Java Operators

ICS 2 – Introduction to Computer Programming

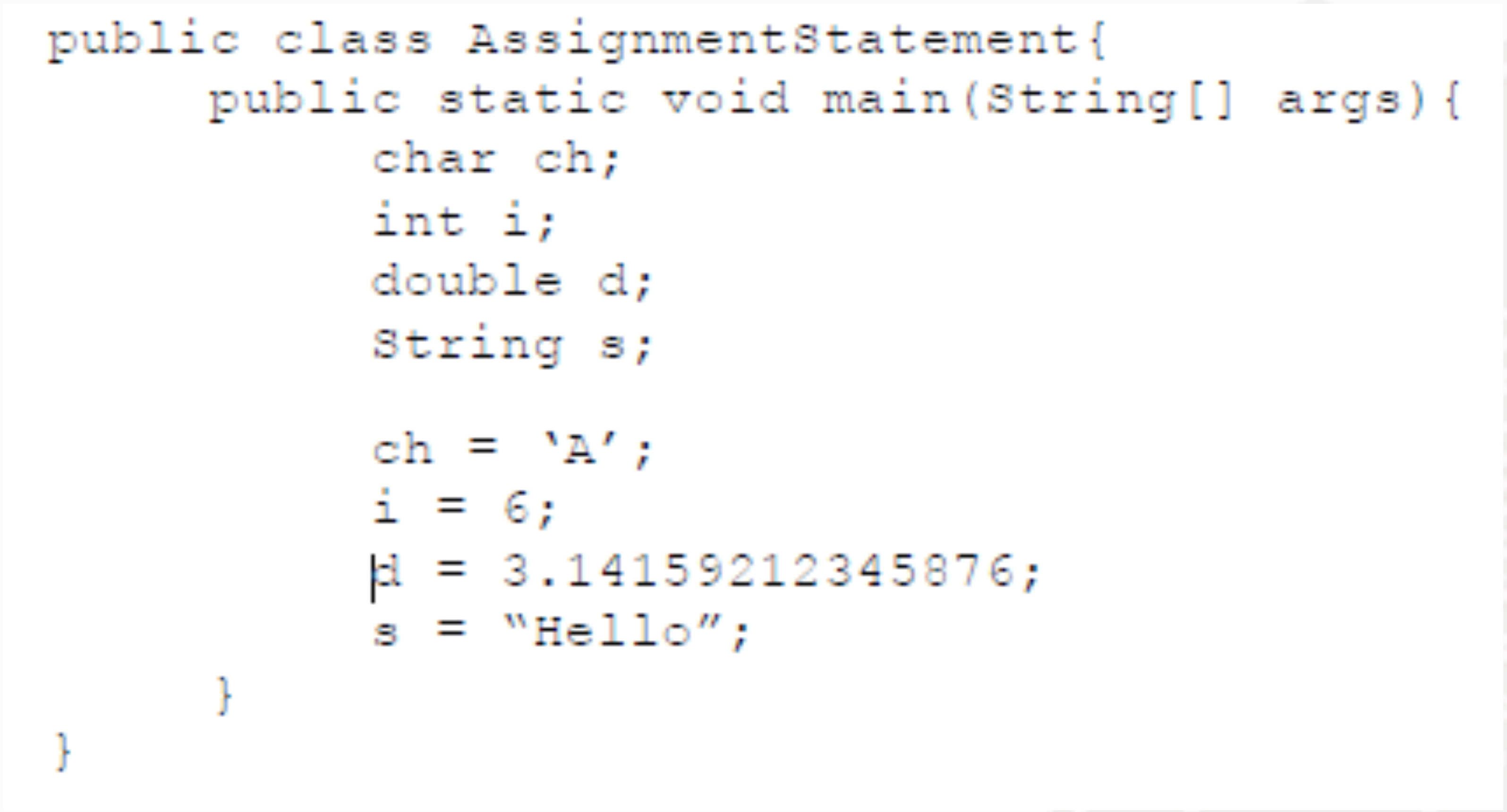
Operators

- Symbols representing operations that can be performed on constants and variables
- Types of Java Operators
  - Assignment Operator
  - Arithmetic Operators
  - Increment and Decrement Operators
  - Relational Operators
  - Logical Operators
  - Conditional Operator

