linked list structure like and then we have to add the methods those are equal to manipulate linked list here the methods namely push, pop whether array is empty not or is empty method, print stack, printing the all elements which are present in the stack and like like. So now, our task is to add all these method one by one. So, let us discuss about how we can add the methods and then our stack data structures can be defined and then finally it can be used.

(Refer Slide Time: 06:20)

**Example 22.2: Defining the method push()**

```
// This part of the program includes the definition of the push() method
// Defining the push operation
void push(T a) {
    if (top < length-1) {
        top++;
        data[top] = a;
    }
    else {
        System.out.println("Stack Overflow");
    }
}

// The next method pop() is to be defined here .... See the next slide
```

NPTEL Online Certification Courses  
IIT Kharagpur

So now, we will add the first method namely it is the push method. So, here is a method push as known, as a push method require the element to be push. So, a is basically of type T, if T is integer then a is of integer T if it is double then a is of type double, T is string then a is string like so whatever it is there. So, this push method and now this operation you see it is very. Now push operation first check whether the stack is already full or not. So, here if top less than length minus 1, that mean stack is not full.

Then we can go here, so if not there then top will be incremented by 1 and then at the location of the top the value a which passed here will be assigned. So, this way the element will be pushed into the stack and then top pointer will be incremented by one. So, top pointer is incremented by 1 here as you see. If top is either equals to length minus 1 or greater than whatever it is there then it will not push any item in that case. We can say that stack overfull that mean stack is full.

So, this is the method call the push method we can add into the stack structure that we are defining class we are defining. Now likewise, we can add one more method this is related to pop, pop operation means it is just deleting the top element from the stack.

(Refer Slide Time: 08:02)

**Example 22.3: Defining the method pop()**

```
// This part of the program includes the definition of the pop() method

// Defining the pop operation
T pop() {
    T a = null;
    if (top == -1) {
        System.out.println("Stack Underflow ");
    }
    else {
        a = data[top];
        top--;
    }
    return a;
}

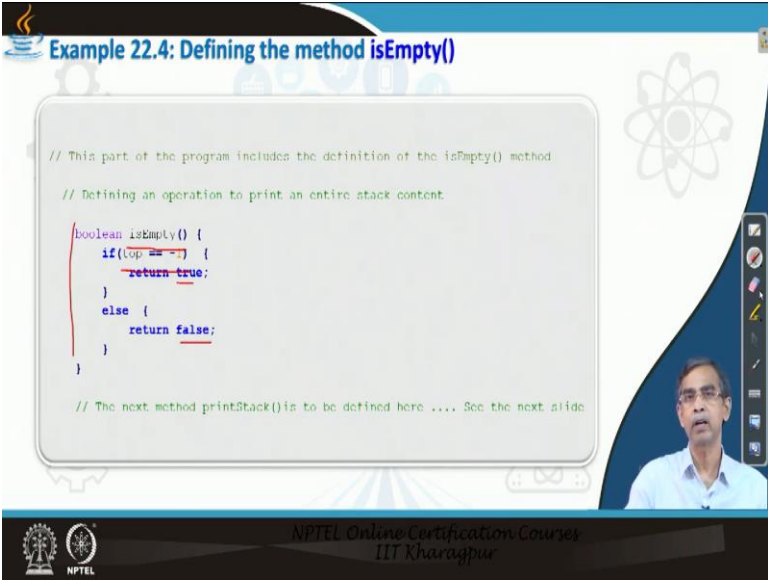
// The next method isEmpty() is to be defined here .... See the next slide
```

NPTEL Online Certification Courses  
IIT Kharagpur

Now that method it is like this again like push method, we have to add one more method and this is the body of the method. Now top, this pop method does not require any argument to pass, so that is why argument is void here and this method you can use in case of push it, does not return anything, but here it return the elements which is basically move from the stack. So, T is basically the type that it will return, so in that case the data type that the stack is used for it will return of that type.

So initially, T a equals to null, so this value that needs to be return is basically null at the movement. Then it check, if there is empty the stack is empty or not. So, top equals to minus 1 it indicates that array is a stack is empty, so stack is under flow, so you will not anything. If it is not empty then a the value will be copied the top most value will be copied into the array a and top will be decremented because it just it is case that okay it is decremented in top, down once and then it return a. So, this way this method will work for you and this is the pop operation related to the stack structure.

(Refer Slide Time: 09:31)



The slide displays a code editor window with the following Java code:

```
// This part of the program includes the definition of the isEmpty() method
// Defining an operation to print an entire stack content

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

// The next method printStack() is to be defined here .... See the next slide
```

The slide also features a small video inset of a man in the bottom right corner and a footer with the NPTEL logo and text: "NPTEL Online Certification Courses IIT Kharagpur".

Now next method it is, is empty it check that whether it is empty not or it is very simple method actually just we have already know that how to check it. So, whether a stack is empty that means T top is equals to minus 1 it return to else return false. So, this will not delete or not add anything only check whether stack is empty or not in many operations it is refer to check whether stack is empty or not before applying stack operation there that is why there.

