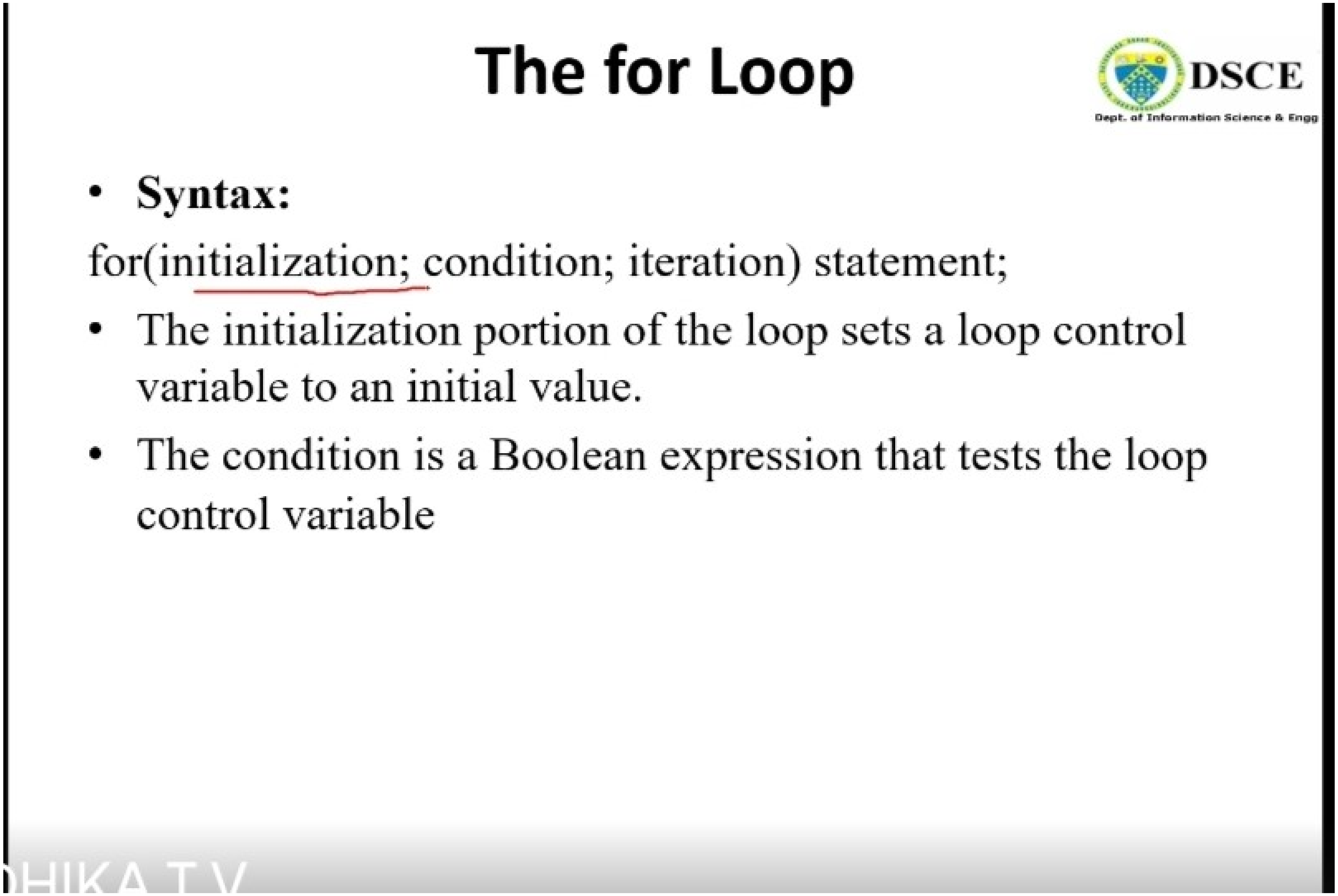


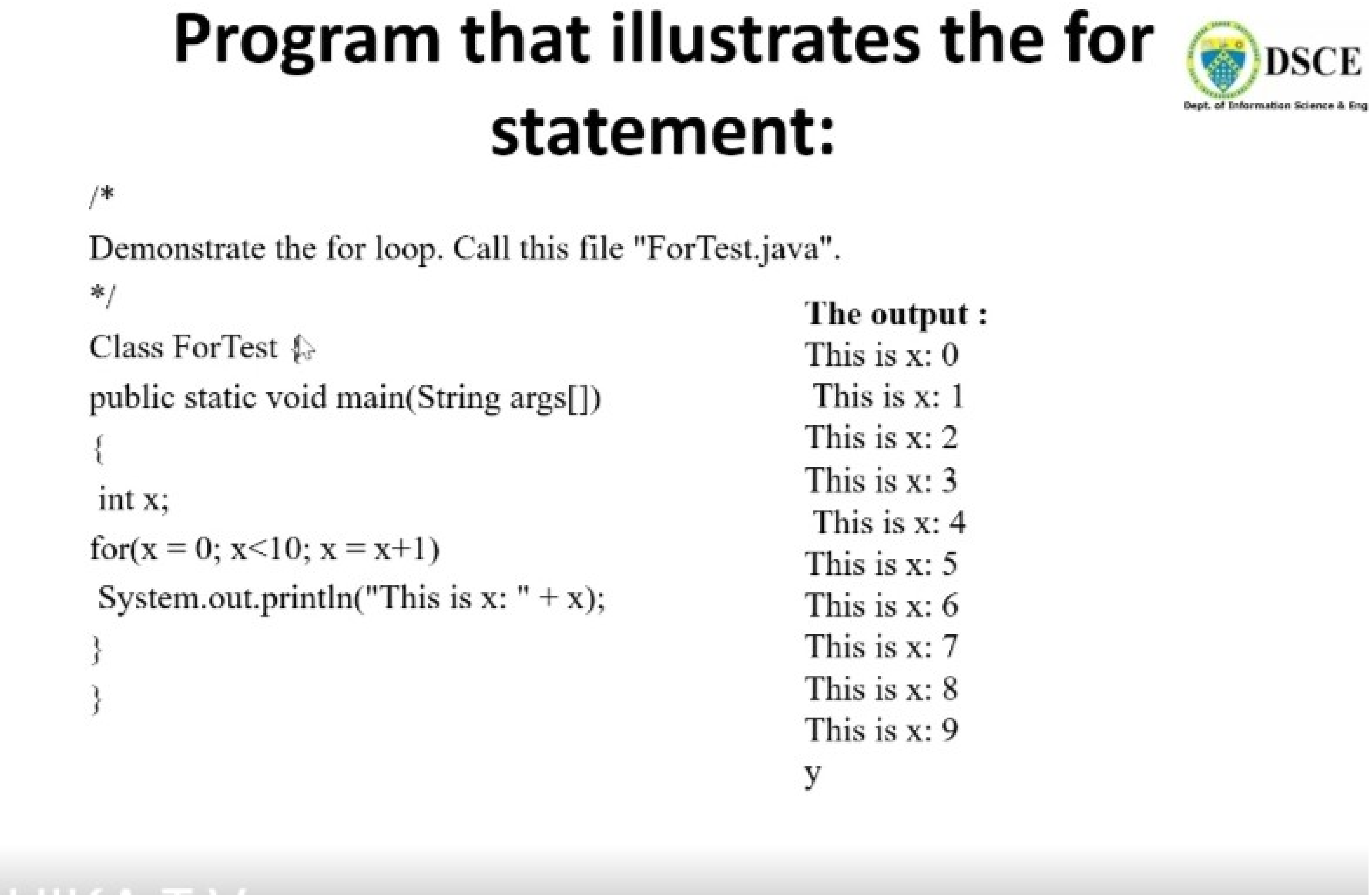
The for Loop

- **Syntax:**

for(initialization; condition; iteration) statement;

- The initialization portion of the loop sets a loop control variable to an initial value.
- The condition is a Boolean expression that tests the loop control variable

DHIKA T V



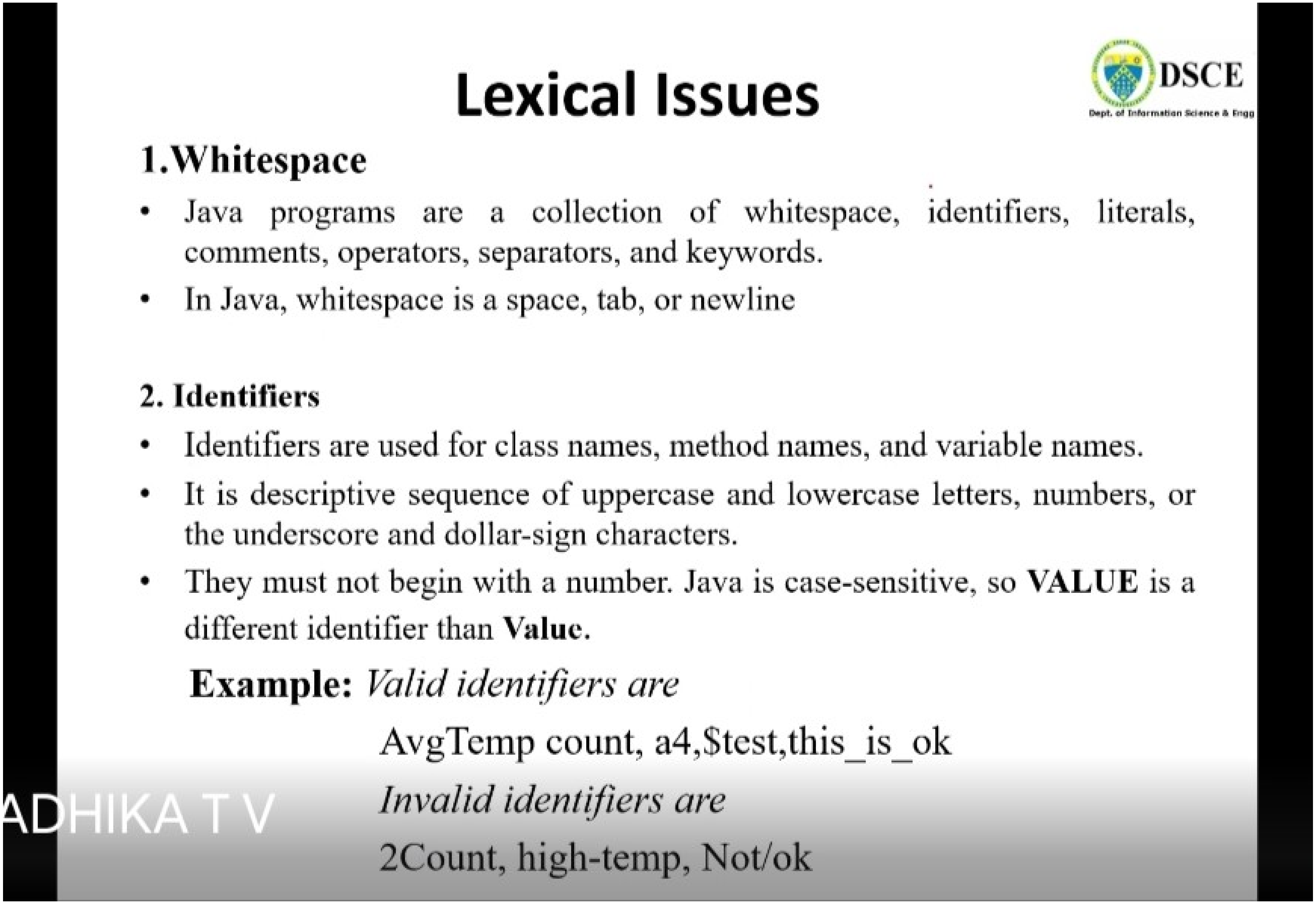
Program that illustrates the for statement:

DSCE
Dept. of Information Science & Eng

```
/*
Demonstrate the for loop. Call this file "ForTest.java".
*/
Class ForTest {
public static void main(String args[])
{
 int x;
for(x = 0; x<10; x = x+1)
 System.out.println("This is x: " + x);
}
}
```

The output :

```
This is x: 0
 This is x: 1
This is x: 2
This is x: 3
 This is x: 4
This is x: 5
This is x: 6
This is x: 7
This is x: 8
This is x: 9
y
```



Lexical Issues

1.Whitespace

- Java programs are a collection of whitespace, identifiers, literals, comments, operators, separators, and keywords.
- In Java, whitespace is a space, tab, or newline

2. Identifiers

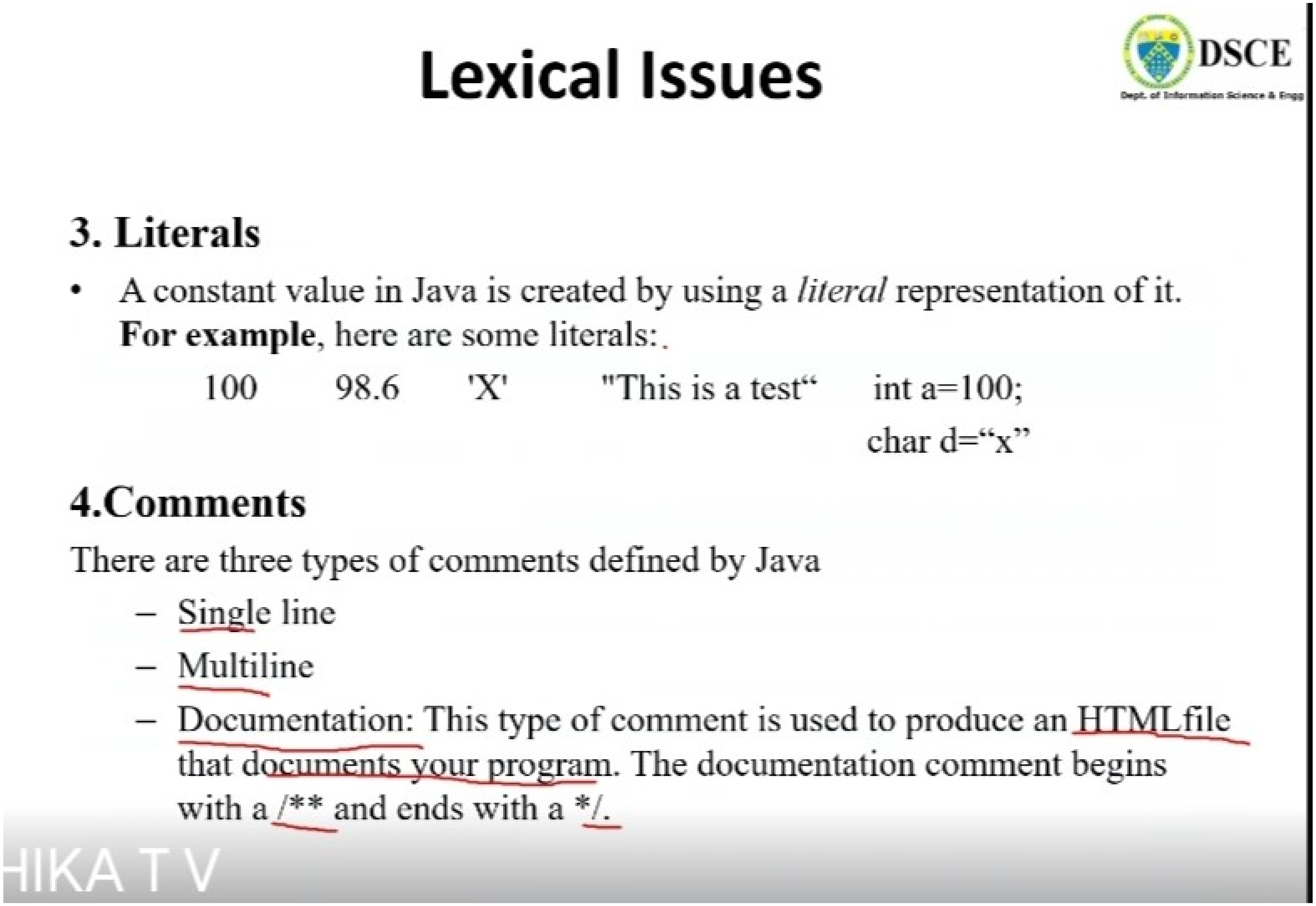
- Identifiers are used for class names, method names, and variable names.
- It is descriptive sequence of uppercase and lowercase letters, numbers, or the underscore and dollar-sign characters.
- They must not begin with a number. Java is case-sensitive, so **VALUE** is a different identifier than **Value.**

Example: *Valid identifiers are*

AvgTemp count, a4,$test,this_is_ok

Invalid identifiers are

2Count, high-temp, Not/ok



Lexical Issues

3. Literals

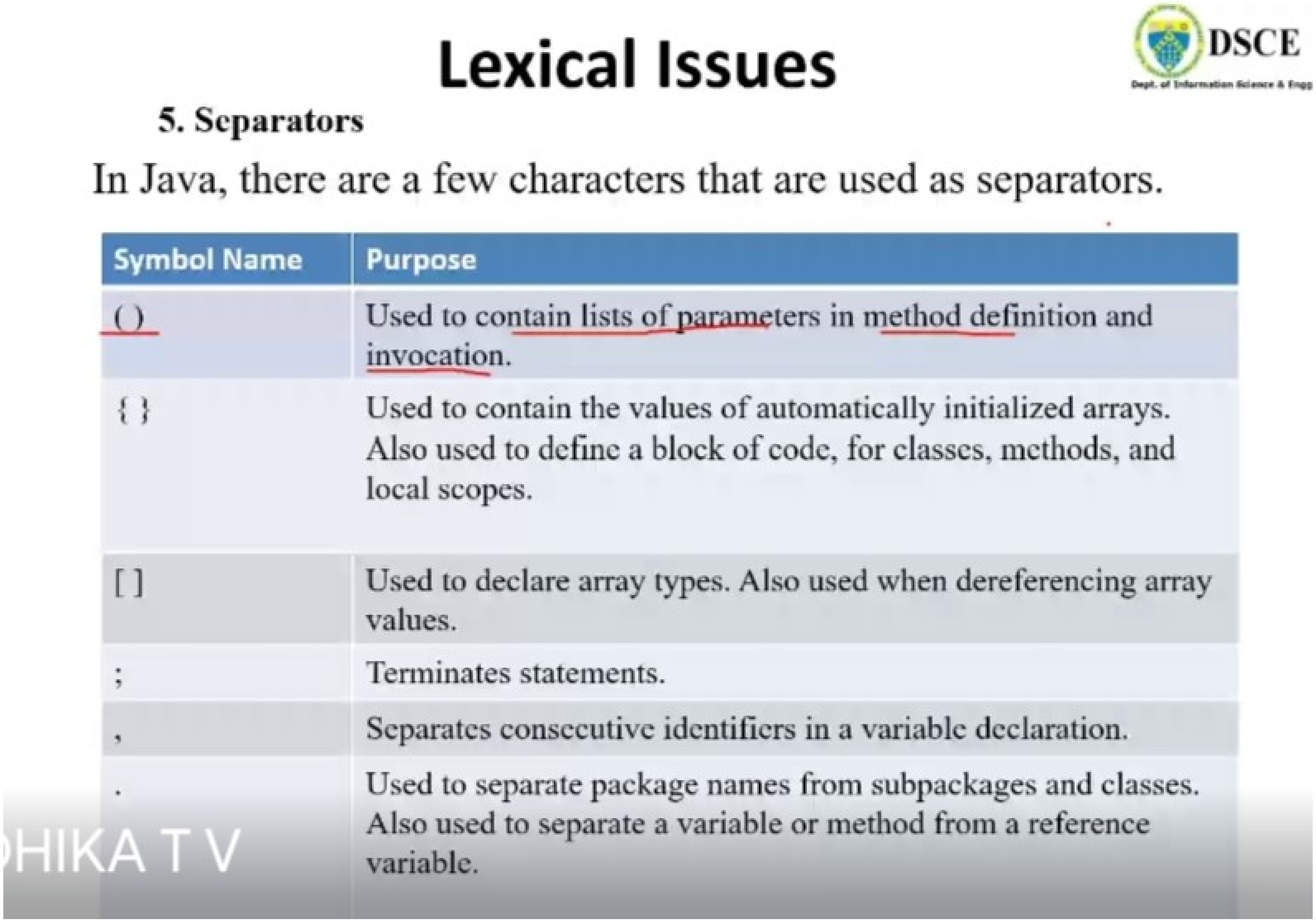
- A constant value in Java is created by using a *literal* representation of it. **For example**, here are some literals:.

100 98.6 'X' "This is a test“ int a=100;
char d=“x”

4.Comments

There are three types of comments defined by Java

- Single line
- Multiline
- Documentation: This type of comment is used to produce an HTMLfile that documents your program. The documentation comment begins with a /** and ends with a */.