Assignment Operator

- Used to store or assign a value from the right-hand side of an expression to the left-hand side.
- Symbol used is =
- Syntax:

  <variable_name>=<expression>;



```
public class AssignmentStatement{
    public static void main(String[] args){
        char ch;
        int i;
        double d;
        String s;

        ch = 'A';
        i = 6;
        d = 3.14159212345876;
        s = "Hello";
    }
}
```

- Note that it is also possible to assign a value of a variable to another variable
  - Example:

    X=5;
    Y=X;

- Storing Values with Different Type
  - There are automatic conversions that can take place among different data types.
  - Conversions from *int* to *double/float*, *char* to *int* and *char* to *double/float* are acceptable
  - Example:

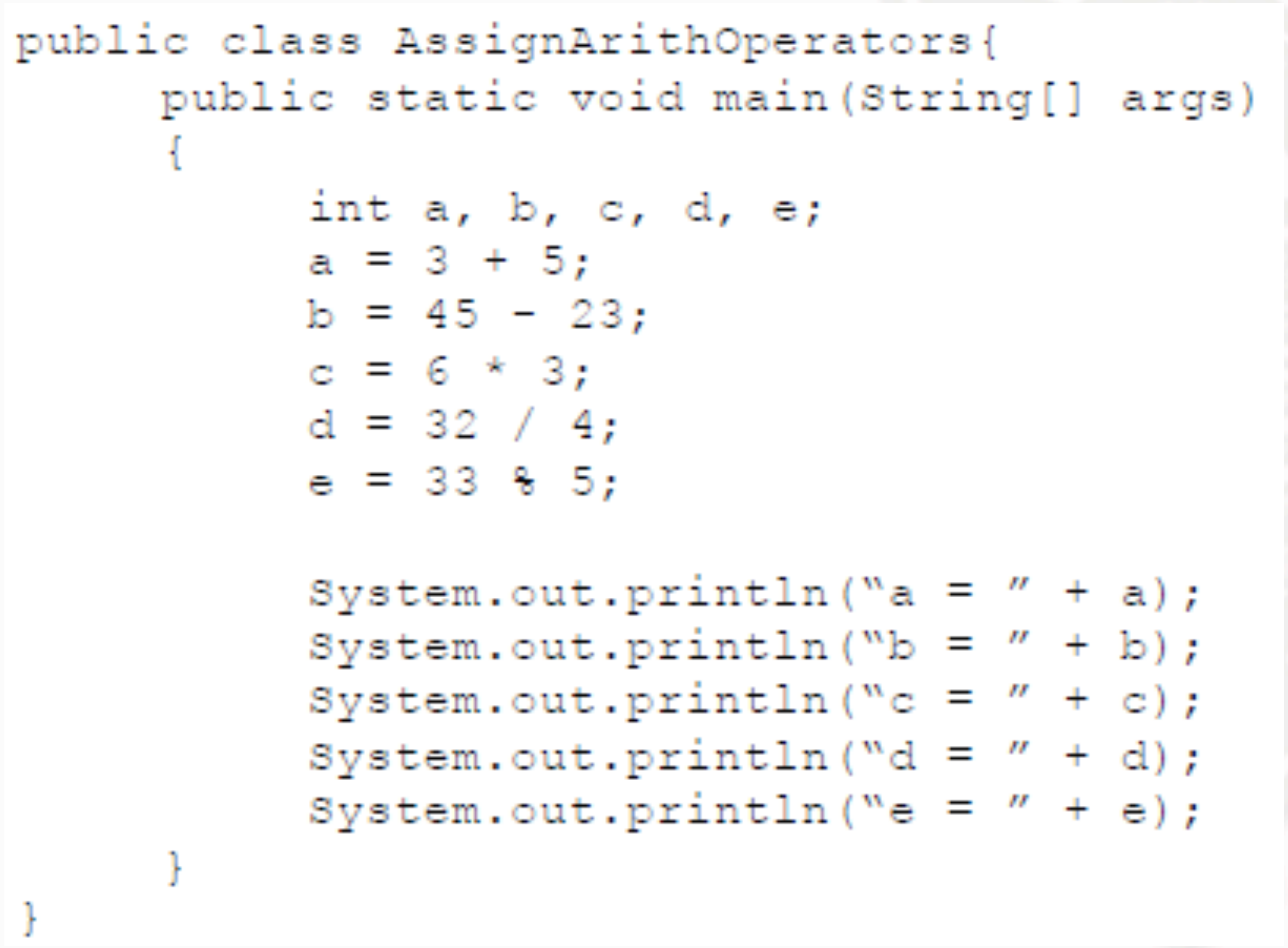
```
int i = 65;
char c = i;
```

Arithmetic Operators

- Also known as mathematical operators
- Basic operators are:

+	addition
-	subtraction
*	multiplication
/	division
%	modulo (yields the remainder)

- The *+, -, *, /* can be used for operands of type *int, float,* and *double*
- The *%* can only be used for *int* types