Now, so we have these are the essential operations related to the stack structure. Our next operations is basically print stack it is basically printing the entire stack it is very simple just, it is the method that it print the whole array because stack is now is an array here.

(Refer Slide Time: 10:28)

**Example 22.5: Defining the method printStack()**

```
// This part of the program includes the definition of the printStack() method
// Defining an operation to print an entire stack content

void printStack() {
    if(top == -1) {
        System.out.println("Stack Empty");
    }
    else {
        for(int i = top; i >= 0; i--) {
            System.out.print(data[i] + " ");
        }
        System.out.println();
    }
}
// End of the definition of the class StackA
```

NPTEL Online Certification Courses  
IIT Kharagpur

So, this method is declared here print stack method, this is the body as this method does not return anything. So it is a void, now here you see first we will check that whether stack is empty, if stack is empty they no more to print anything, that is why else it will just a loop in this loop, it will traverse each elements in the array starting from top to whatever it is their I mean 0, so here and it is there.

So, this way it will print all the elements starting from the top to the bottom elements it is there and that is all, so this is a print stack method and this completes adding all the method push, pop, is empty and print stack and your stack a is ready. Now to test your program again it is advisable by you should run a mater program, a driver program calling all the methods that you have defined starting with an empty array or adding some array into there.

(Refer Slide Time: 11:29)

**Example 22.6: Basic stack operations using the class StackA**

```
/* This is the main program, illustration the usage of the class defined. You
should include the package, where this program is stored. */

class StackImplementationDemo {
    public static void main(String[] args) {
        Integer a[] = new Integer[2];
        Stack<Integer> st = new Stack<Integer>(5); +
        st.push(5);
        st.printStack();
        st.push(5);
        st.push(5); ←
        st.printStack();
        st.pop();
        st.printStack();
        st.pop();
        st.printStack();
        st.pop();
        System.out.println("Is empty? " + st.isEmpty());
    }
} // End of the demo class
```

NPTEL Online Certification Courses  
IIT Kharagpur

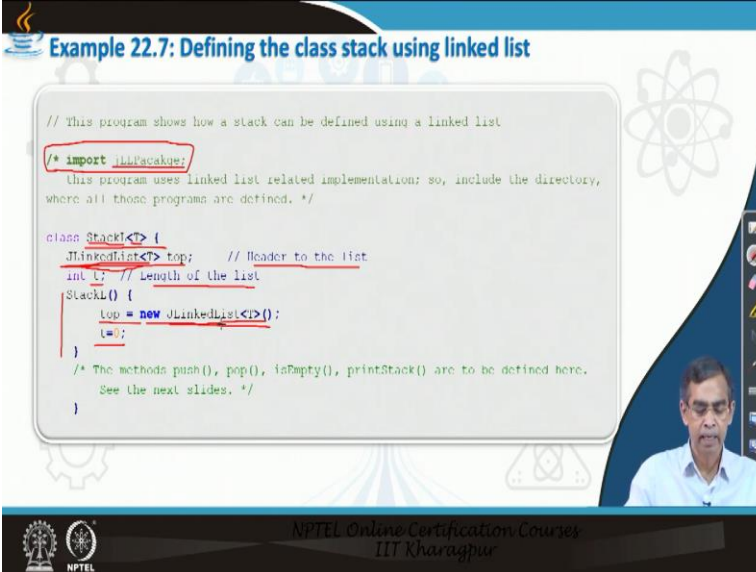
So, this is the driver method that I have given you for your practice, you can check this program. Here you see we declare a is an array of size two here and here we create using this array a, one stack the name of the stack object is st. Now st dot push 5, we can insert one element here then printing stack then here. Now you can understand what will happen, is basically, it will not be able to push it. Because size of the is two only, so this will basically is a failed operation in this case that you can check and the otherwise you can see all these operations are valid and finally this also is empty for the stack it check that whether stack is empty, anyway.

So you can just (())(12:26) instead of 2 contain and everything you can go on adding and more data and pushing more data into the stack and pop operation second in between push pop whatever it is there and you can check that this basically program works for you. So, stack implementation or programing a stack using array is very simple actually. Now we will discuss about the same thing but using linked list.

Now stack implementation of stack or programing of stack using linked list again very simple what actually you have to do is that push operations, if you see as you said that stack is basically, last in first out operation that mean you can understand that for the linked list insertion and deletion related to the push and pop operation only. So here, push is basically, insert at in that case it is basically front and then removal I mean pop is also front like whatever it is.



(Refer Slide Time: 13:48)



**Example 22.7: Defining the class stack using linked list**

```
// This program shows how a stack can be defined using a linked list
/* import JLinkedList:
   this program uses linked list related implementation: so, include the directory,
   where all those programs are defined. */

class Stack<T> {
    JLinkedList<T> top; // Header to the list
    int L; // Length of the list
    Stack() {
        top = new JLinkedList<T>();
    }
    /* The methods push(), pop(), isEmpty(), printStack() are to be defined here.
       See the next slides. */
}
```

NPTEL Online Certification Courses  
IIT Kharagpur

Now let us see how we can define this operations like, first of all we have to import this, this basically is a directory where all the programs related to linked list that basically, that includes the linked list, linked list class declaration JLL declaration that we have already done.

