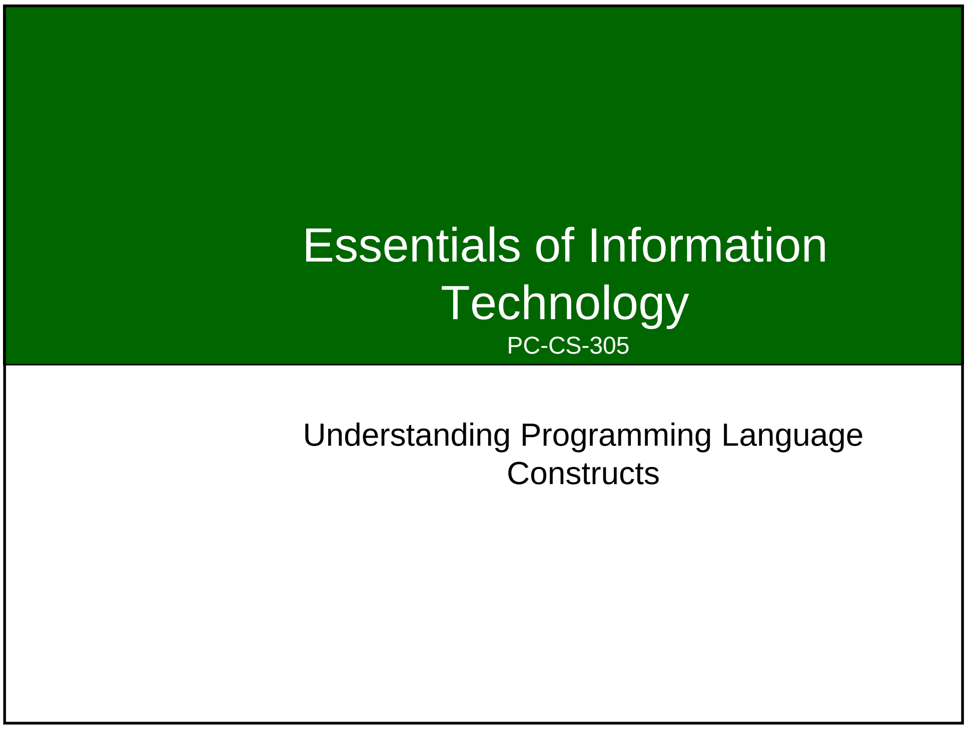
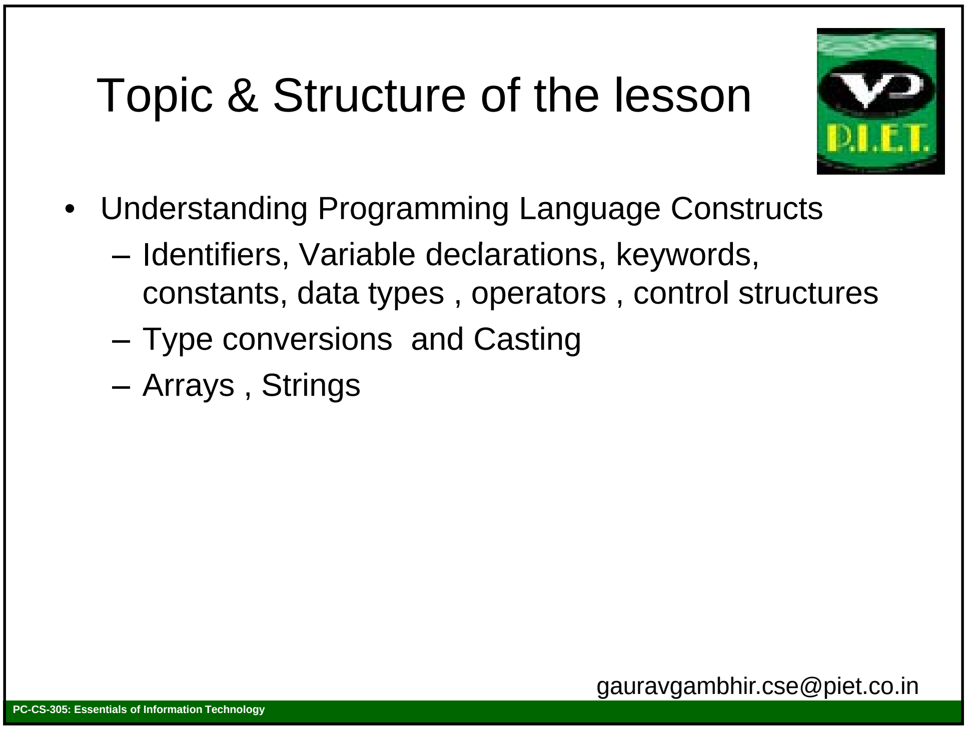


Essentials of Information Technology

PC-CS-305

Understanding Programming Language Constructs



Topic & Structure of the lesson

- Understanding Programming Language Constructs
  - Identifiers, Variable declarations, keywords, constants, data types , operators , control structures
  - Type conversions and Casting
  - Arrays , Strings

gauravgambhir.cse@piet.co.in

PC-CS-305: Essentials of Information Technology

Comments

Three permissible style of comment in a Java technology program are:

Single Line Comments

```
// comment on one line.
```

Multi line Comments

```
/* comment on one or
   more line */
```

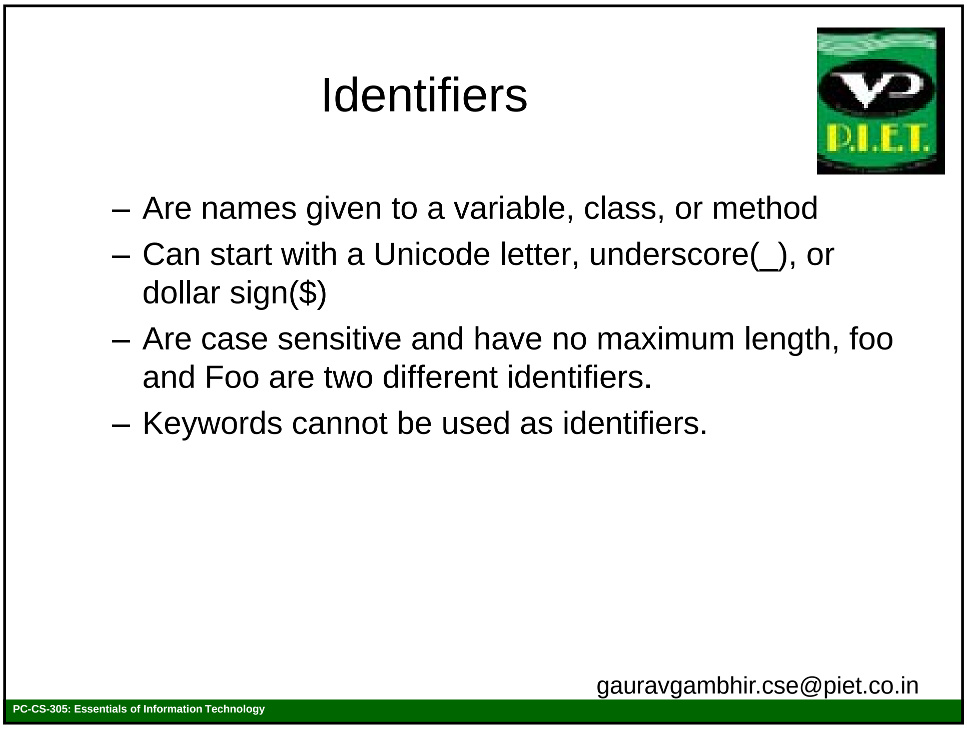
Comments

Documentation Comments

```
/** documentation comment */
```

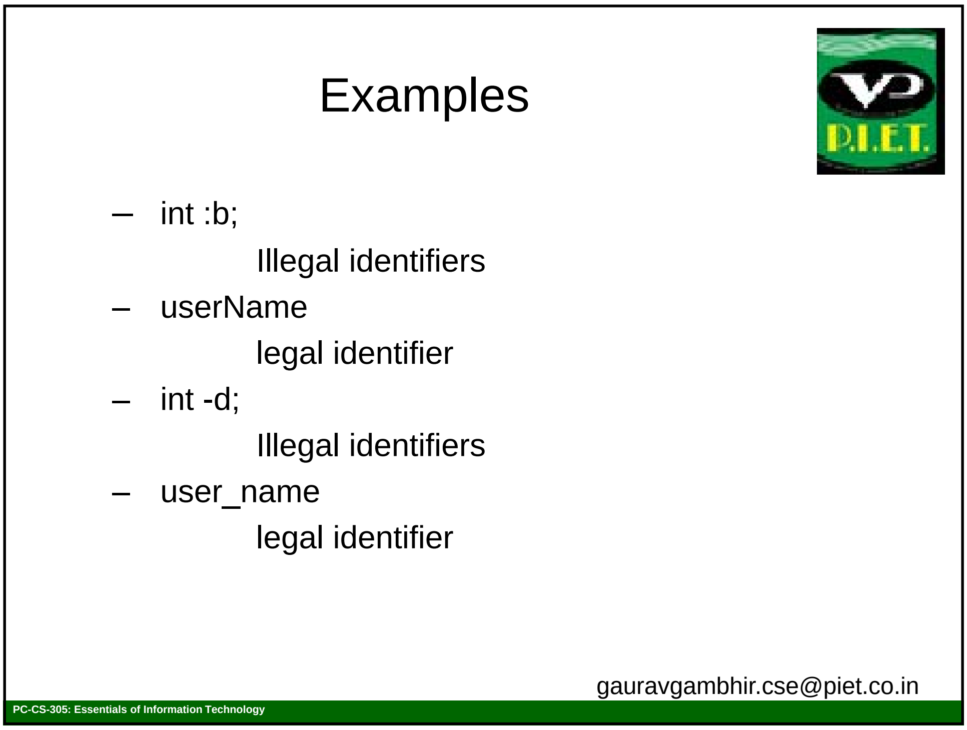
Example javadoc PowerPlant.java

(Documentation comment)



Identifiers

- Are names given to a variable, class, or method
- Can start with a Unicode letter, underscore(_), or dollar sign($)
- Are case sensitive and have no maximum length, foo and Foo are two different identifiers.
- Keywords cannot be used as identifiers.



Examples

- int :b;

  Illegal identifiers
- userName

  legal identifier
- int -d;

  Illegal identifiers
- user_name

  legal identifier

Example

- int e#;

  Illegal identifiers
- int .f;

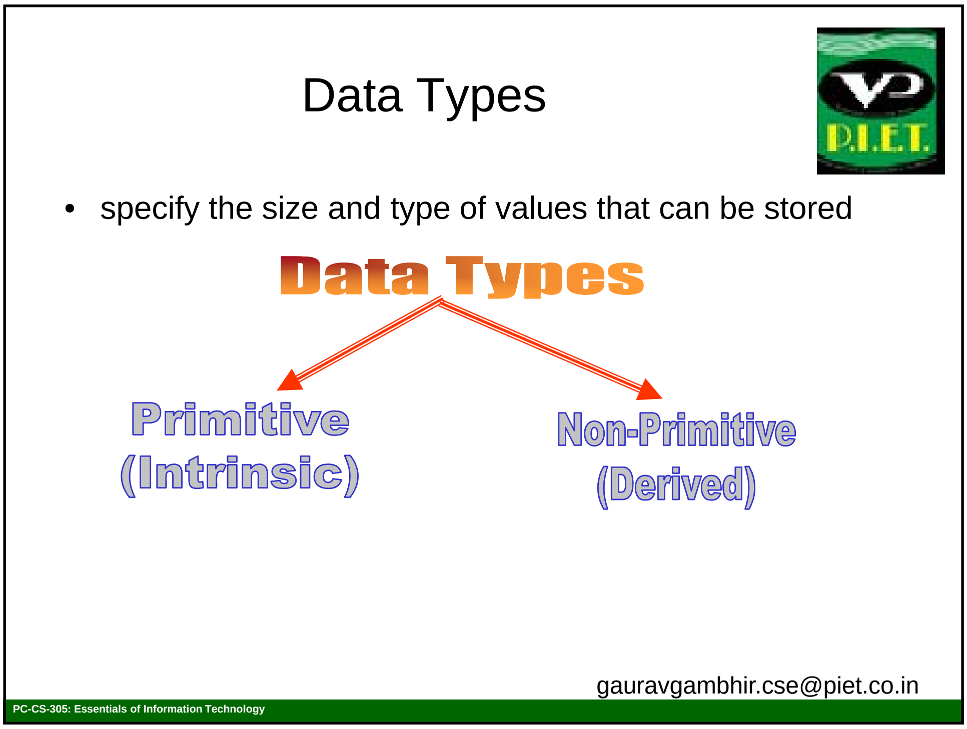
  Illegal identifiers
- int 7g;

  Illegal identifiers
- _sys_var1

  legal identifiers

Java Keywords

abstract	boolean	break	byte	case	catch
char	class	const	continue	default	do
double	else	extends	final	finally	float
for	goto	if	implements	import	instanceof
int	interface	long	native	new	package
private	protected	public	return	short	static
strictfp	super	switch	synchronized	this	throw
throws	transient	try	void	volatile	while
assert	enum				