- The *+* can also be used as operator for *String* types



```java
public class AssignArithOperators{
     public static void main(String[] args)
     {
          int a, b, c, d, e;
          a = 3 + 5;
          b = 45 - 23;
          c = 6 * 3;
          d = 32 / 4;
          e = 33 % 5;

          System.out.println(“a = ” + a);
          System.out.println(“b = ” + b);
          System.out.println(“c = ” + c);
          System.out.println(“d = ” + d);
          System.out.println(“e = ” + e);
     }
}
```

- Precedence and Associativity Rule

Operator	Associativity
*,/, %	Left to Right
+, -	Left to Right

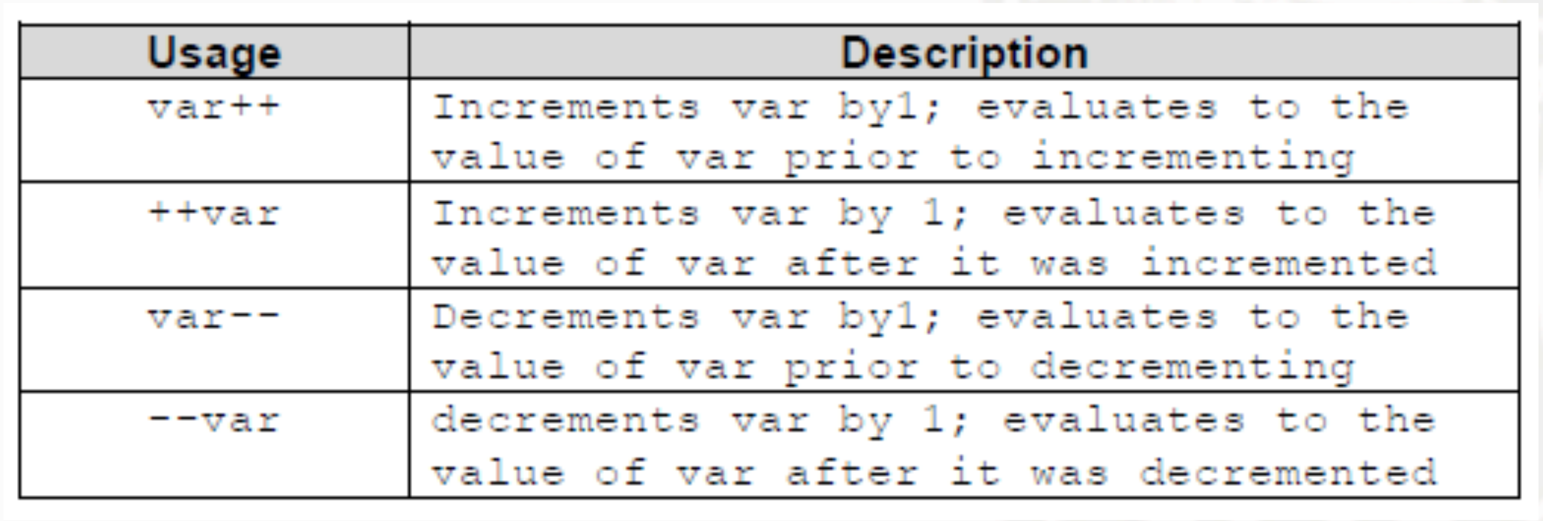
Note: to override precedence, use parentheses to group expression

```
X = 3 + 6 * 2 - 5 + 10 / 2 * 8 /2 - 3
X = 3 + 12 - 5 + 10 / 2 * 8 /2 - 3
X = 3 + 12 - 5 + 5 * 8 /2 - 3
X = 3 + 12 - 5 + 40/2 - 3
X = 3 + 12 - 5 +20 - 3
X = 15 - 5 +20 - 3
X = 10 + 20 - 3
X = 30 - 3
X = 27
```

```
X = (3 + 6) * (2 - (5 + (10 / 2)) * 8) /2-3
X = (3 + 6) * (2 - (5 + 5) * 8) /2 - 3
X = (3 + 6) * (2 - 10 * 8) /2 - 3
X = (3 + 6) * (2 - 80) /2 - 3
X = 9 * (2 - 80) /2 - 3
X = 9 * -78 /2 - 3
X = -702 / 2 - 3
X = -351 - 3
X = -354
```