J linked list actually it is name of the class that we have done and all the methods that is there. So we have to import that class actually here. Then we can define our stack objects using linked list let the name of the stack class for this type of object is stack L and, we again we want to, we want to make it generic, so that is why we just, we are basically we have to define a generic class. So, stack L is a generic class with one only generic argument.

Now here, this J linked list T as you see this is the one class we have already defined which is there in this directory and so we create a top, top is basically is a pointer and this pointer is a type node, because it a node. So this top is basically is a node to a linked list, so we can say it is header and T is an integer variable just to note the length of the list we can L. So, this is because size of the stack we can say and here is a constructor, so that mean initially it will define a stack and, here you see we created linked list actually these are top is basically points to the linked list that means it is basically top is nothing but the header.

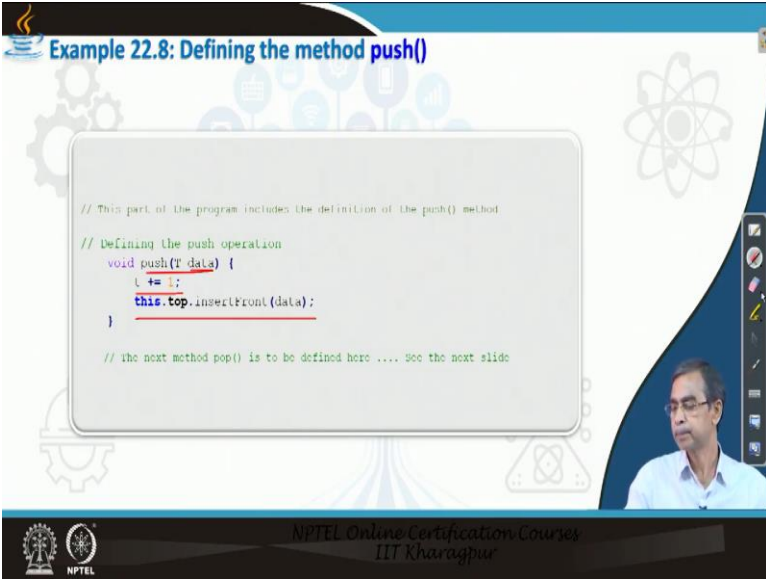
So, top is header and whenever the linked list constructor if you can recall it basically create a empty list with header points to a null or link fill of the header is basically null or header if it is

header dot linked is equals to null we can say so it is basically top actually. So, top is basically always points the first node of the linked list that means it is our node which basically points the header node actually.

So, this is the top and T equal to 0 this indicates that initially the size of the stack is 0 because there is no elements in it. So, that is all, so this basically completes the declaration of a linked list and its inclusion into a stack maintaining a stack actually is in linked list. Now our next step is basically, to add different methods related to this kind of stack manipulation namely push, pop, is empty, print stack. Now we have already exercise this methods but using array representation.

Now, push pop and is empty, print stack method is for the linked list implementation. Now here so as I told you the idea is that push is basically insert front, pop is basically delete front and is empty and print stack is basically traversal and is empty is basically check that whether top point scale null or not that is all. Now let us see how all these methods, pretty simple methods actually how we can define all this method one by one.

(Refer Slide Time: 17:38)



The slide displays the following C++ code snippet:

```
// This part of the program includes the definition of the push() method
// Defining the push operation
void push(T data) {
    t += 1;
    this.top.insertFront(data);
}
// The next method pop() is to be defined here ... see the next slide
```

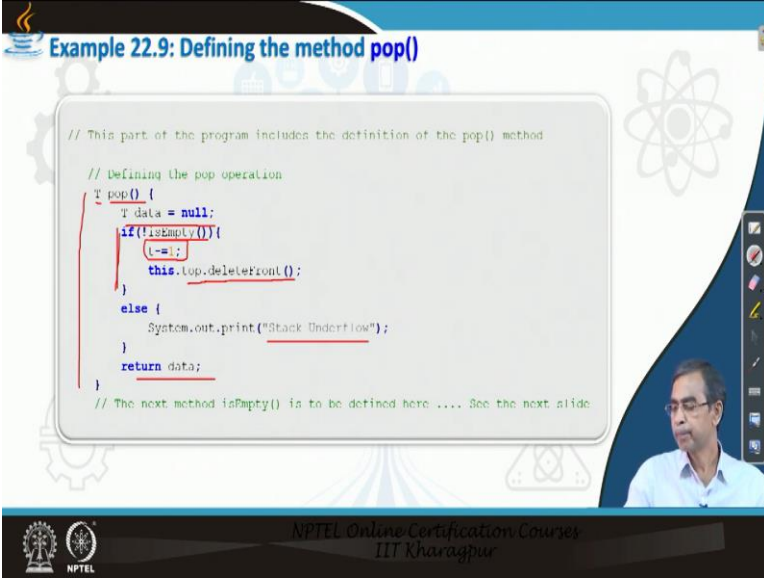
The slide also features a video inset of a man in the bottom right corner and logos for NPTEL and IIT Kharagpur at the bottom.