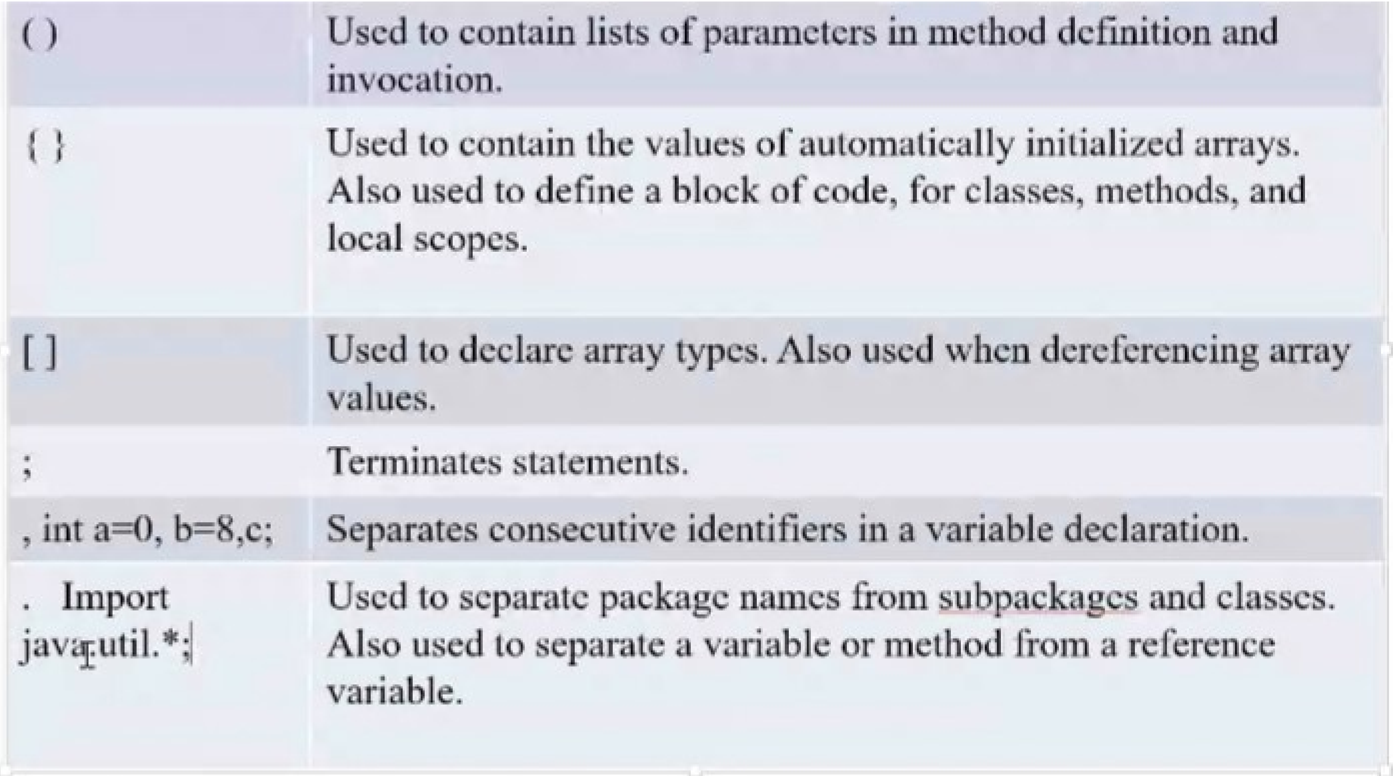


Lexical Issues

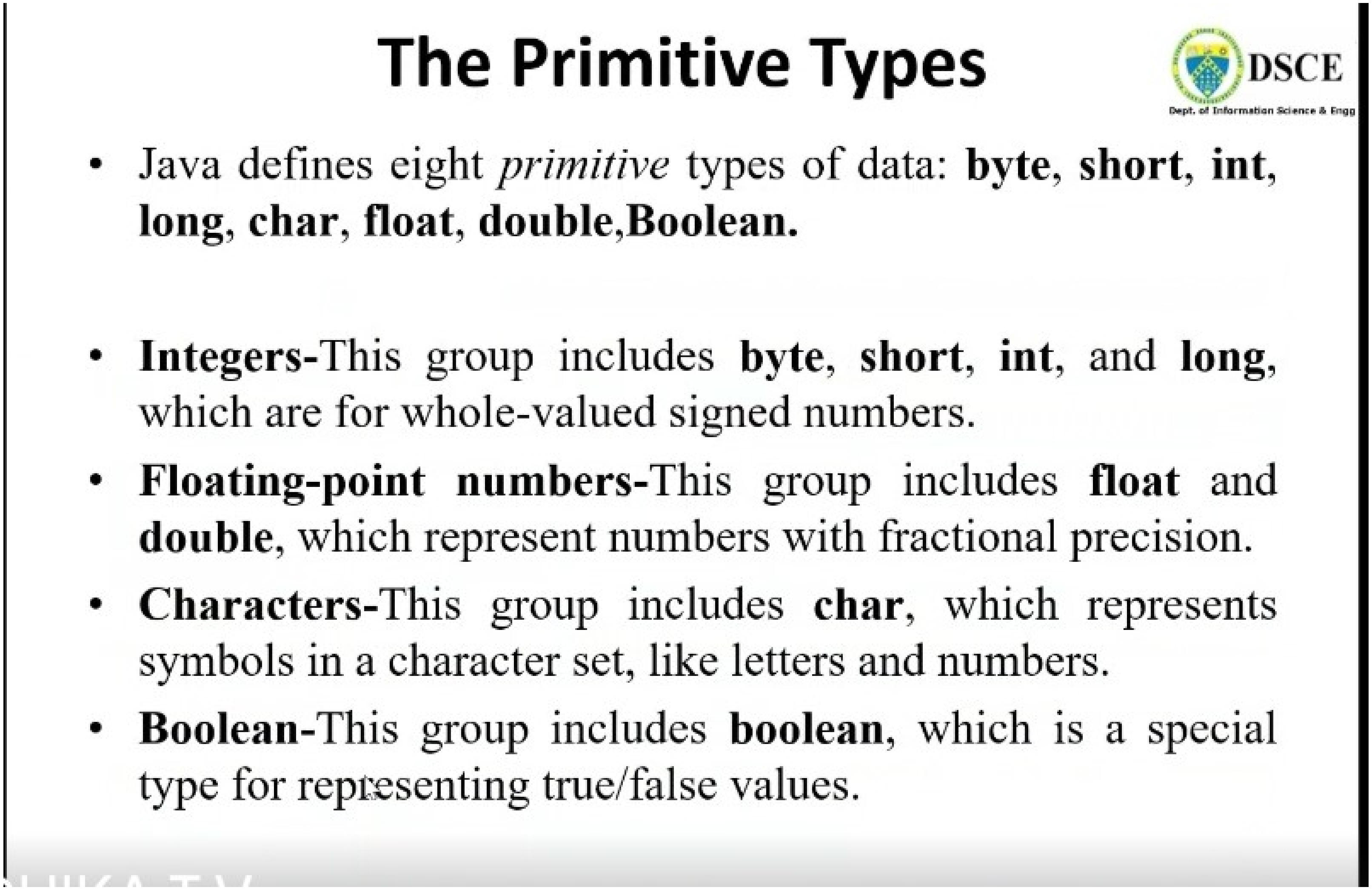
5. Separators

In Java, there are a few characters that are used as separators.

Symbol Name	Purpose
( )	Used to contain lists of parameters in method definition and invocation.
{ }	Used to contain the values of automatically initialized arrays. Also used to define a block of code, for classes, methods, and local scopes.
[ ]	Used to declare array types. Also used when dereferencing array values.
;	Terminates statements.
,	Separates consecutive identifiers in a variable declaration.
.	Used to separate package names from subpackages and classes. Also used to separate a variable or method from a reference variable.



( )	Used to contain lists of parameters in method definition and invocation.
{ }	Used to contain the values of automatically initialized arrays. Also used to define a block of code, for classes, methods, and local scopes.
[ ]	Used to declare array types. Also used when dereferencing array values.
;	Terminates statements.
, int a=0, b=8,c;	Separates consecutive identifiers in a variable declaration.
. Import java.util.*;	Used to separate package names from subpackages and classes. Also used to separate a variable or method from a reference variable.



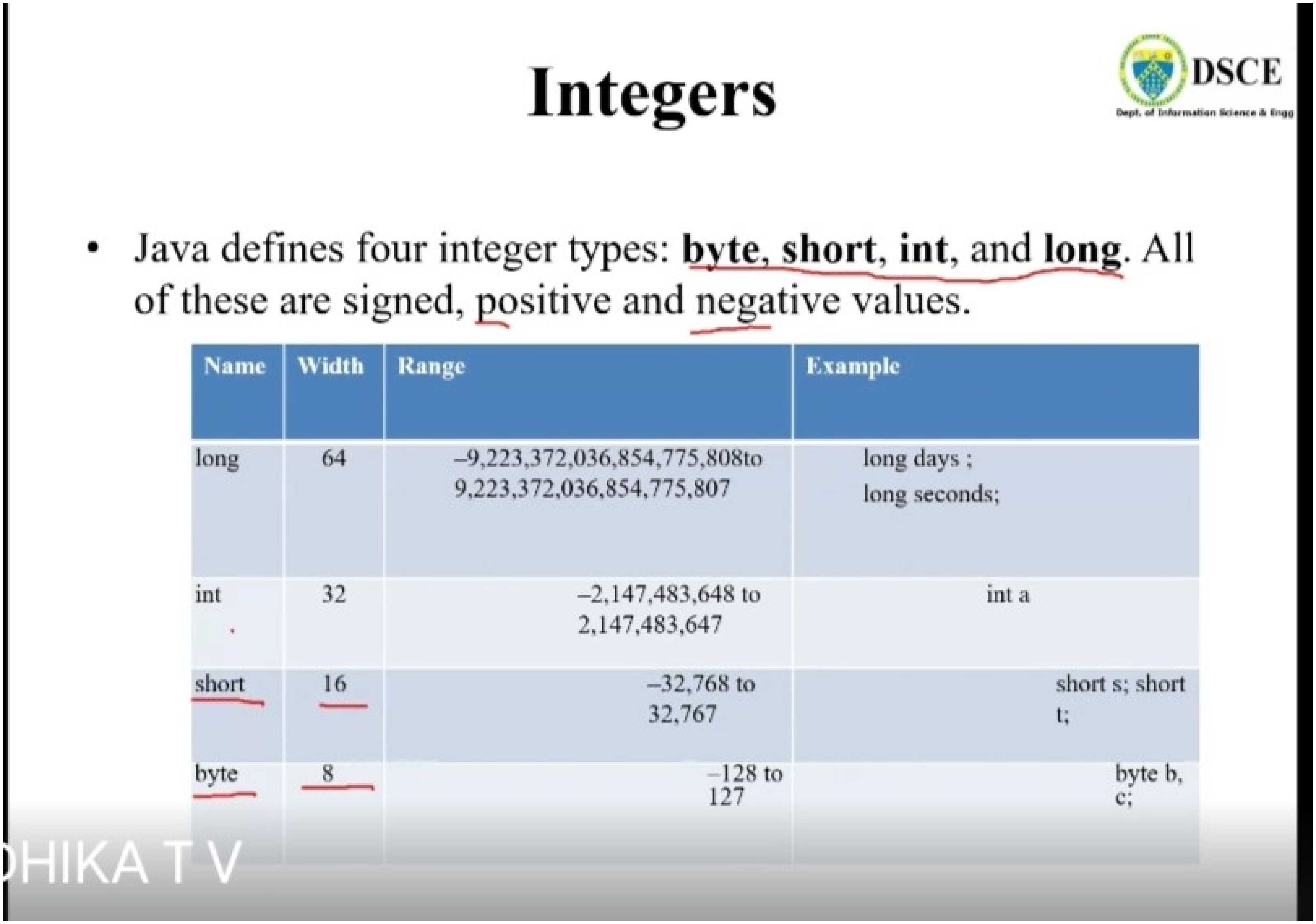
The Primitive Types

DSCE
Dept. of Information Science & Engg

- Java defines eight *primitive* types of data: **byte**, **short**, **int**, **long**, **char**, **float**, **double**,**Boolean.**

- **Integers-**This group includes **byte**, **short**, **int**, and **long**, which are for whole-valued signed numbers.
- **Floating-point numbers-**This group includes **float** and **double**, which represent numbers with fractional precision.
- **Characters-**This group includes **char**, which represents symbols in a character set, like letters and numbers.
- **Boolean-**This group includes **boolean**, which is a special type for representing true/false values.