Data Types

- specify the size and type of values that can be stored

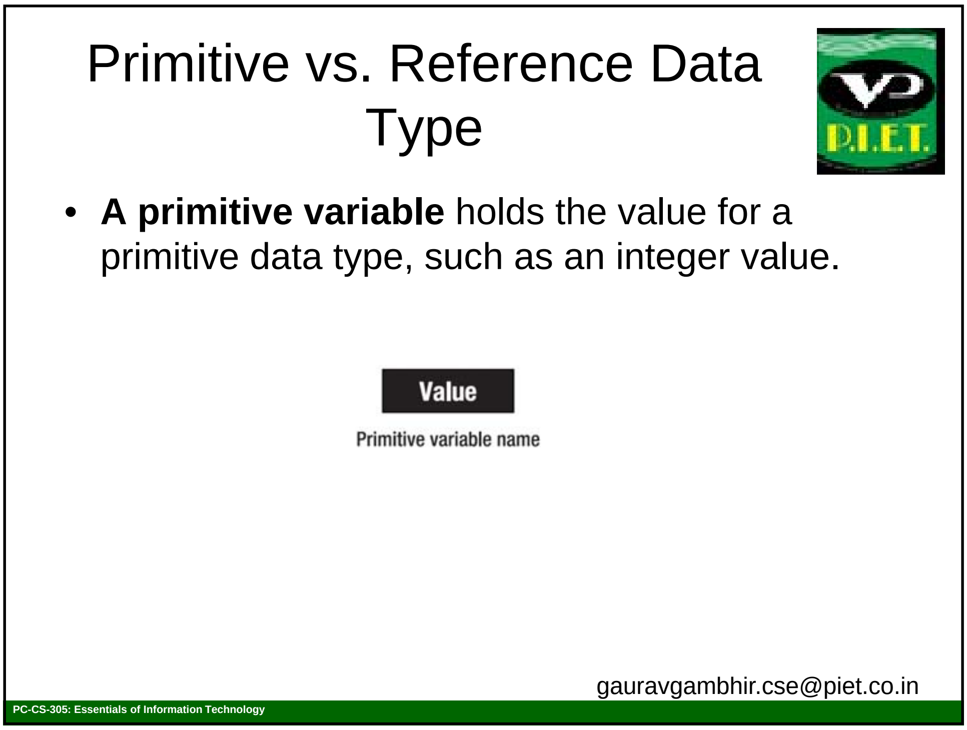
Data Types

Primitive (Intrinsic)

Non-Primitive (Derived)

gauravgambhir.cse@piet.co.in

PC-CS-305: Essentials of Information Technology



Primitive vs. Reference Data Type

- **A primitive variable** holds the value for a primitive data type, such as an integer value.

Value

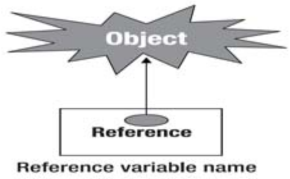
Primitive variable name

gauravgambhir.cse@piet.co.in

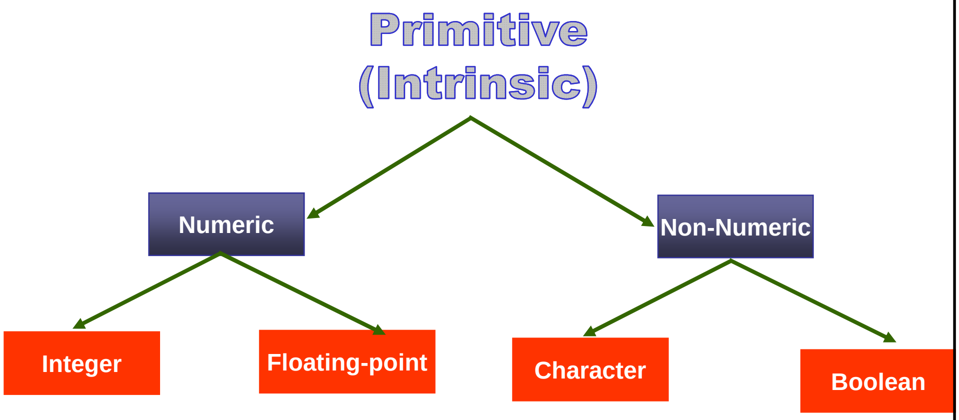
PC-CS-305: Essentials of Information Technology

Primitive vs. Reference Data Type

- **A reference variable** holds the reference to an object stored elsewhere in memory.



Data Types



Data Types   Integer

- can hold whole numbers
- 4 types of integer

Type	Size	Min	Max
byte	1 byte	-128	127
short	2 bytes	-32,768	32,767
int	4 bytes	-2,147,483,648	2,147,483,647
long	8 bytes	-9,223,372,036,854,775,808	9,223,372,036,854,775,807

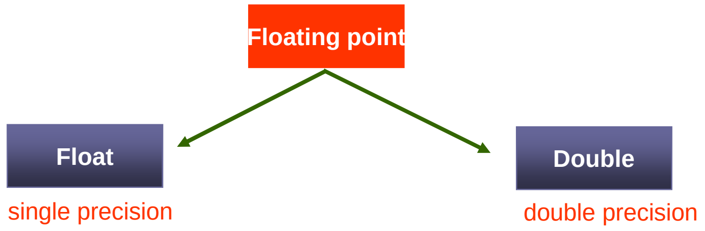
Data Types   Integer

- all Java values are signed – they can be positive or negative
- Use smaller data types wherever possible to improve speed of execution of the program
- can make integers long by appending letter L or l at the end of the number – e.g. 123L or 123l

gauravgambhir.cse@piet.co.in

Data Types   Floating point

- to hold numbers containing fractional parts



- floating point numbers are treated as double precision quantities
- to force them to be in single precision mode, we must append f or F to the numbers – e.g. 1.23f, 7.56923e5F

gauravjambhose@piet.co.in

Data Types Floating point

Type	Size	Min	Max
float	4 bytes	3.4e-038	3.4e+0.38
double	8 bytes	1.7e-308	1.7e+308

- float
  - single precision numbers
- double
  - double precision numbers
  - used for return values of mathematical functions such as sin, cos and sqrt

Data Types char

- char
  - size = 2 bytes (16-bits)
  - can hold only single character
  - Must have its literal enclosed in single quotes (‘ ’)
  - Uses the following notations:

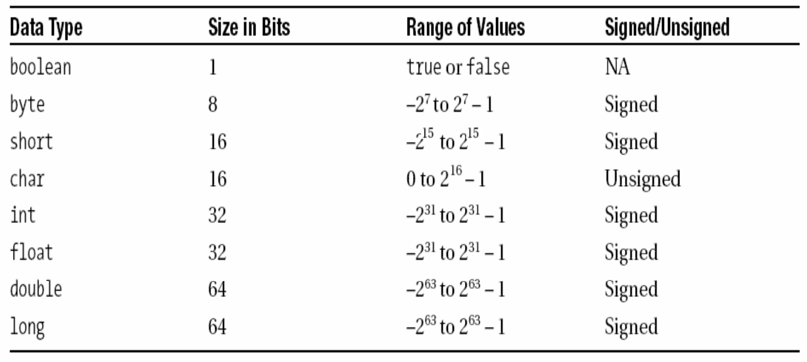
    'a' The letter a

Data Types boolean

- used when want to test a particular condition during execution of program
  - selection
  - iteration
- 2 values
  - True
  - False
- keyword   boolean
- uses only one bit of storage
- Example

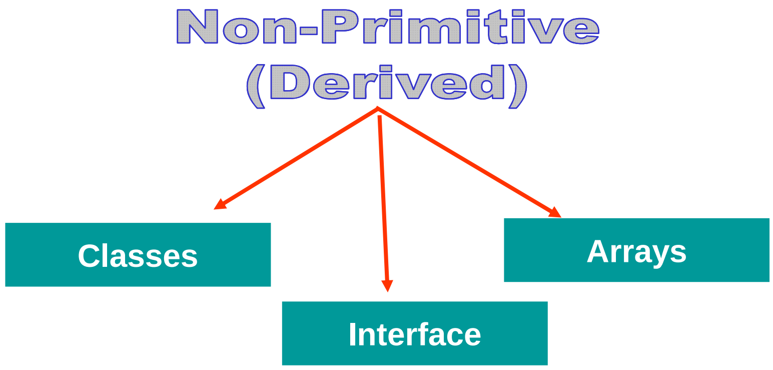
  boolean truth = true;

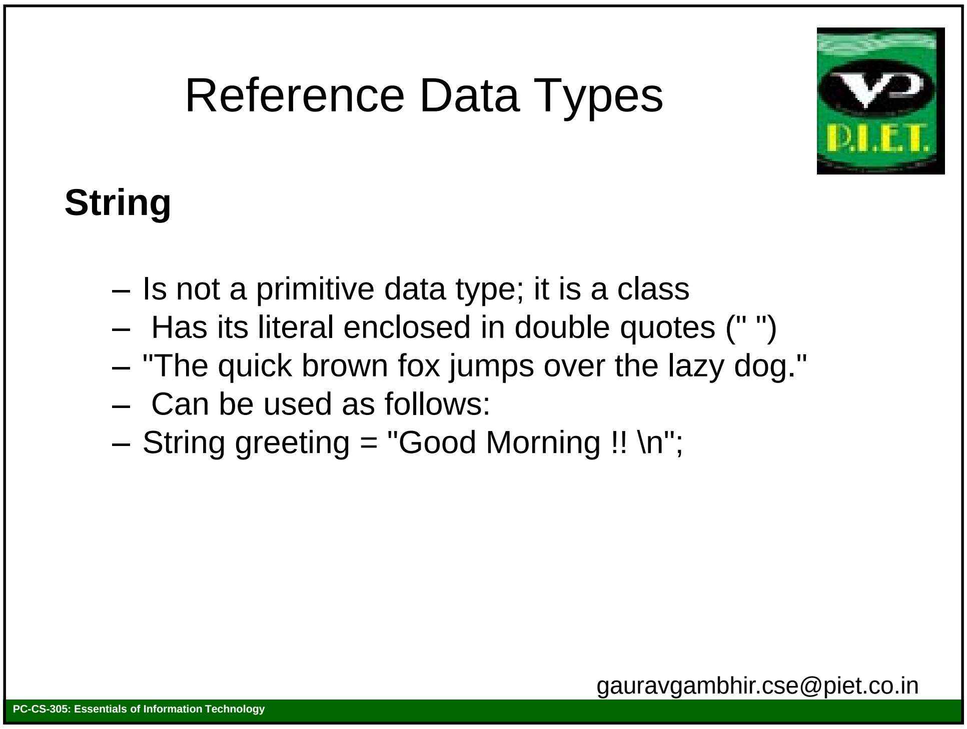
Summary of Primitive Datatypes



Data Type	Size in Bits	Range of Values	Signed/Unsigned
boolean	1	true or false	NA
byte	8	-2^7 to $2^7 - 1$	Signed
short	16	-2^{15} to $2^{15} - 1$	Signed
char	16	0 to $2^{16} - 1$	Unsigned
int	32	-2^{31} to $2^{31} - 1$	Signed
float	32	-2^{31} to $2^{31} - 1$	Signed
double	64	-2^{63} to $2^{63} - 1$	Signed
long	64	-2^{63} to $2^{63} - 1$	Signed