Now here the push method and the push and that this data that we want to add into the stack. So, initially it will basically it will increment the high job the stack. Because once we perform a push operation then it is one and then we just simply you see what we are doing is that these dot this basically the stack top means its top and at that top we have to insert front. So, we just simply add the insert, the insert a new node at the front of this, of the linked list single linked list we can

say in this case we are considering linked list, you can use again double linked list to maintain the stack also no issue.

So, this is the idea about push operation and one more point that you can note is that in this case we do not have to check whether it is overflow or not or full or not. Because the stack implementation using linked list theoretically having the unlimited size, because it will grow dynamically, unlike the array implementation where the size or length of the stack is limited to the size of the array. So this is obviously one advantage why you can think for implementing a stack using linked list. Now this is a push operation.

(Refer Slide Time: 19:12)



**Example 22.9: Defining the method pop()**

```
// This part of the program includes the definition of the pop() method

// Defining the pop operation
T pop() {
    T data = null;
    if (!isEmpty()) {
        top--;
        this.top.deleteFront();
    }
    else {
        System.out.println("Stack Underflow");
    }
    return data;
}

// The next method isEmpty() is to be defined here .... See the next slide
```

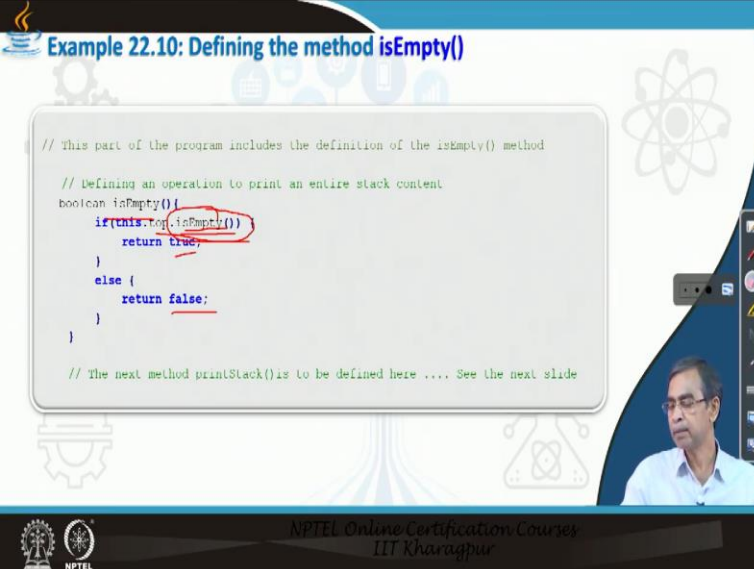
NPTEL Online Certification Courses  
IIT Kharagpur

Now let us come to the pop operation, the pop operation is again like this one, but for the pop operation we have to check that whether list is empty or not, in case of list is empty so pop there will be I mean pop operation is not possible, that is it is called stack is underflow.

Now here is the method for the pop operation and it does not require any argument to passed, but it will return the value that is has popped successfully and, initially these data that it will return is null. Now we can check if is not empty is empty is a method that we can discuss for this linked list again and if it is not empty then T is equal, T is in decremented by 1 because once the pop is successful it basically reduce the size of the stack by 1 and here delete front operation that mean it will delete the front most node from the list which basically pointed by the top.

So this basically removing. Now here if fails that mean if list is empty then it will print the stack underflow no more pop operation successful and if it is successful then it will return data. So this is the pop operation.

(Refer Slide Time: 20:39)



**Example 22.10: Defining the method isEmpty()**

```
// This part of the program includes the definition of the isEmpty() method

// Defining an operation to print an entire stack content
boolean isEmpty() {
    if (this.top.isEmtpy())
        return true;
    else {
        return false;
    }
}

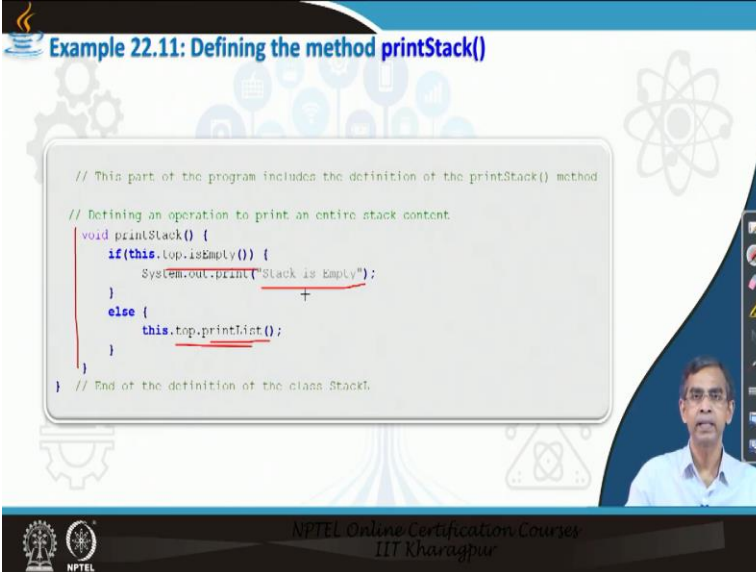
// The next method printStack() is to be defined here ... See the next slide
```

Now next operation is basically, is empty it is pretty simple it will basically check that if top is empty return true or false. So, that means here for this is empty method is defined for the linked list, if you do not define it you can write that if these dot top is equals to null then return true or false, that is also you can do.