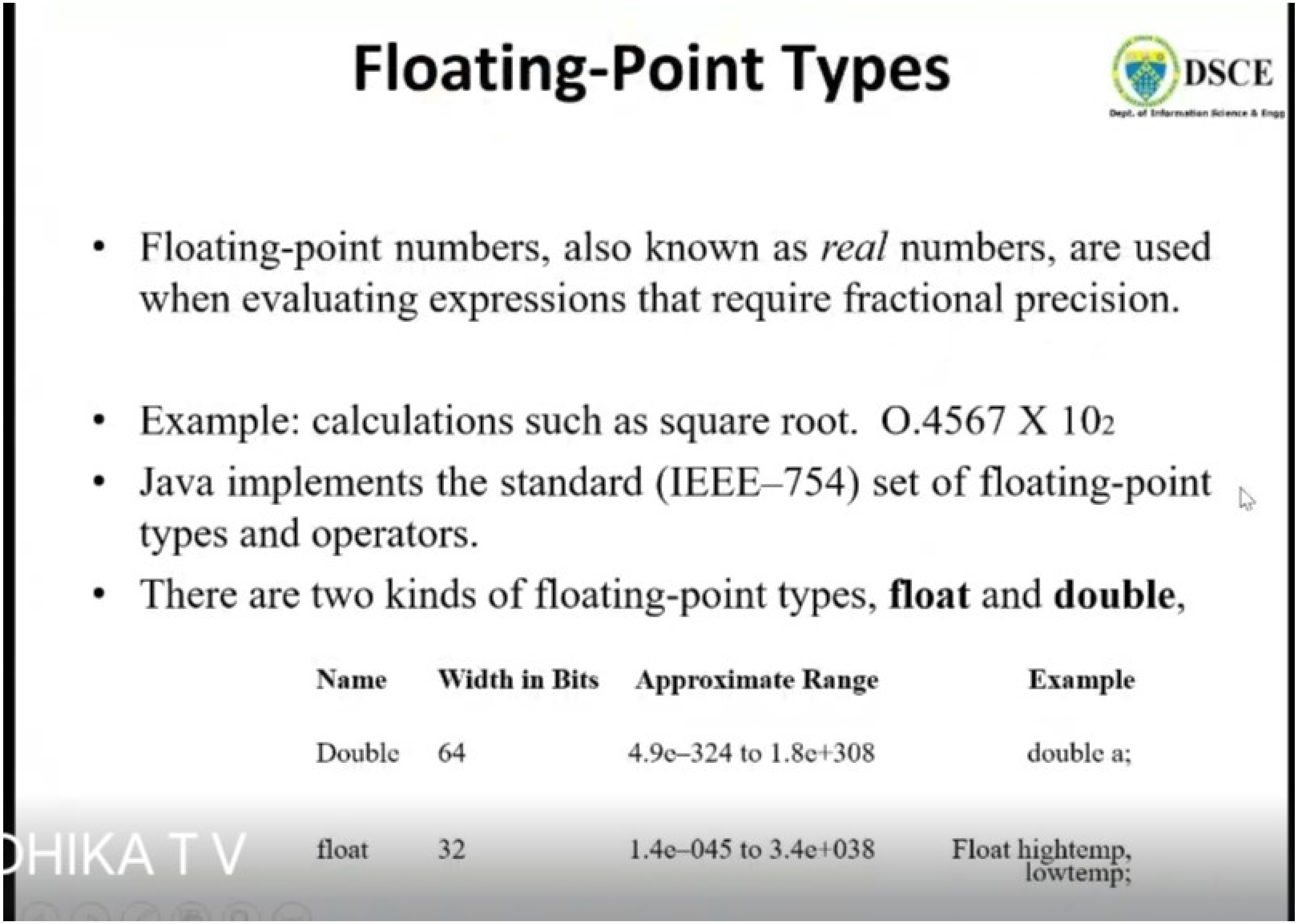
NIKA T.V



Integers

- Java defines four integer types: **byte, short, int,** and **long**. All of these are signed, positive and negative values.

Name	Width	Range	Example
long	64	–9,223,372,036,854,775,808to 9,223,372,036,854,775,807	long days ; long seconds;
int	32	–2,147,483,648 to 2,147,483,647	int a
short	16	–32,768 to 32,767	short s; short t;
byte	8	–128 to 127	byte b, c;

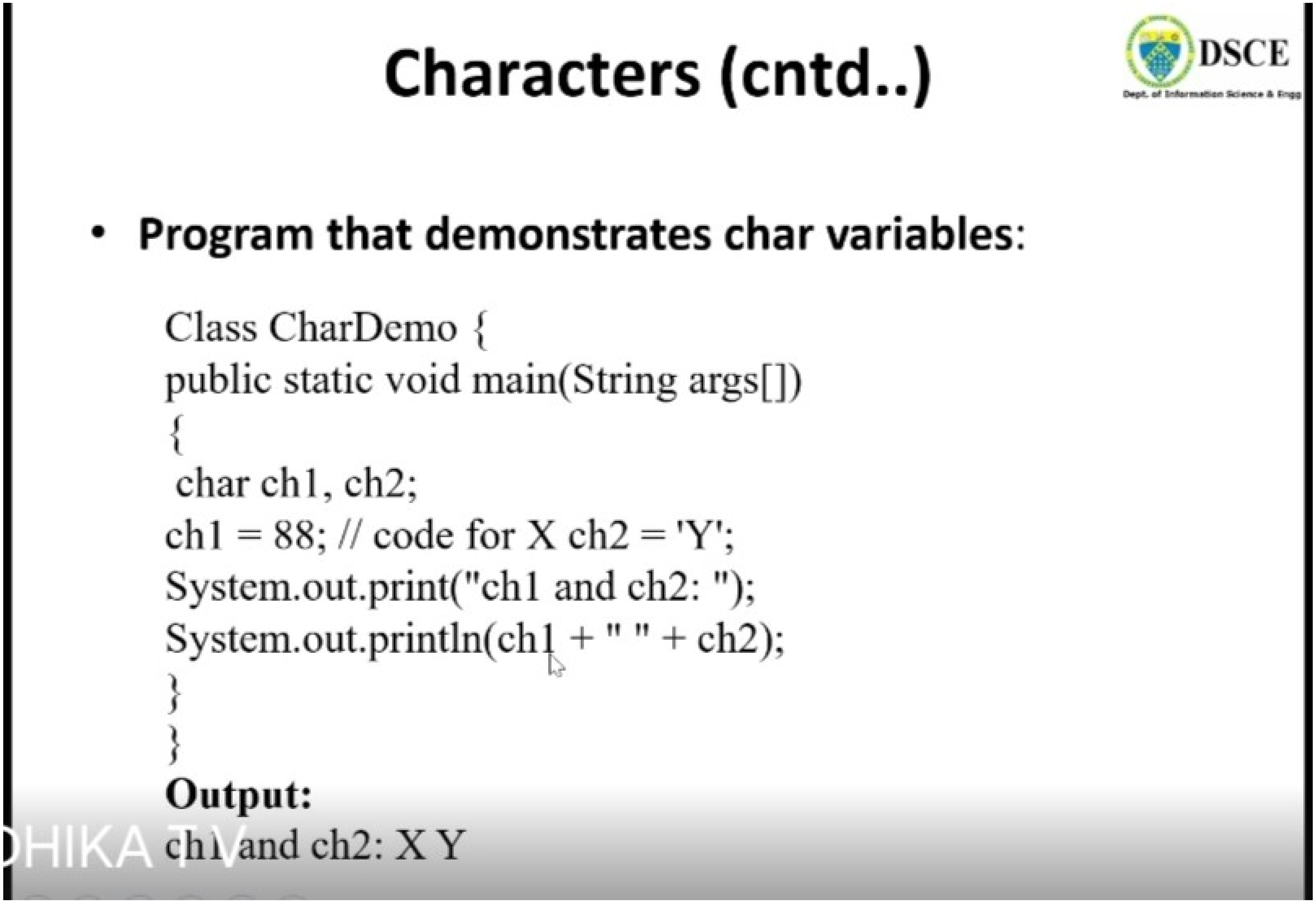


Floating-Point Types

- Floating-point numbers, also known as *real* numbers, are used when evaluating expressions that require fractional precision.

- Example: calculations such as square root. O.4567 X 10_2
- Java implements the standard (IEEE–754) set of floating-point types and operators.
- There are two kinds of floating-point types, **float** and **double**,

Name	Width in Bits	Approximate Range	Example
Double	64	4.9e–324 to 1.8e+308	double a;
float	32	1.4e–045 to 3.4e+038	Float hightemp, lowtemp;



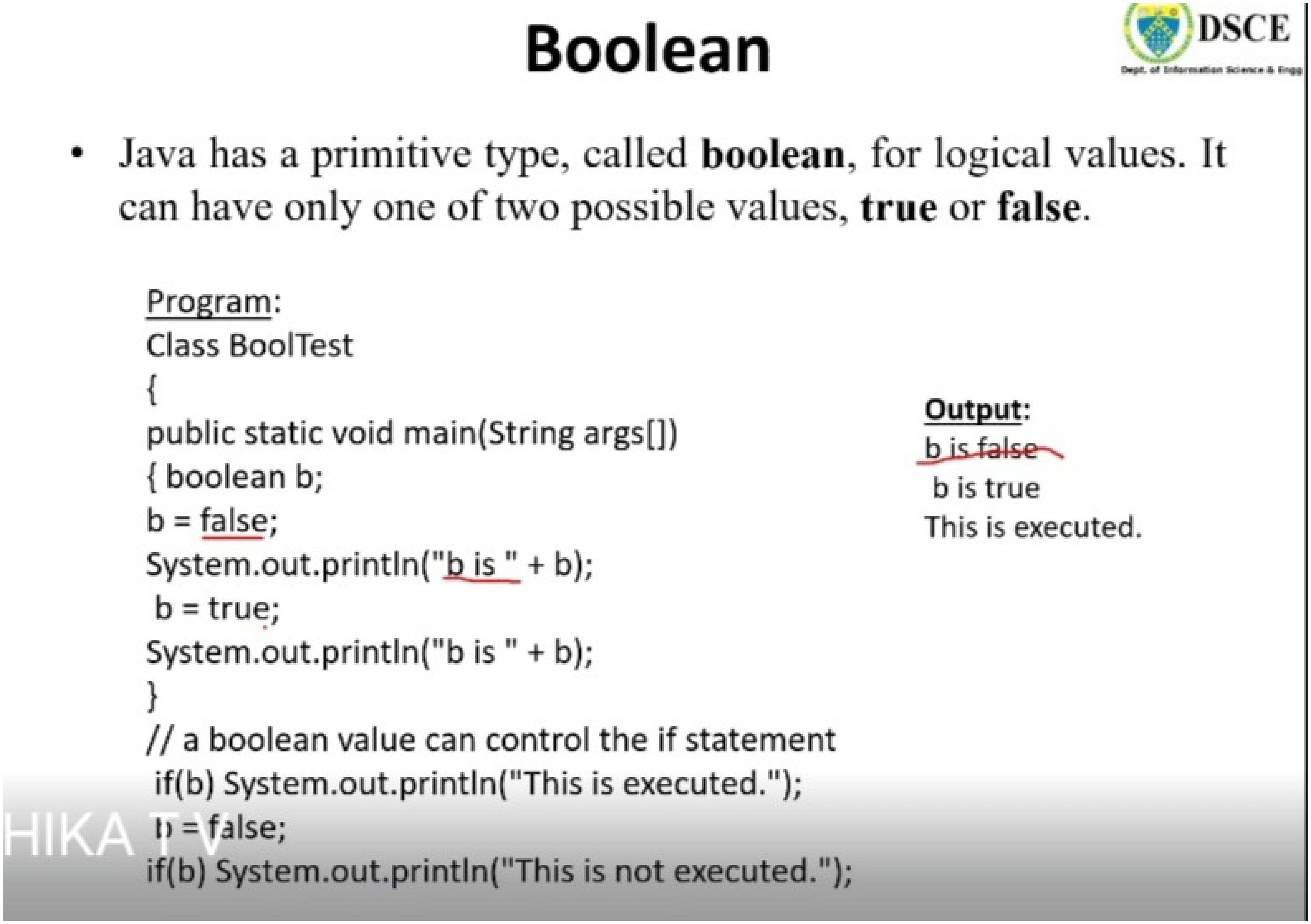
Characters (cntd..)