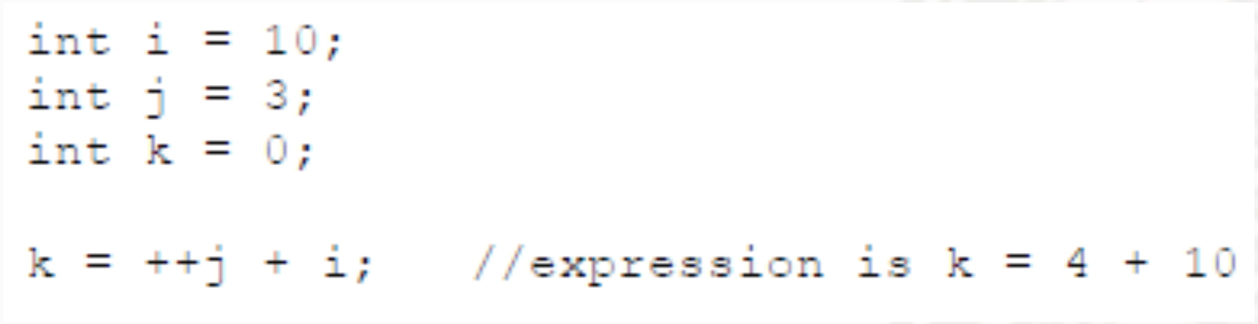
Increment and Decrement Operators

- Aside from the basic arithmetic operators, the *increment (++)* and *decerement (--)* operators can be written before or after a vaiable