I am assuming that is empty method is defined for the linked list, if it is not there you can do it whatever it is. So, we can actually what we are doing you can check it, we are using all the methods those have been implemented for linked list to implement the method for the stack, that is what actually, so that is why.

So we use the, is empty method which is defined for the linked list to define the is empty method for the stack, so that is the idea. Otherwise, you can just simply if top is equals to null then you can say the return to else false. So this also a valid operation that you can think about it.

(Refer Slide Time: 21:52)



**Example 22.11: Defining the method printStack()**

```
// This part of the program includes the definition of the printStack() method
// Defining an operation to print an entire stack content
void printStack() {
    if (this.top.isEmpty()) {
        System.out.println("Stack is Empty");
    }
    else {
        this.top.printList();
    }
}
// End of the definition of the class StackL
```

And our next operation for this I mean programming of stack using linked list is basically print stack method, it is just like a traversal one, that means printing the whole linked list actually or just we can go on what. Now we will let us see what is the idea about it, if this is not empty then obviously print list come into play.

So, that is why you have to check that whether the list is empty or not. So, if it is empty then it will print a message that it is empty, otherwise it will call the print list method. You can recall print list method once you have defined for the linked list structure. So, we call this method to print the entire list here and the it basically defines the print stack method.

So all the method, push, pop, is empty and print stack are defined. Now this completes the declaration of the stack class namely stack L using linked list. Now once the program is over, I mean stack class is defined completely we can again test the implementation by writing a master program.

(Refer Slide Time: 22:55)

**Example 22.12: Basic Stack operation using the class StackL**

```
/* This is the main program, illustration the usage of the class defined.
You should include the package, where this program is stored. */

class StackImplementationDemo {
    public static void main(String[] args) {
        StackL<Integer> st = new StackL<Integer>();

        st.push();
        st.printStack();
        st.push();
        st.push();
        st.printStack();
        st.pop();
        st.printStack();
        st.pop();
        st.printStack();
        st.pop();
    }
} // End of the demo class
```

NPTEL Online Certification Courses  
IIT Kharagpur

So, here is an example of a master program that I have declared here. Now here you can see this is we are declaring a stack st and then this is basically using linked list, so stack L and here the stack is to store the integer type, so that is why and these are the few method push, push, pop all this driver method that we have discussed one for testing like you have discussed also there and this includes the, this basically you can use, you can call the different methods in different orders and then you can test it. So, this basically completes the declaration of stack and testing this using both array as well as linked list.

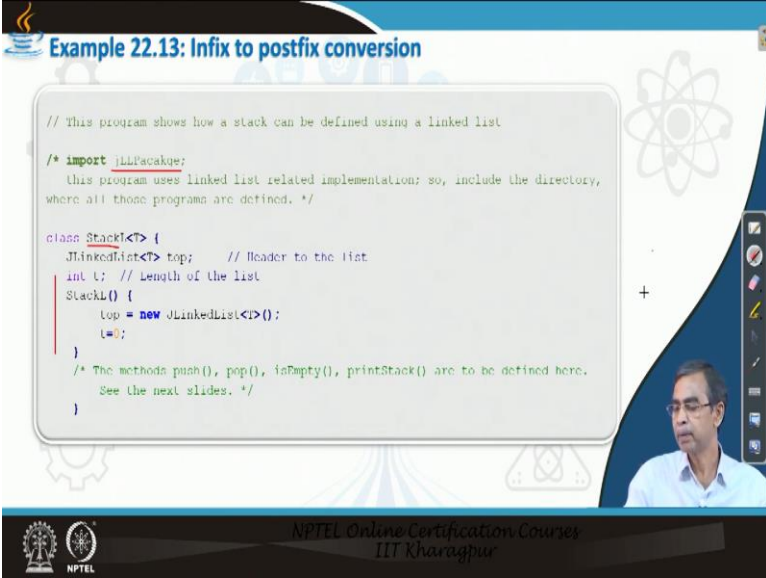
Now, our next important discussion will be application of stack. Now while we are discussing about stack that time we have mention that there are many applications on stack is possible. But we will not be able to discuss all applications rather we will limit to only one application, this application is basically related to evaluation of arithmetic expression. Arithmetic expression evaluation actually there are two steps two full method, the first is we have to convert an input expression into a postfix notation.

So, input expression usually it is infix notation, so converting an infix notation into a postfix notation. Now there is an algorithm for this conversion, for this algorithm we have to take the confidence of precedence and associativity of all the orders and I have also defined an algorithm that basically infix to postfix operation is basically depending on the elements to be pushed or

popped from a stack or to a stack depending on their priority of the precedence like this one. So, those algorithms you can follow.