- **Program that demonstrates char variables**:

```
Class CharDemo {
public static void main(String args[])
{
 char ch1, ch2;
ch1 = 88; // code for X ch2 = 'Y';
System.out.print("ch1 and ch2: ");
System.out.println(ch1 + " " + ch2);
}
}
```

Output:

```
ch1 and ch2: X Y
```



Boolean

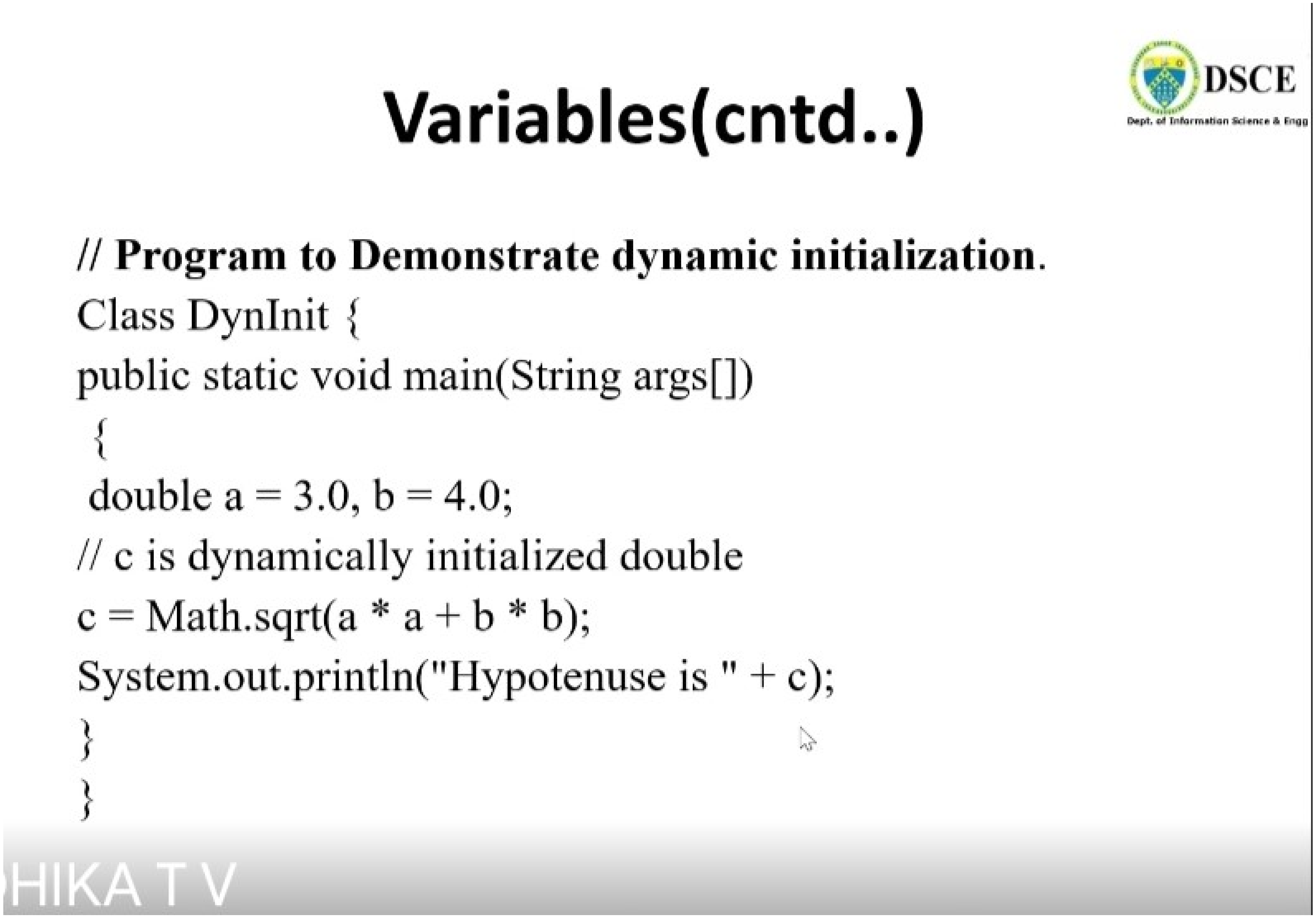
- Java has a primitive type, called **boolean**, for logical values. It can have only one of two possible values, **true** or **false**.

Program:

```
Class BoolTest
{
public static void main(String args[])
{ boolean b;
b = false;
System.out.println("b is " + b);
 b = true;
System.out.println("b is " + b);
}
// a boolean value can control the if statement
 if(b) System.out.println("This is executed.");
 b = false;
if(b) System.out.println("This is not executed.");
```

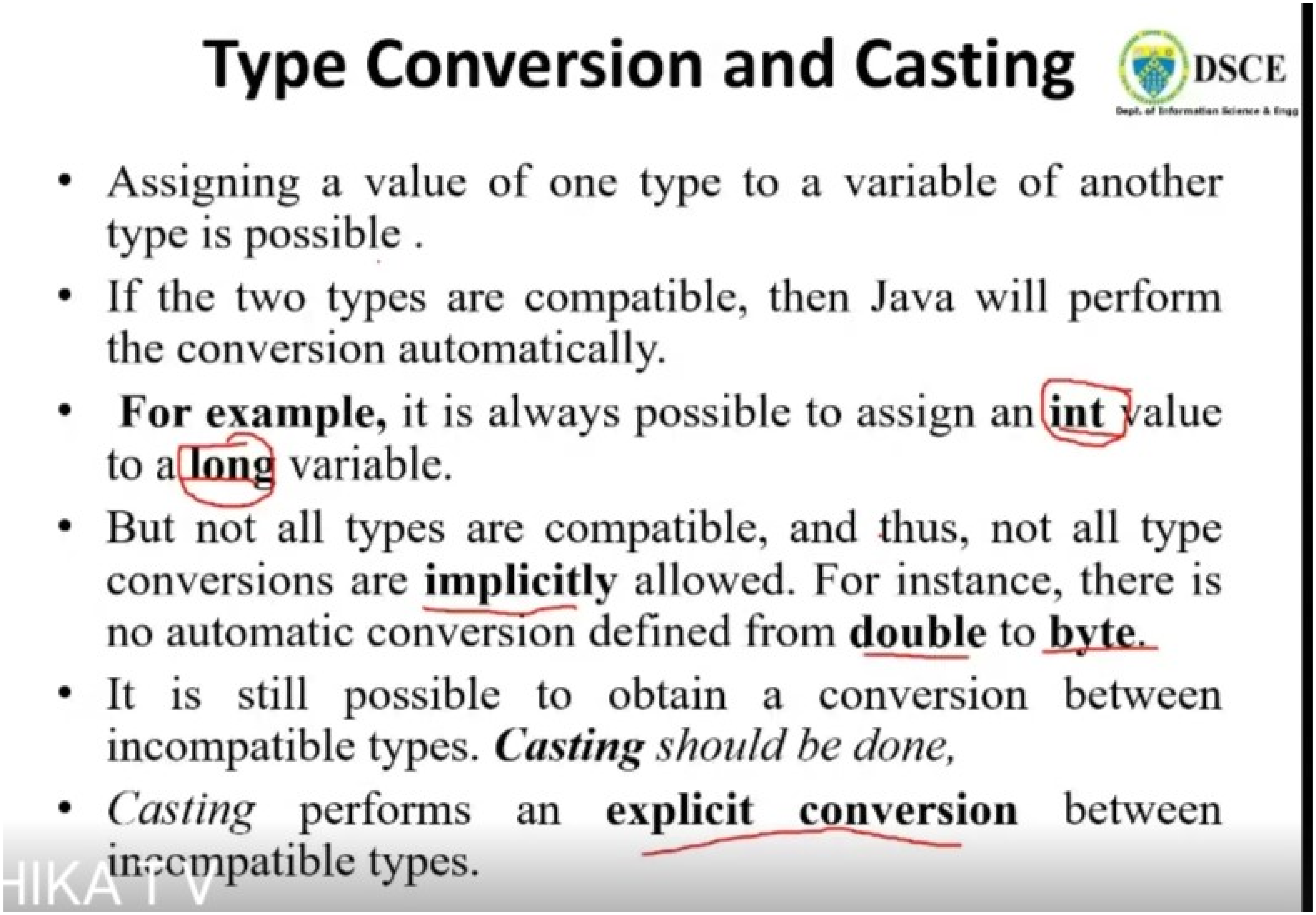
Output:

```
b is false
 b is true
This is executed.
```



Variables(cntd..)

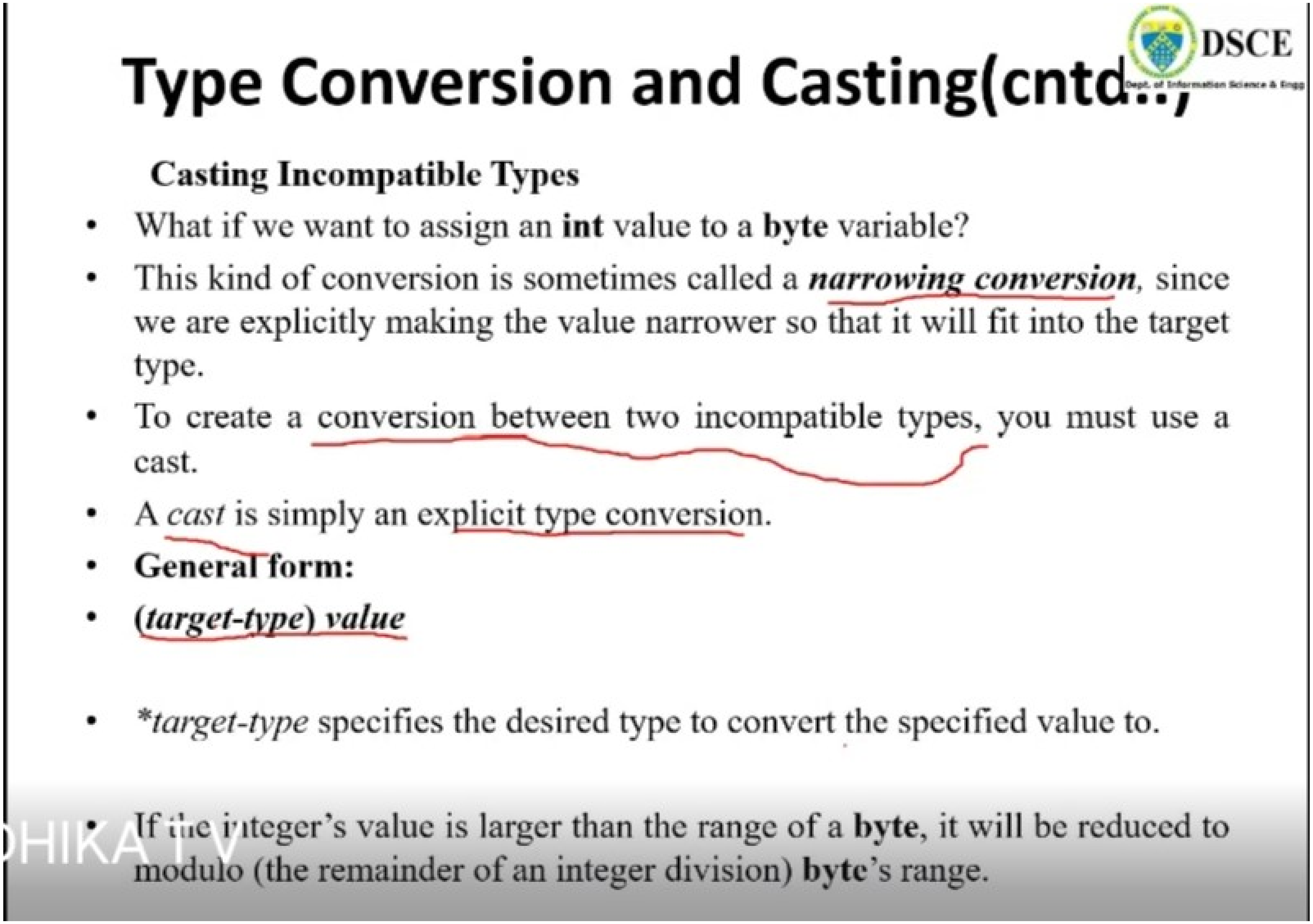
```
// Program to Demonstrate dynamic initialization.
Class DynInit {
public static void main(String args[])
 {
 double a = 3.0, b = 4.0;
// c is dynamically initialized double
c = Math.sqrt(a * a + b * b);
System.out.println("Hypotenuse is " + c);
}
}
```