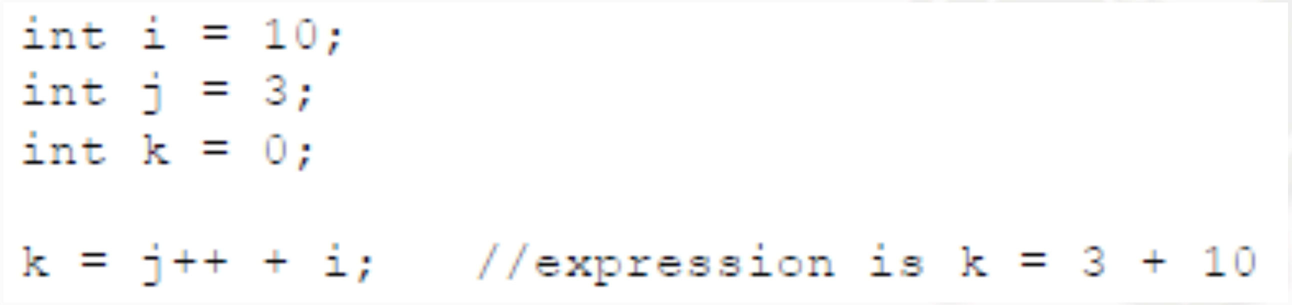
Usage	Description
`var++`	`Increments var by1; evaluates to the value of var prior to incrementing`
`++var`	`Increments var by 1; evaluates to the value of var after it was incremented`
`var--`	`Decrements var by1; evaluates to the value of var prior to decrementing`
`--var`	`decrements var by 1; evaluates to the value of var after it was decremented`

- Example 1:



```
int i = 10;
int j = 3;
int k = 0;

k = ++j + i;    //expression is k = 4 + 10
```

- Example 2:



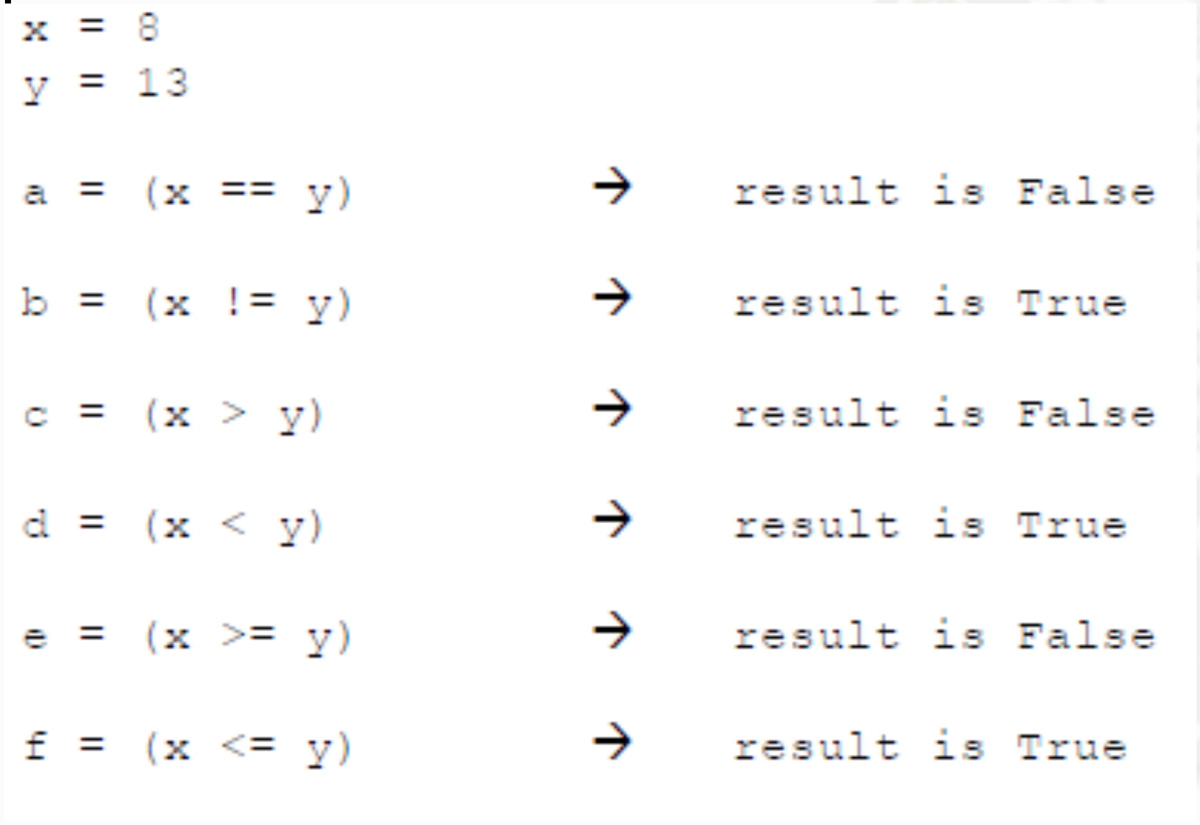
```
int i = 10;
int j = 3;
int k = 0;

k = j++ + i;    //expression is k = 3 + 10
```