So here, actually implementation of the algorithm or writing a program so far infix to postfix notation is basically maintenance stack and then scan each element into the expression namely infix expression and then check whether it we have to push or pop. Whatever the it is pop and everything this basically, result into the ultimate output operation. Now I have defined one program, I have already written one program for you, but discussing the entire program on step by step it is very tedious job.

(Refer Slide Time: 26:10)



The slide displays a C++ code snippet for a stack implemented using a linked list. The code includes a header file, a class definition for Stack, and a constructor. The code is as follows:

```
// This program shows how a stack can be defined using a linked list

/* import JLinkedList:
   this program uses linked list related implementation: so, include the directory,
   where all those programs are defined. */

class Stack<T> {
    JLinkedList<T> top; // Header to the list
    int l; // Length of the list
public:
    Stack() {
        top = new JLinkedList<T>();
        l = 0;
    }
    /* The methods push(), pop(), isEmpty(), printStack() are to be defined here.
       See the next slides. */
};
```

The slide also features a small video inset of a man in the bottom right corner and a footer with the NPTEL logo and text: "NPTEL Online Certification Courses IIT Kharagpur".

So, I should suggest you to check the program once you can write the program and then run it and check it every block of code you can check it actually, so that you can understand and while you are checking the code only going through the code cannot be, I mean it is not good to understand a program. So, side by side you can follow the algorithm for this method that may infix to suppose with that we have discussed in the previous discussion.

Now, what you have to do I am giving an idea about that how we can apply it now we will assume that in this program linked list implementation is for stack. So, whatever the method those are there for linked list implementation that you have to import here and then first we have to maintain a stack for this purpose, so this is the stack that we have here. So, this is the stack we have to make the stack ready for this program and then we can discuss about the method.

(Refer Slide Time: 27:20)

**Example 22.13: Infix to postfix conversion**

```
class InfixToPostfix {  
    // The method to return precedence of a given operator  
    // Higher returned value means higher precedence  
  
    static int precedence(char ch)  
    {  
        switch (ch) {  
            case '+':  
            case '-':  
                return 1;  
  
            case '*':  
            case '/':  
                return 2;  
  
            case '^':  
                return 3;  
        }  
        return -1;  
    }  
};
```

NPTEL Online Certification Course  
IIT Kharagpur

Now this method has few block of code the first actually, for every what is called the symbol in your expression you have to just return its precedence or priority value. Now we will assume that symbol that is there in an expression is either a character or operator that mean plus minus and everything and here this basically, all this priority or precedence of all these values is there. Here for others is basically this one, either mean starting brackets or ending brackets is basically there.

So those are basically a single character as a operant and these are the operator we can assume that our expression includes this one only. Then fine, then we just, so this block here, this block or we can say this is one function we can say, who is basically, starting a symbol it will return what is its priority value that is all. So, this is a first what is called a block that is towards the infix to postfix conversion it is require.



(Refer Slide Time: 28:33)

**Example 22.13: Infix to postfix conversion**

```
// The method that converts given infix expression to postfix expression.
static String infixToPostfix(string exp) {
    // initializing empty string to store result.
    String result = new String("");
    // initializing empty stack
    Stack<Character> stack = new Stack<>();

    for (int i = 0; i < exp.length(); ++i) {
        char c = exp.charAt(i);

        // If the scanned character is an operand, add it to output.
        if (Character.isLetterOrDigit(c))
            result += c;

        // If the scanned character is an '(', push it to the stack.
        else if (c == '(')
            stack.push(c);
    }
}
```

NPTEL Online Certification Courses  
IIT Kharagpur

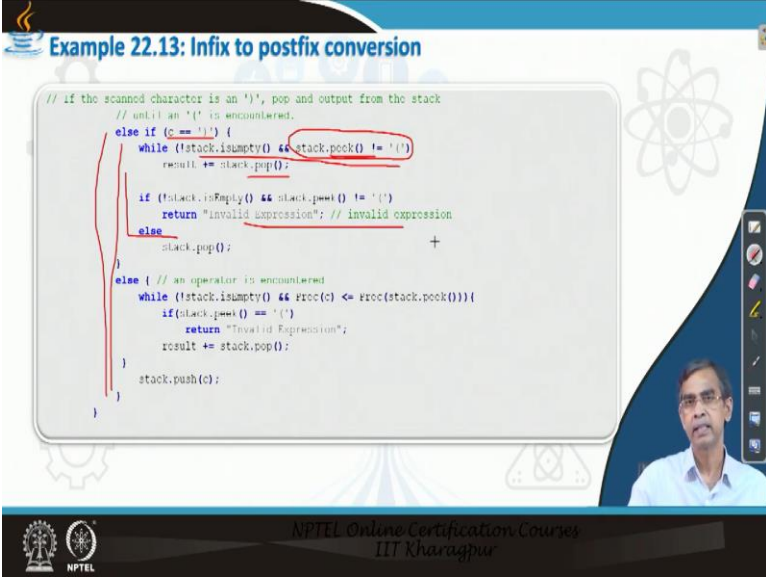
And the next part is basically implementation of the algorithm that means when you just read a symbol and then depending on its precedence compared to in stack one symbol whether it needs to be pushed or pop and accordingly you have to just call it is like this here. So here, this is the method that we are going to discuss about infix to postfix expression evaluation and this basically results, so I told you while we are processing this it basically output towards the postfix notation actually there.