Type Conversion and Casting

DSCE
Dept. of Information Science & Engg

- Assigning a value of one type to a variable of another type is possible .
- If the two types are compatible, then Java will perform the conversion automatically.
- **For example,** it is always possible to assign an **int** value to a **long** variable.
- But not all types are compatible, and thus, not all type conversions are **implicitly** allowed. For instance, there is no automatic conversion defined from **double** to **byte**.
- It is still possible to obtain a conversion between incompatible types. ***Casting*** *should be done,*
- *Casting* performs an **explicit conversion** between incompatible types.

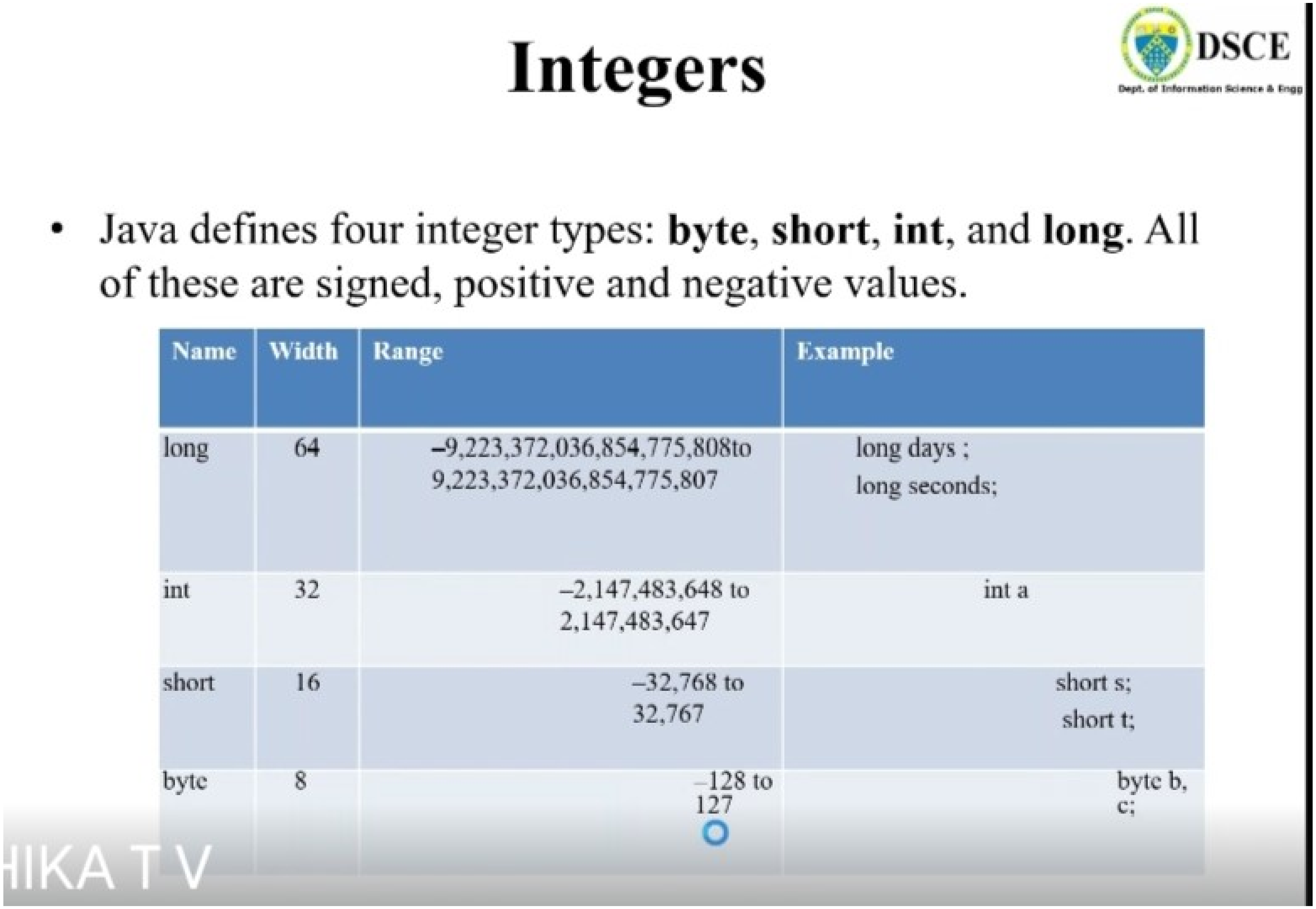


Type Conversion and Casting(cntd..)

DSCE
Dept. of Information Science & Engg

Casting Incompatible Types

- What if we want to assign an **int** value to a **byte** variable?
- This kind of conversion is sometimes called a ***narrowing conversion***, since we are explicitly making the value narrower so that it will fit into the target type.
- To create a conversion between two incompatible types, you must use a cast.
- A *cast* is simply an explicit type conversion.
- **General form:**
- **(*target-type*) *value***
- **target-type* specifies the desired type to convert the specified value to.
- If the integer's value is larger than the range of a **byte**, it will be reduced to modulo (the remainder of an integer division) **byte**'s range.



Integers

DSCE

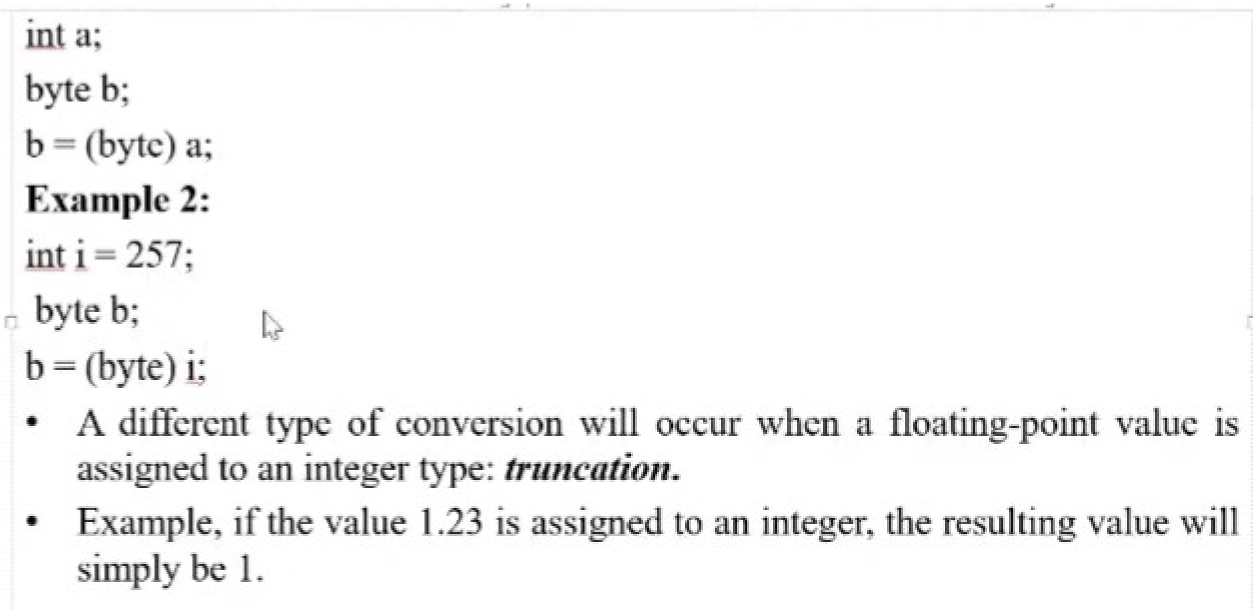
Dept. of Information Science & Engg

- Java defines four integer types: **byte**, **short**, **int**, and **long**. All of these are signed, positive and negative values.

Name	Width	Range	Example
long	64	–9,223,372,036,854,775,808to 9,223,372,036,854,775,807	long days ; long seconds;
int	32	–2,147,483,648 to 2,147,483,647	int a

short	16	–32,768 to 32,767	short s;
			short t;

byte	8	–128 to 127	byte b, c;