Data Type – discussed later chapters



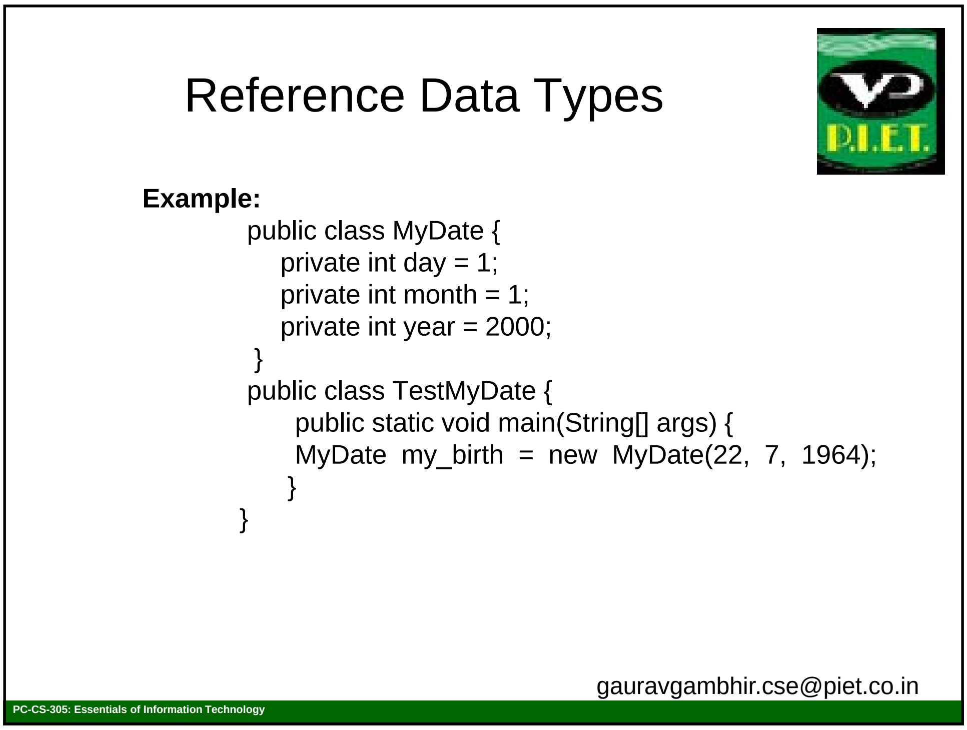


Reference Data Types

String

- Is not a primitive data type; it is a class
- Has its literal enclosed in double quotes (" ")
- "The quick brown fox jumps over the lazy dog."
- Can be used as follows:
- String greeting = "Good Morning !! \n";

gauravgambhir.cse@piet.co.in



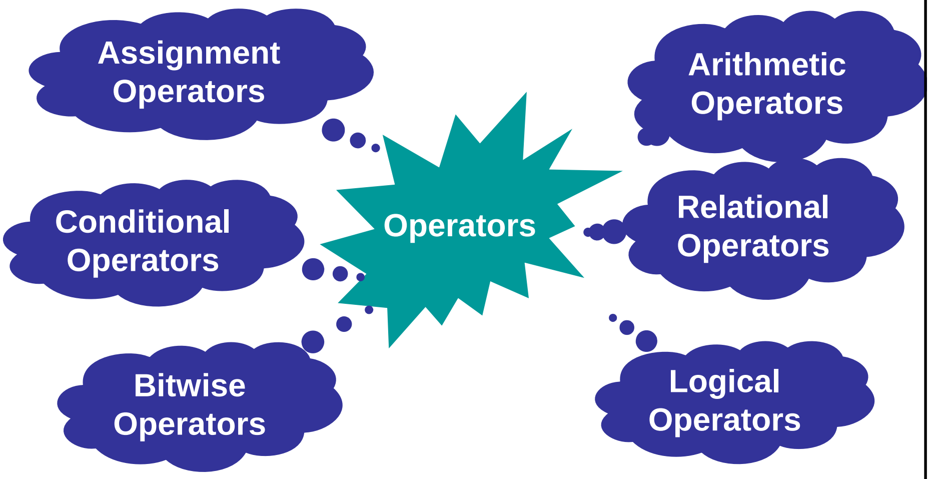
Reference Data Types

Example:

```
public class MyDate {
    private int day = 1;
    private int month = 1;
    private int year = 2000;
 }
public class TestMyDate {
      public static void main(String[] args) {
      MyDate  my_birth  =  new  MyDate(22,  7,  1964);
     }
}
```

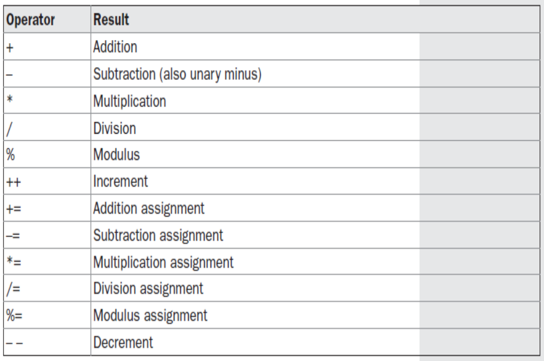
gauravgambhir.cse@piet.co.in

Operators



Arithmetic Operators

- Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra.



Operator	Result
+	Addition
–	Subtraction (also unary minus)
*	Multiplication
/	Division
%	Modulus
++	Increment
+=	Addition assignment
–=	Subtraction assignment
*=	Multiplication assignment
/=	Division assignment
%=	Modulus assignment
– –	Decrement

Arithmetic Operators

- The operands of the arithmetic operators must be of a numeric type. You cannot use them on **boolean types, but you can use them on char types, since the char type in Java is,** essentially, a subset of **int.**

- Example : Arithmeticopr.java

The Modulus Operator

- The modulus operator, **%, returns the remainder of a division operation. It can be applied to** floating-point types as well as integer types.

- The following example program demonstrates the **%**

```
// Demonstrate the % operator.
class Modulus {
    public static void main(String args[]) {
        int x = 42;
        double y = 42.25;
        System.out.println("x mod 10 = " + x % 10);
        System.out.println("y mod 10 = " + y % 10);
    }
}
```

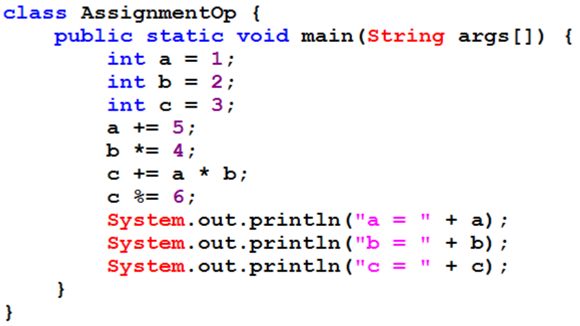
Assignment Operators

- Java provides special operators that can be used to combine an arithmetic operation with an assignment

  a = a + 4;

- This can be rewritten as

  a += 4;



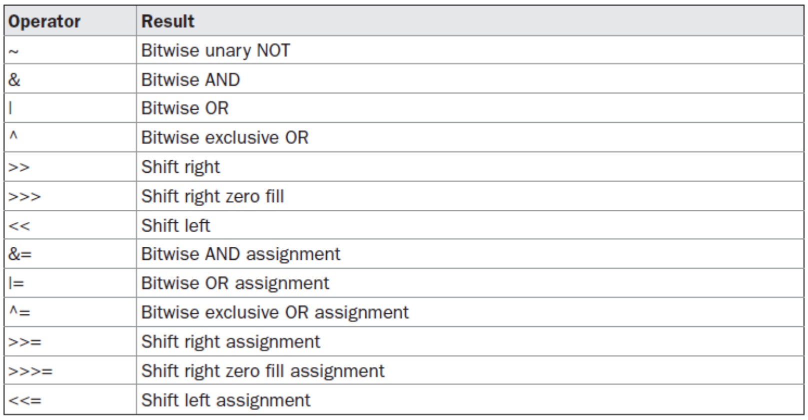
```
class AssignmentOp {
    public static void main(String args[]) {
        int a = 1;
        int b = 2;
        int c = 3;
        a += 5;
        b *= 4;
        c += a * b;
        c %= 6;
        System.out.println("a = " + a);
        System.out.println("b = " + b);
        System.out.println("c = " + c);
    }
}
```