Now initially we will assume that our expression is stored in the form of a stack and then it will contains the least actually, so this basically stack is basically is an linked list implementation and here and then you see for loop will continue, we have other parts in the next slides also. So expression dot length, expression is basically here, the expression means our input expression in infix notation rather.

So, we have to continue starting from the first what is called the symbol in the expression and then we have to take the character value because this is a character symbol that we have to take it and then is later to digit so if it is a character symbol we have to convert into the digit form like this one, so whatever it is here. Now here if you see, if the starting bracket in your input expression it is the push, this is as per the algorithm if you can follow then you will be follow

that, once you scan each elements into it and then what to do, either push or pop and the storing into the result into this array.

(Refer Slide Time: 30:20)



**Example 22.13: Infix to postfix conversion**

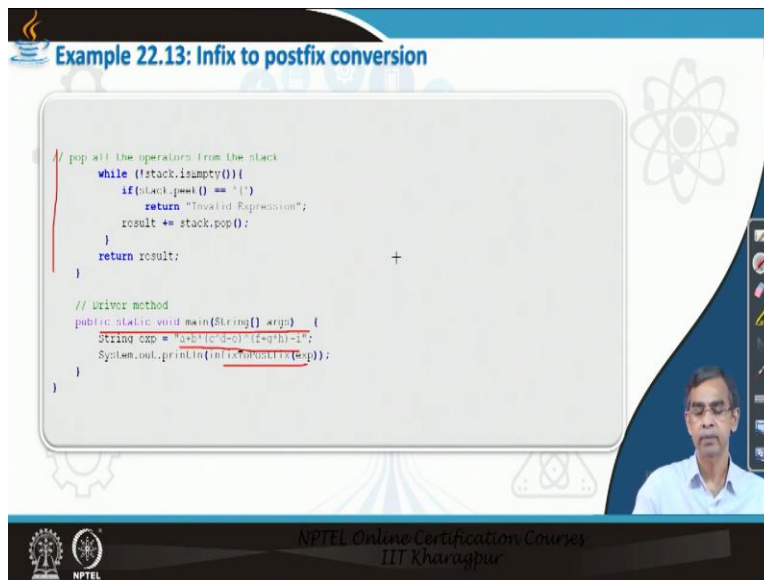
```
// if the scanned character is an ')', pop and output from the stack
// until an '(' is encountered.
else if (c == ')') {
    while (!stack.isEmpty() && stack.peek() != '(')
        result += stack.pop();
    if (!stack.isEmpty() && stack.peek() != '(')
        return "Invalid expression"; // invalid expression
    else
        stack.pop(); // +
}
else { // an operator is encountered
    while (!stack.isEmpty() && prec(c) <= prec(stack.peek())){
        if(stack.peek() == '(')
            return "Invalid Expression";
        result += stack.pop();
    }
    stack.push(c);
}
}
```

NPTEL Online Certification Courses  
IIT Kharagpur

So, this is the part of the program it is continue and the different situation is basically will be implemented using if and else statement like. If c is this one and stack is not empty then your peek it is there, peak is basically one method is that it will check only top most element but we will not remove it actually. That peek method you can write you have not defined it but you can easily do that then in that case the stack to a, that element to be popped and this one and if it does not follow according the correct valid expression then it will check it also whatever.

So, this basically complete implementation of the program related to algorithm related to infix to confix it is there. So, if you follow the algorithm there and implement it, it is enough to understand this program. Now once this method is there we can test this method by writing a driver program and then you can pass any input valid infix expression or even invalid also no problem, this algorithm this program will take care, it basically convert into a postfix notation.

(Refer Slide Time: 31:26)



### Example 22.13: Infix to postfix conversion

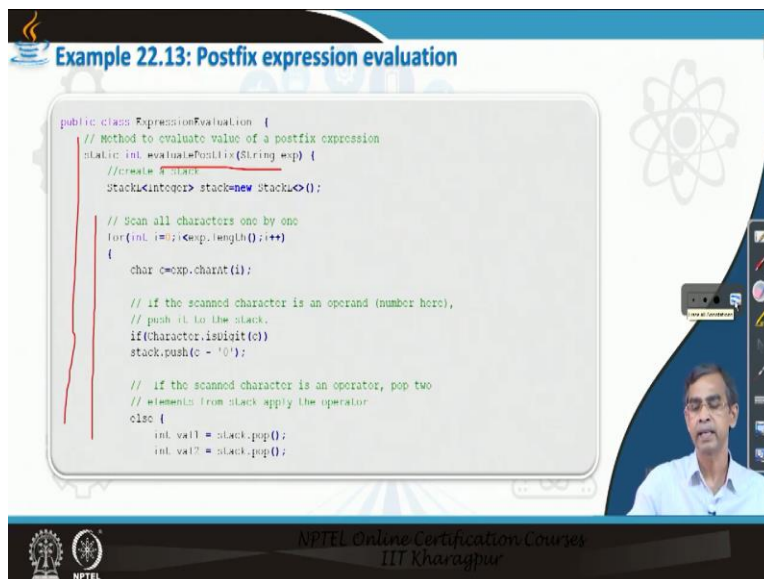
```
// pop all the operators from the stack
while (!stack.isEmpty()) {
    if (stack.peek() == '(')
        return "Invalid Expression";
    result += stack.pop();
}
return result;

// Driver method
public static void main(String[] args) {
    String exp = "a+b*(c-d)-(e+f)-1";
    System.out.println(infixToPostfix(exp));
}
```