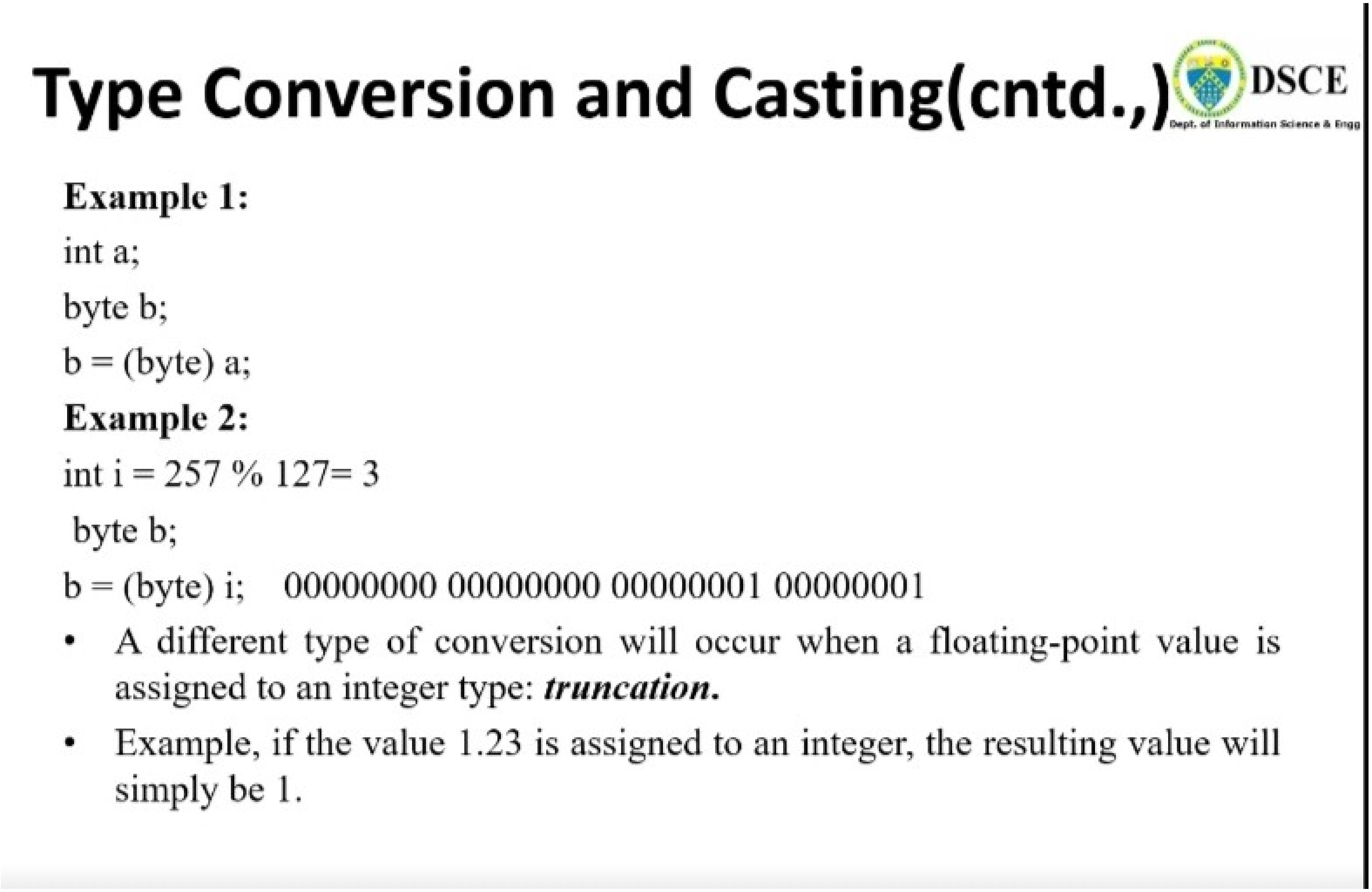


```
int a;
byte b;
b = (byte) a;
```

Example 2:

```
int i = 257;
 byte b;
b = (byte) i;
```

- A different type of conversion will occur when a floating-point value is assigned to an integer type: ***truncation.***
- Example, if the value 1.23 is assigned to an integer, the resulting value will simply be 1.



Type Conversion and Casting(cntd.,)

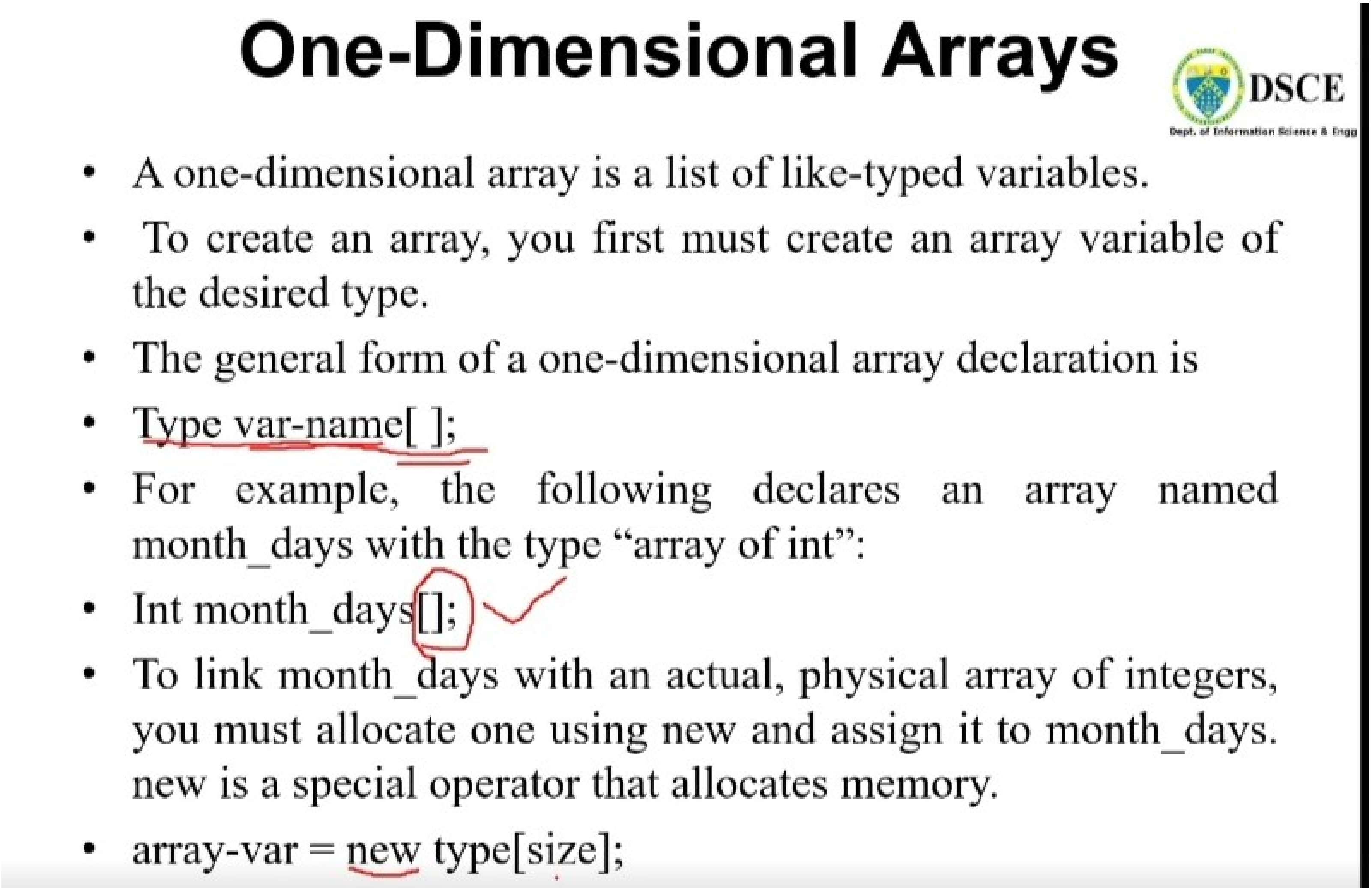
Example 1:

```
int a;
byte b;
b = (byte) a;
```

Example 2:

```
int i = 257 % 127= 3
 byte b;
b = (byte) i;    00000000 00000000 00000001 00000001
```

- A different type of conversion will occur when a floating-point value is assigned to an integer type: ***truncation.***
- Example, if the value 1.23 is assigned to an integer, the resulting value will simply be 1.

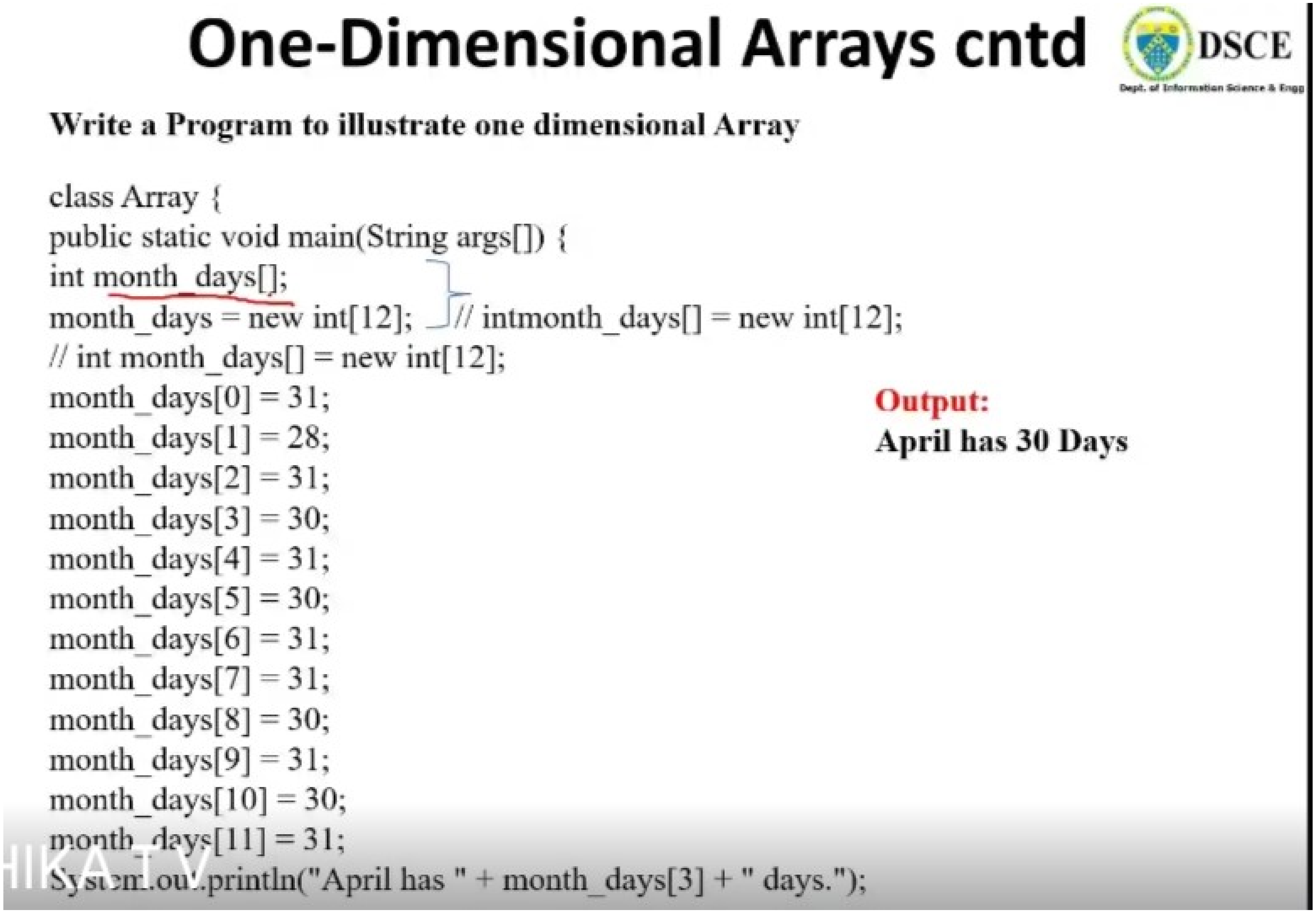


One-Dimensional Arrays

DSCE

Dept. of Information Science & Engg

- A one-dimensional array is a list of like-typed variables.
- To create an array, you first must create an array variable of the desired type.
- The general form of a one-dimensional array declaration is
- Type var-name[ ];
- For example, the following declares an array named month_days with the type "array of int":
- Int month_days[];
- To link month_days with an actual, physical array of integers, you must allocate one using new and assign it to month_days. new is a special operator that allocates memory.
- array-var = new type[size];