Bitwise Operators

- Java defines several *bitwise operators that can be applied to the integer types,* ***long, int, short,*** **char, and byte.**

- **These operators**
  - act upon the individual bits of their operands.
  - are used for manipulation of data values of bit level
  - are used for testing the bits or shifting them to right or left
  - may not be applied to float or double

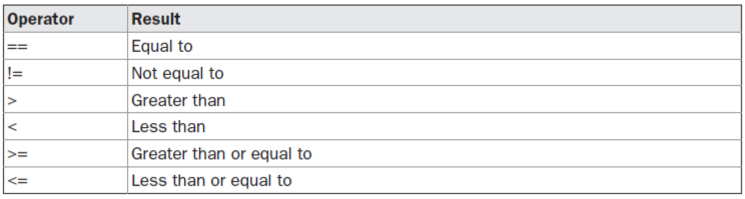
Bitwise Operators



Operator	Result
~	Bitwise unary NOT
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
>>	Shift right
>>>	Shift right zero fill
<<	Shift left
&=	Bitwise AND assignment
|=	Bitwise OR assignment
^=	Bitwise exclusive OR assignment
>>=	Shift right assignment
>>>=	Shift right zero fill assignment
<<=	Shift left assignment

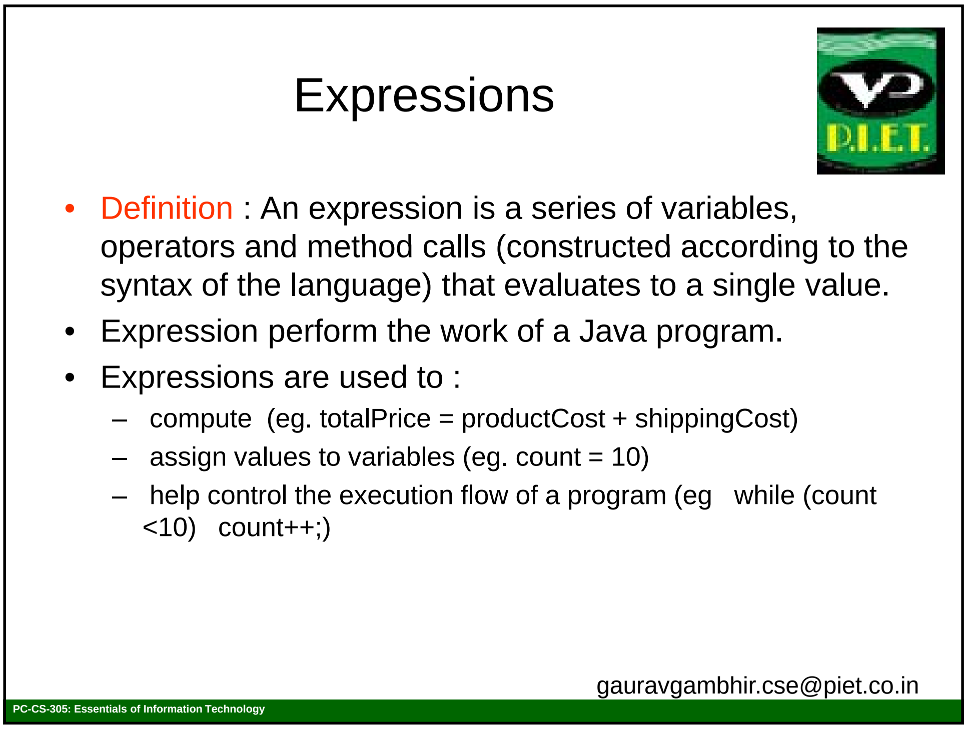
Relational Operators

- The *relational operators determine the relationship that one operand has to the other.* The relational operators are shown as:



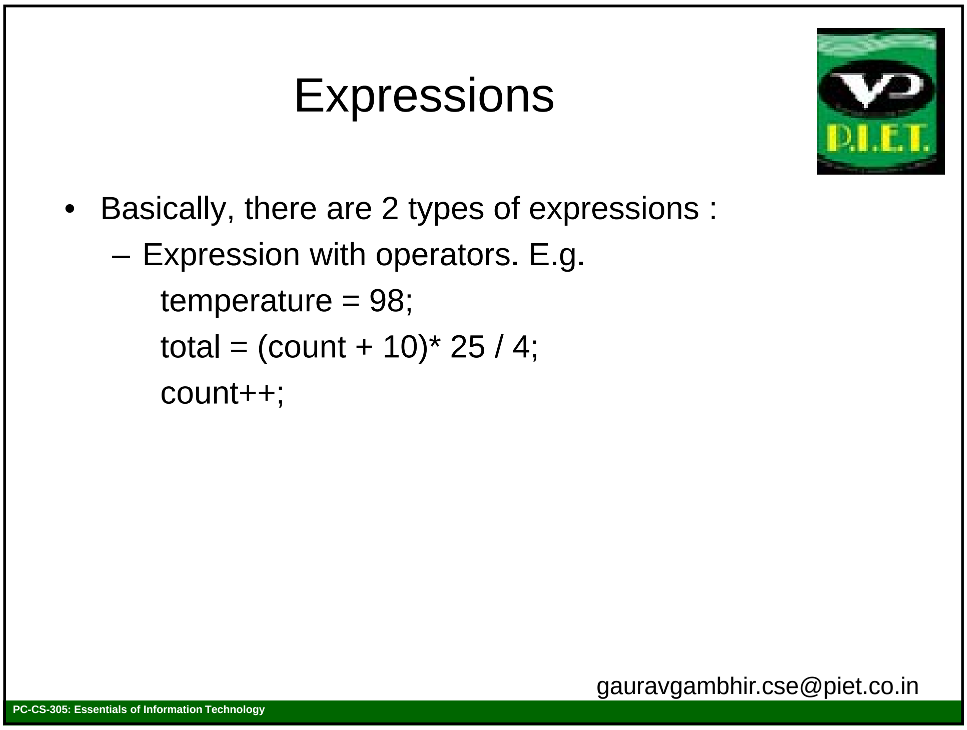
Operator	Result
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

- The outcome of these operations is a boolean value. The relational operators are most frequently used in the expressions that control the if statement and the various loop statements.