Relational Operators

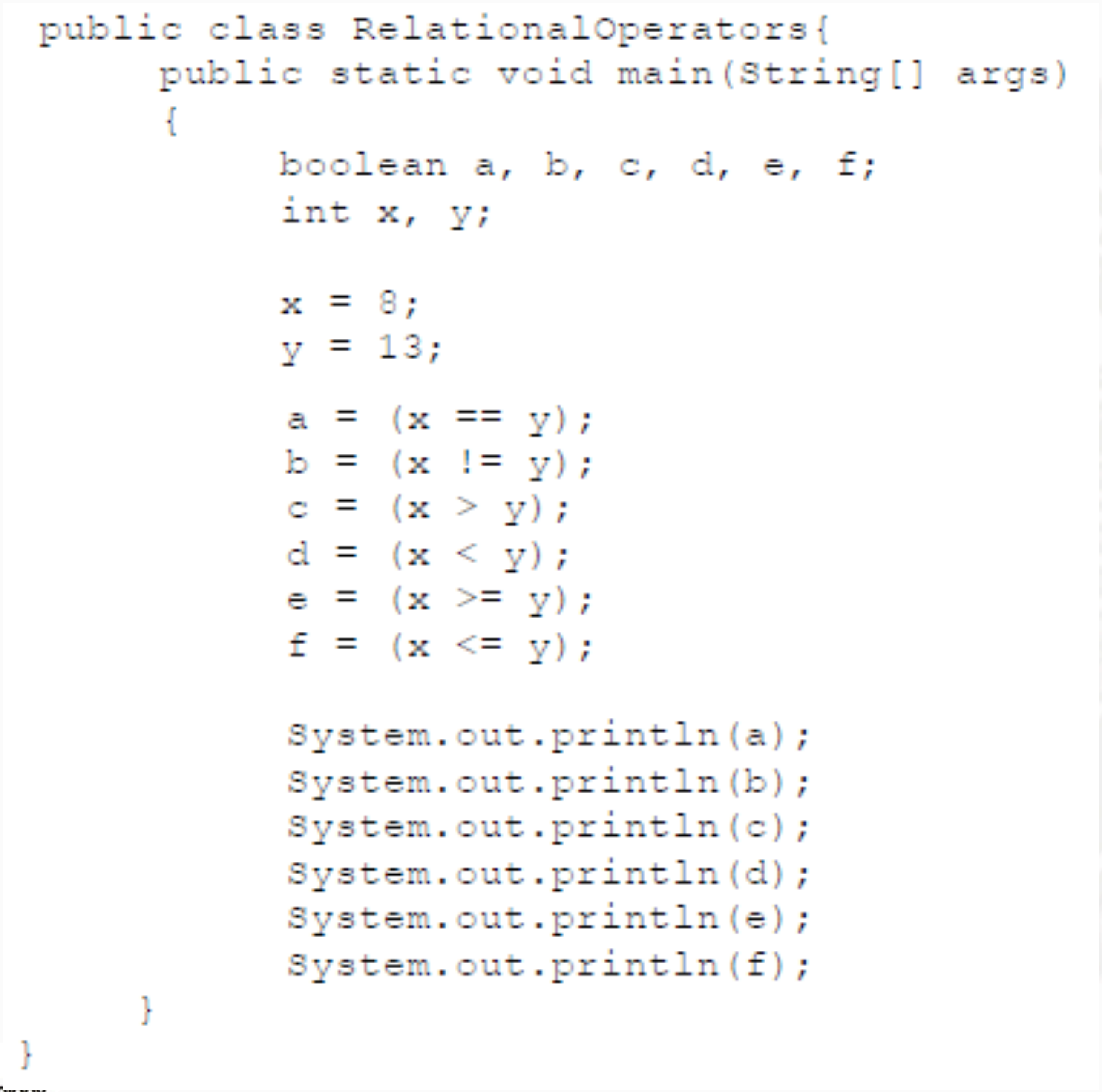
- Used to check association of the left-hand side expression to the right-hand side expression
- The result of the operation yields either a *true* or *false* value
- Operators:

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

- Example:



```
x = 8
y = 13

a = (x == y)          →    result is False

b = (x != y)          →    result is True

c = (x > y)           →    result is False

d = (x < y)           →    result is True

e = (x >= y)          →    result is False

f = (x <= y)          →    result is True
```



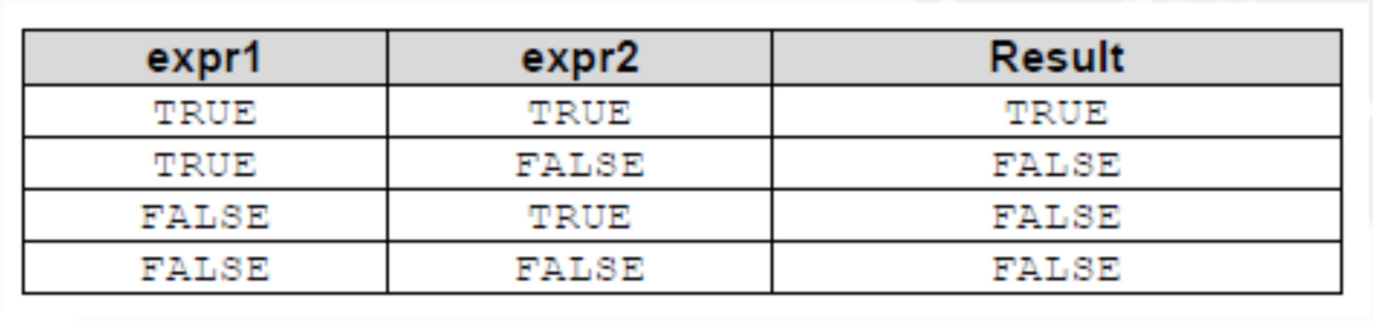
```
public class RelationalOperators{
    public static void main(String[] args)
    {
        boolean a, b, c, d, e, f;
        int x, y;

        x = 8;
        y = 13;

        a = (x == y);
        b = (x != y);
        c = (x > y);
        d = (x < y);
        e = (x >= y);
        f = (x <= y);

        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
        System.out.println(d);
        System.out.println(e);
        System.out.println(f);
    }
}
```

Logical Operators

- Used to test multiple conditions and are normally used in conjunction with relational operators
- Operators are:

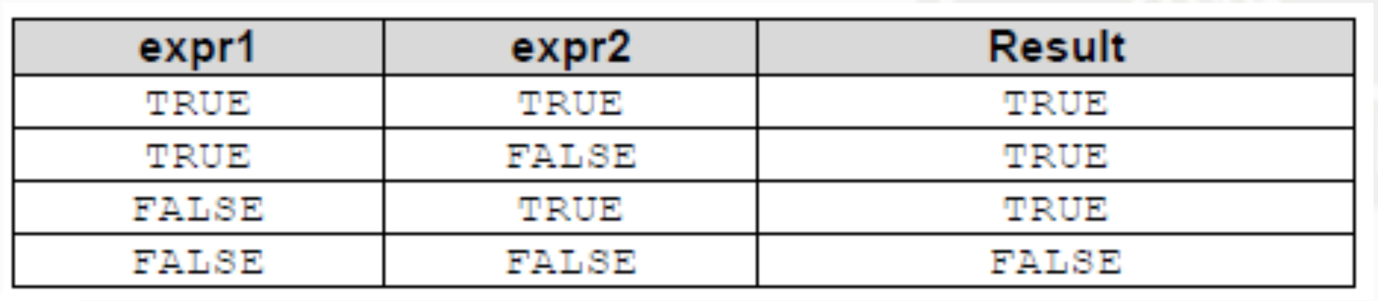
!	Logical NOT
&&	Logical AND
&	boolean Logical AND
||	Logical OR
|	boolean Logical OR
^	boolean Logical Exclusive OR