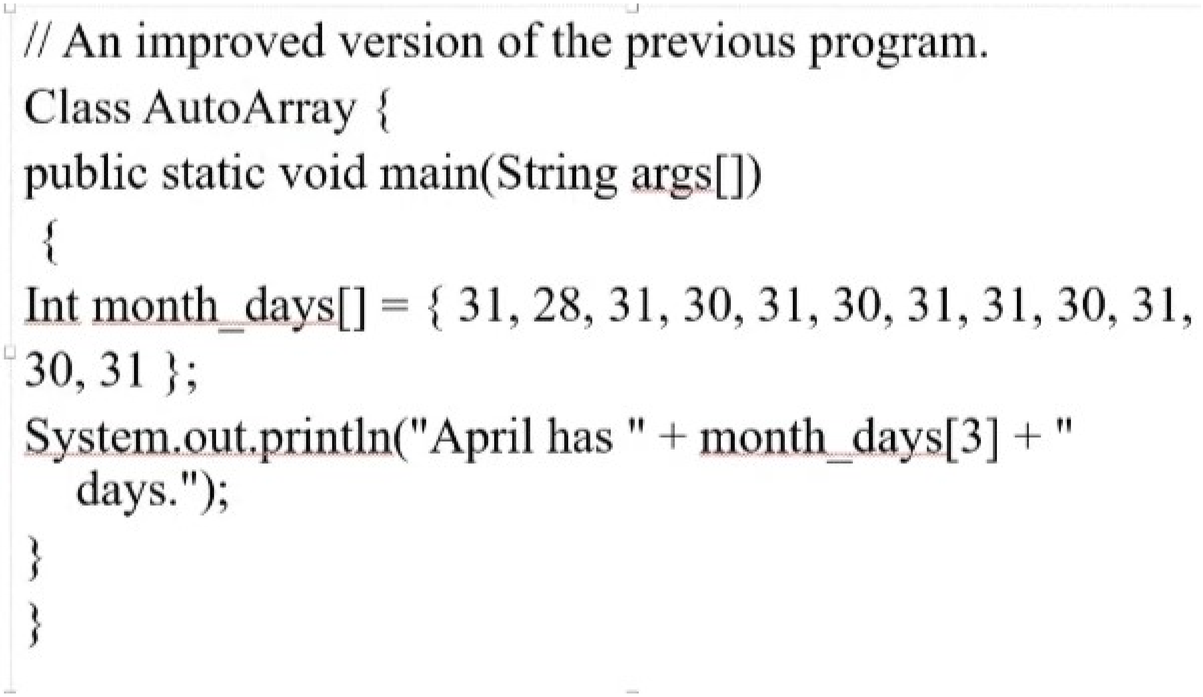


One-Dimensional Arrays cntd

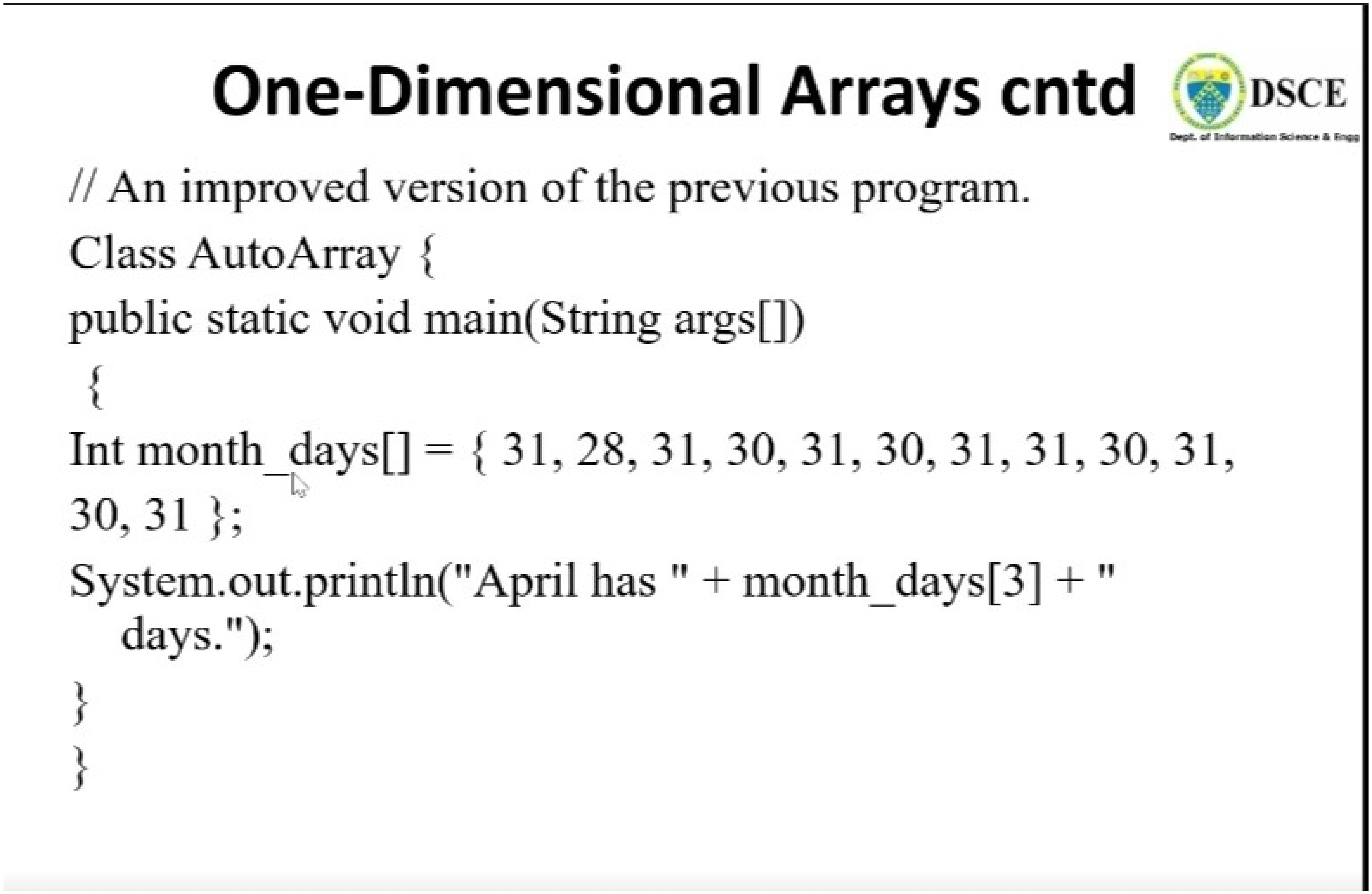
Write a Program to illustrate one dimensional Array

```
class Array {
public static void main(String args[]) {
int month_days[];
month_days = new int[12];  // intmonth_days[] = new int[12];
// int month_days[] = new int[12];
month_days[0] = 31;
month_days[1] = 28;
month_days[2] = 31;
month_days[3] = 30;
month_days[4] = 31;
month_days[5] = 30;
month_days[6] = 31;
month_days[7] = 31;
month_days[8] = 30;
month_days[9] = 31;
month_days[10] = 30;
month_days[11] = 31;
System.out.println("April has " + month_days[3] + " days.");
```

Output:
April has 30 Days



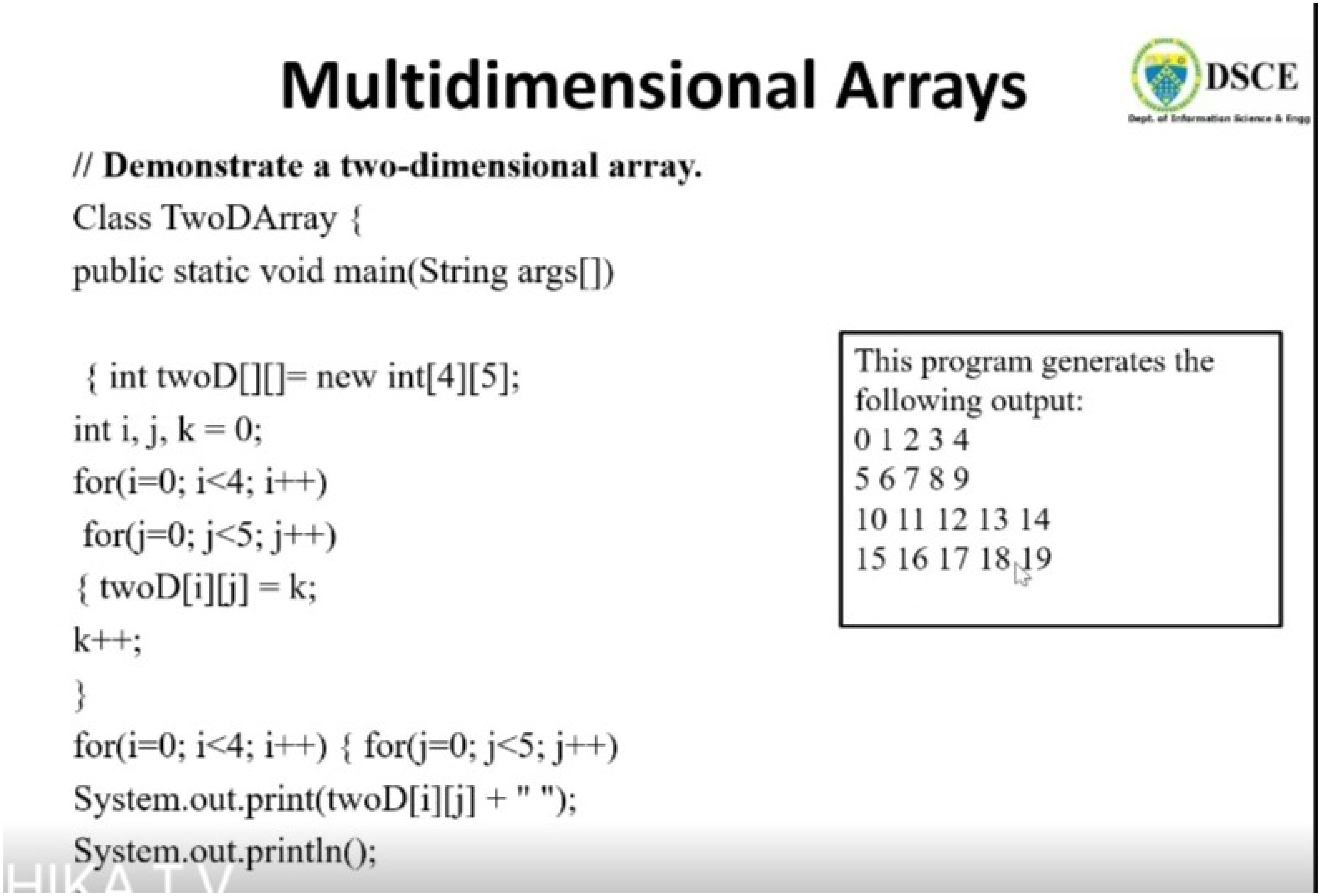
```
// An improved version of the previous program.
Class AutoArray {
public static void main(String args[])
 {
Int month_days[] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31,
30, 31 };
System.out.println("April has " + month_days[3] + "
   days.");
}
}
```



One-Dimensional Arrays cntd

DSCE
Dept. of Information Science & Engg

```
// An improved version of the previous program.
Class AutoArray {
public static void main(String args[])
 {
Int month_days[] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31,
30, 31 };
System.out.println("April has " + month_days[3] + "
   days.");
}
}
```



Multidimensional Arrays

```
// Demonstrate a two-dimensional array.
Class TwoDArray {
public static void main(String args[])

 { int twoD[][]= new int[4][5];
int i, j, k = 0;
for(i=0; i<4; i++)
 for(j=0; j<5; j++)
{ twoD[i][j] = k;
k++;
}
for(i=0; i<4; i++) { for(j=0; j<5; j++)
System.out.print(twoD[i][j] + " ");
System.out.println();
```

This program generates the following output:

```
0 1 2 3 4
5 6 7 8 9
10 11 12 13 14
15 16 17 18 19
```