Expressions

- Definition : An expression is a series of variables, operators and method calls (constructed according to the syntax of the language) that evaluates to a single value.
- Expression perform the work of a Java program.
- Expressions are used to :
  - compute (eg. totalPrice = productCost + shippingCost)
  - assign values to variables (eg. count = 10)
  - help control the execution flow of a program (eg while (count <10) count++;)



Expressions

- Basically, there are 2 types of expressions :
  - Expression with operators. E.g.

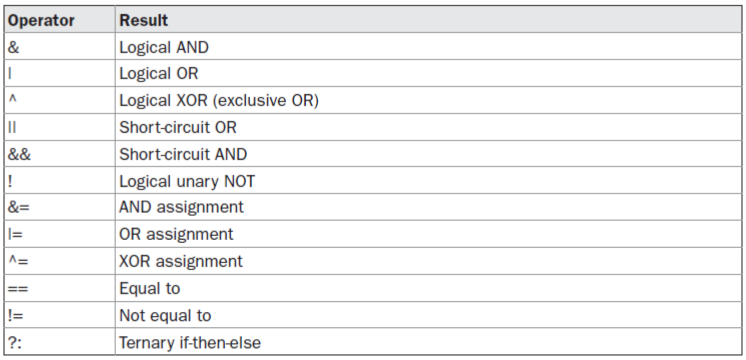
```
temperature = 98;
total = (count + 10)* 25 / 4;
count++;
```

Expressions

- Method call expression. E.g.
  - Keyboard.readInt(),
  - Integer.parseInt()
    - A method(function) call evaluates to the return value of the method.
    - The return data type of a method expression call is the same as the data type of the return value of that method.

Logical Operators

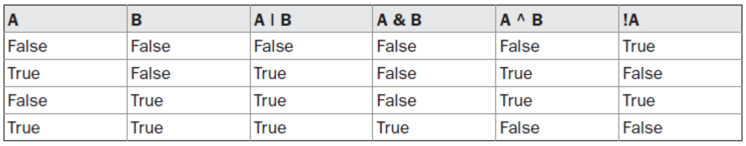
- The Boolean logical operators shown here operate only on **boolean operands.**



Operator	Result
&	Logical AND
|	Logical OR
^	Logical XOR (exclusive OR)
||	Short-circuit OR
&&	Short-circuit AND
!	Logical unary NOT
&=	AND assignment
|=	OR assignment
^=	XOR assignment
==	Equal to
!=	Not equal to
?:	Ternary if-then-else

Logical Operators

- The following table shows the effect of each logical operation

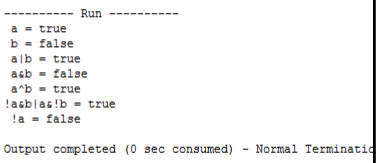


A	B	A | B	A & B	A ^ B	!A
False	False	False	False	False	True
True	False	True	False	True	False
False	True	True	False	True	True
True	True	True	True	False	False

Logical Operators

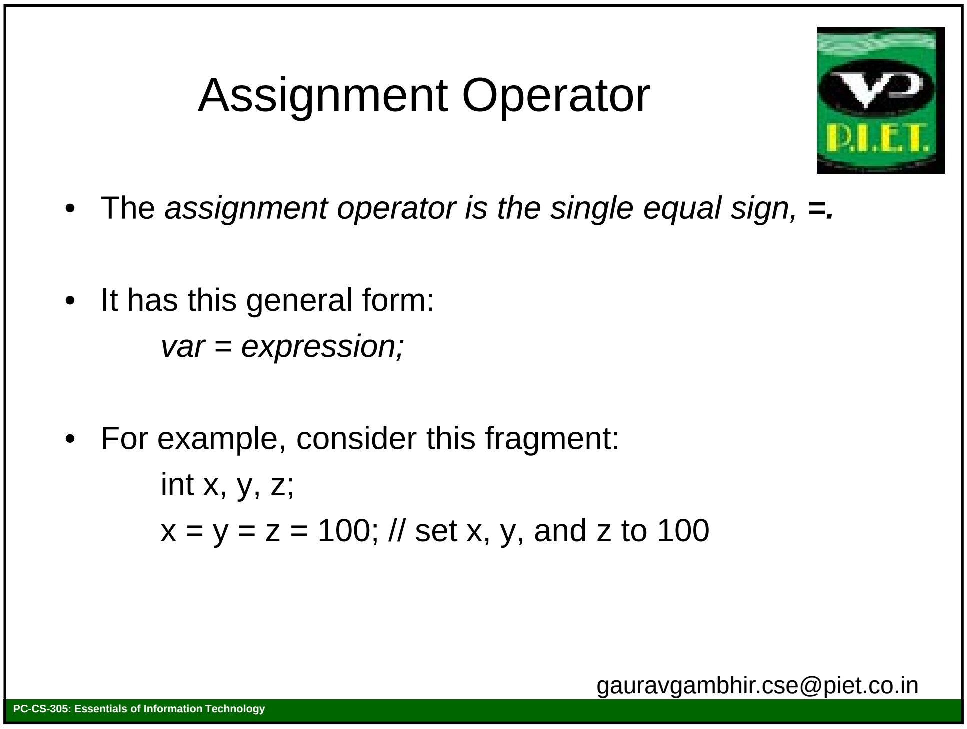
- Example :

```
class Logicalop {
    public static void main(String args[]) {
        boolean a = true;
        boolean b = false;
        boolean c = a | b;
        boolean d = a & b;
        boolean e = a ^ b;
        boolean f = (!a & b) | (a & !b);
        boolean g = !a;
        System.out.println(" a = " + a);
        System.out.println(" b = " + b);
        System.out.println(" a|b = " + c);
        System.out.println(" a&b = " + d);
        System.out.println(" a^b = " + e);
        System.out.println("!a&b|a&!b = " + f);
        System.out.println(" !a = " + g);
    }
}
```



```
---------- Run ----------
 a = true
 b = false
 a|b = true
 a&b = false
 a^b = true
!a&b|a&!b = true
 !a = false

Output completed (0 sec consumed) - Normal Terminatic
```