NPTEL Online Certification Courses  
IIT Kharagpur

So, this is the one program it is there and so this completes the program and here you see the driver program master program expression is given like this and it basically print the result, expression is basically result it is there and it will print. So, you can see how the program that basically infix to postfix conversion takes place. Now so I just advice you to check the program write the code in your machine and then you can run it. Our next part one is the postfix form of the expression is available will be able to apply this program to evaluate and expression.

(Refer Slide Time: 32:11)



### Example 22.13: Postfix expression evaluation

```
public class ExpressionEvaluation {
    // Method to evaluate value of a postfix expression
    static int evaluatePostfix(String exp) {
        // create a stack
        Stack<Integer> stack = new Stack<>();

        // Scan all characters one by one
        for (int i = 0; i < exp.length(); i++) {
            char c = exp.charAt(i);

            // if the scanned character is an operand (number here),
            // push it to the stack.
            if (Character.isDigit(c))
                stack.push(c - '0');

            // if the scanned character is an operator, pop two
            // elements from stack apply the operator
            else {
                int val1 = stack.pop();
                int val2 = stack.pop();
            }
        }
    }
}
```

NPTEL Online Certification Courses  
IIT Kharagpur

So, our next part is basically evaluating the postfix expression and this is the program for the purpose. Here we can see this is the program, this program basically define a method and this again body, body as per the algorithm that we have already discussed earlier. So that is, I do not want to discuss it, it is basically depending upon the algorithm either push and pop operation it is there so this will basically do it.

(Refer Slide Time: 32:44)

```
switch(c) {
    case '+':
        stack.push(stack.pop()+stack.pop());
        break;

    case '*':
        stack.push(stack.pop()* stack.pop());
        break;

    case '/':
        stack.push(stack.pop()/stack.pop());
        break;

    case '=':
        stack.push(stack.pop());
        break;
}

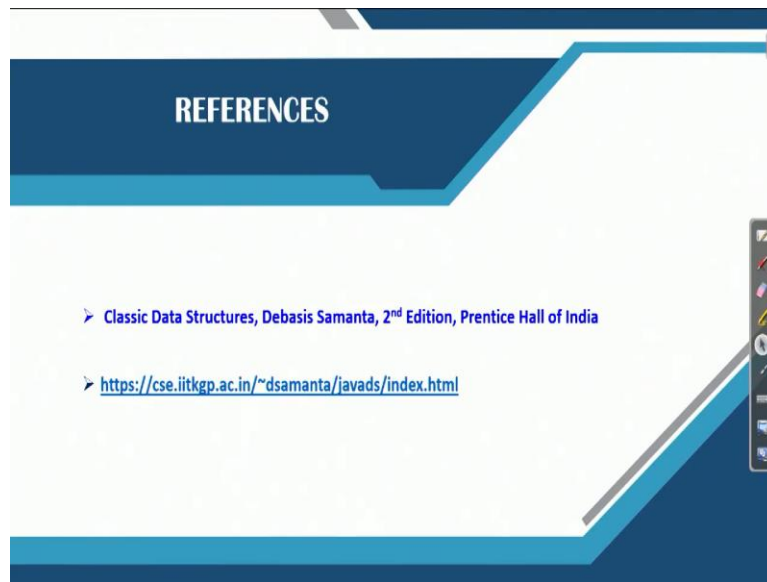
return stack.pop();

// driver program to test above functions
public static void main(String[] args) {
    String exp="(2+3)*1";
    System.out.println("postfix evaluation: "+evalPostfix(exp));
}
```

And then, finally are the main method we can call it this is a main method. In this main method if you see expression will be given, here we will assume that all the expression will be express in terms of single digit apparent that is 2, 3, 1 star like this one and then this expression is in postfix notation and then you can call otherwise, one thing is that you can just call in this case you can give any input expression in this method and then call infix to postfix expression here and then carry on.

So, it will basically starting from infix notation of an expression and finally it is evaluating that also you can do it. So, you can just simply do many experiments with this program you can modify and then according to your understanding you can change the code and then you can test it and that is obviously useful to run the program much better.

(Refer Slide Time: 33:46)



So, okay fine this is we have discuss on application. There may be many more applications you can find the book which I have referred here and all the programs which I have followed in this lecture slide also you can find from this link. Thank you very much.