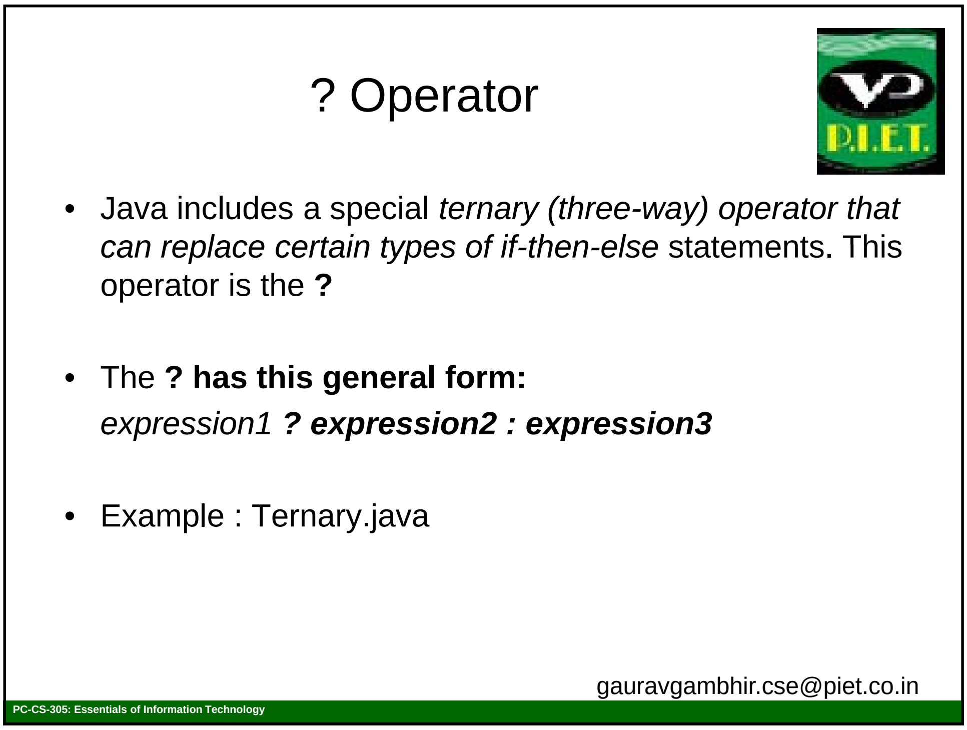


Assignment Operator

- The *assignment operator is the single equal sign,* ***=.***
- It has this general form:

  var = expression;
- For example, consider this fragment:

```
int x, y, z;
x = y = z = 100; // set x, y, and z to 100
```

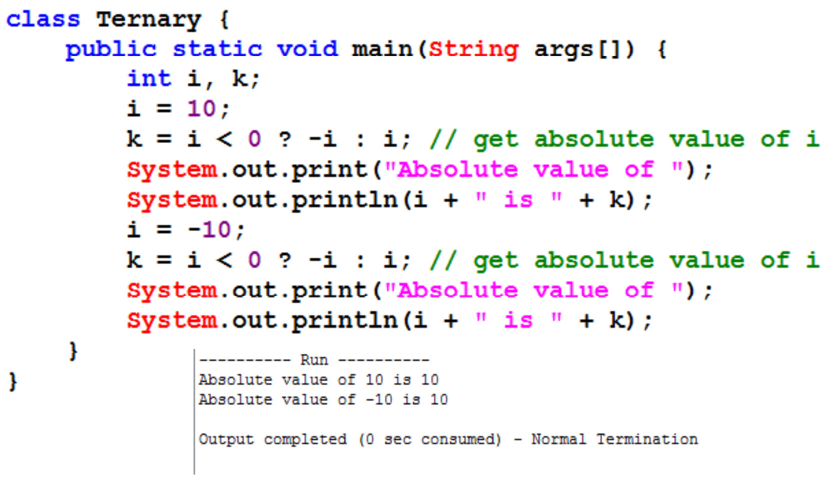


? Operator

- Java includes a special *ternary (three-way) operator that can replace certain types of if-then-else* statements. This operator is the **?**
- The **? has this general form:**

  expression1 ***? expression2 : expression3***
- Example : Ternary.java

? Operator Example



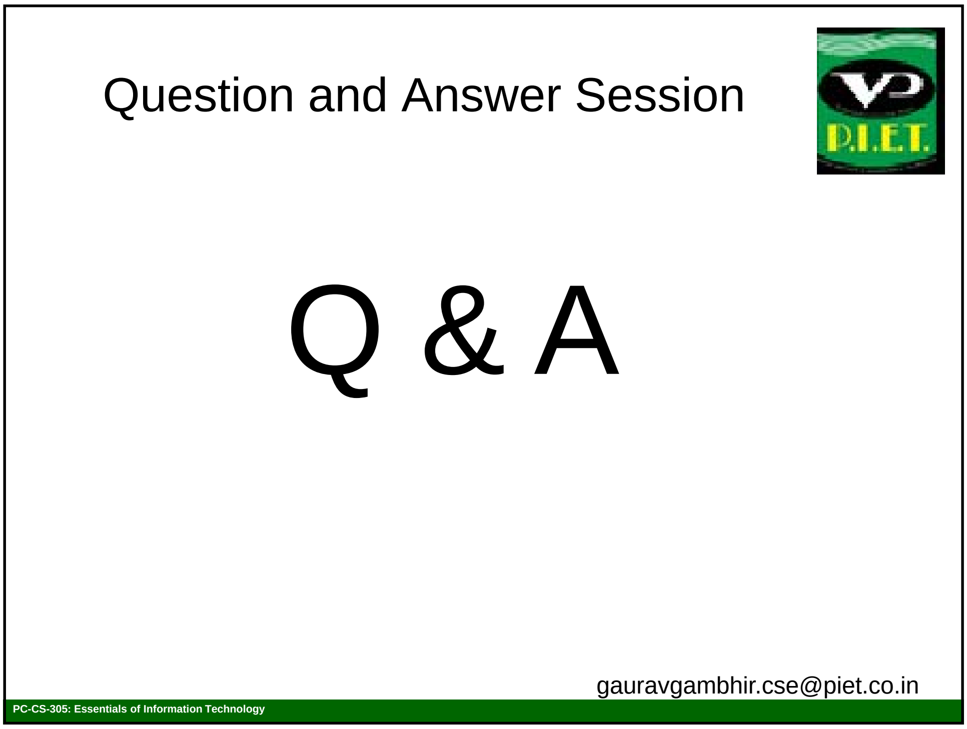
```
class Ternary {
    public static void main(String args[]) {
        int i, k;
        i = 10;
        k = i < 0 ? -i : i; // get absolute value of i
        System.out.print("Absolute value of ");
        System.out.println(i + " is " + k);
        i = -10;
        k = i < 0 ? -i : i; // get absolute value of i
        System.out.print("Absolute value of ");
        System.out.println(i + " is " + k);
    }
}
```

```
---------- Run ----------
Absolute value of 10 is 10
Absolute value of -10 is 10

Output completed (0 sec consumed) - Normal Termination
```

Summary of Main Teaching Points

- Understanding Programming Language Constructs
  - Variable declarations, definitions, keywords, constants, primitive data types
  - Syntax and professional programming practices



Question and Answer Session

Q & A