- Logical AND (&&) and boolean Logical AND(&)
  - General idea is that if all expressions evaluated are *true*, the result will be *true*
  - The difference between && and & is that once an expression is evaluated as *false* && will not anymore evaluate the other expression since it will already yield a *false* value; & evaluates both expressions regardless of the value



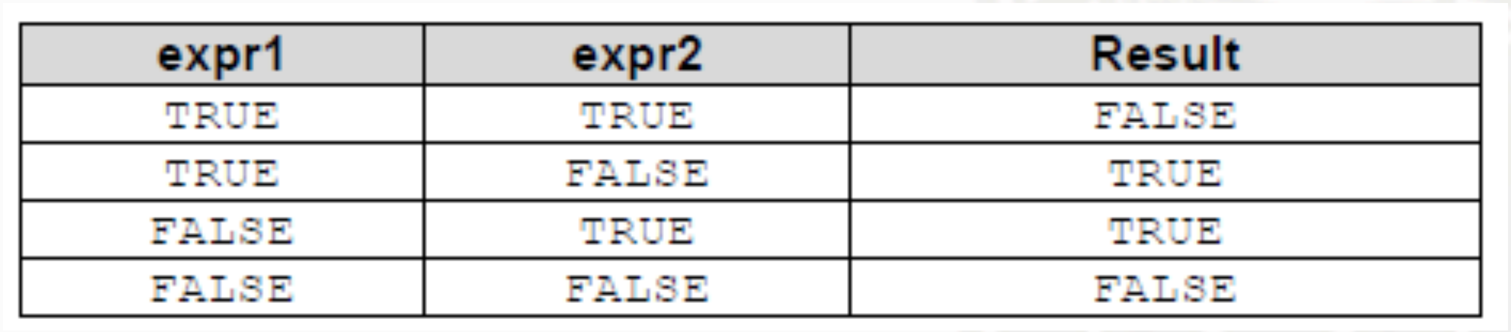
expr1	expr2	Result
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

- Logical OR (||) and boolean Logical OR (|)
  - General idea is that if one expression is evaluated is *true*, the statement will yield to a *true* value
  - The difference between || and | is that once an expression is evaluated as *true* && will not anymore evaluate the other expression since it will already yield a *true* value; | evaluates both expressions regardless of the value



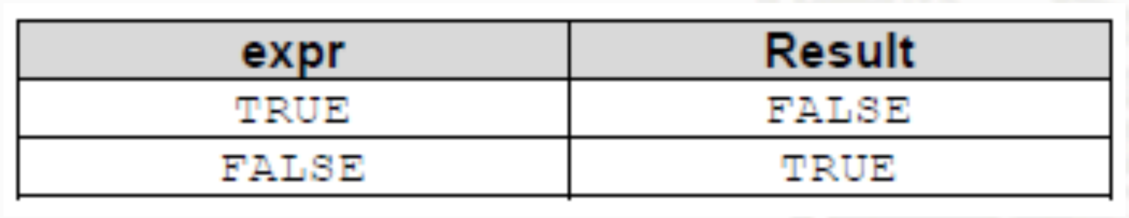
expr1	expr2	Result
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

- boolean Logical Exclusive OR (^)
    - The idea behind ^ is that the statement is true if and only if one operand is *true* and the other is *false*



expr1	expr2	Result
TRUE	TRUE	FALSE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

- Logical NOT (!)
  - A unary operator used to negate or get the opposite of a certain result in a relational operation



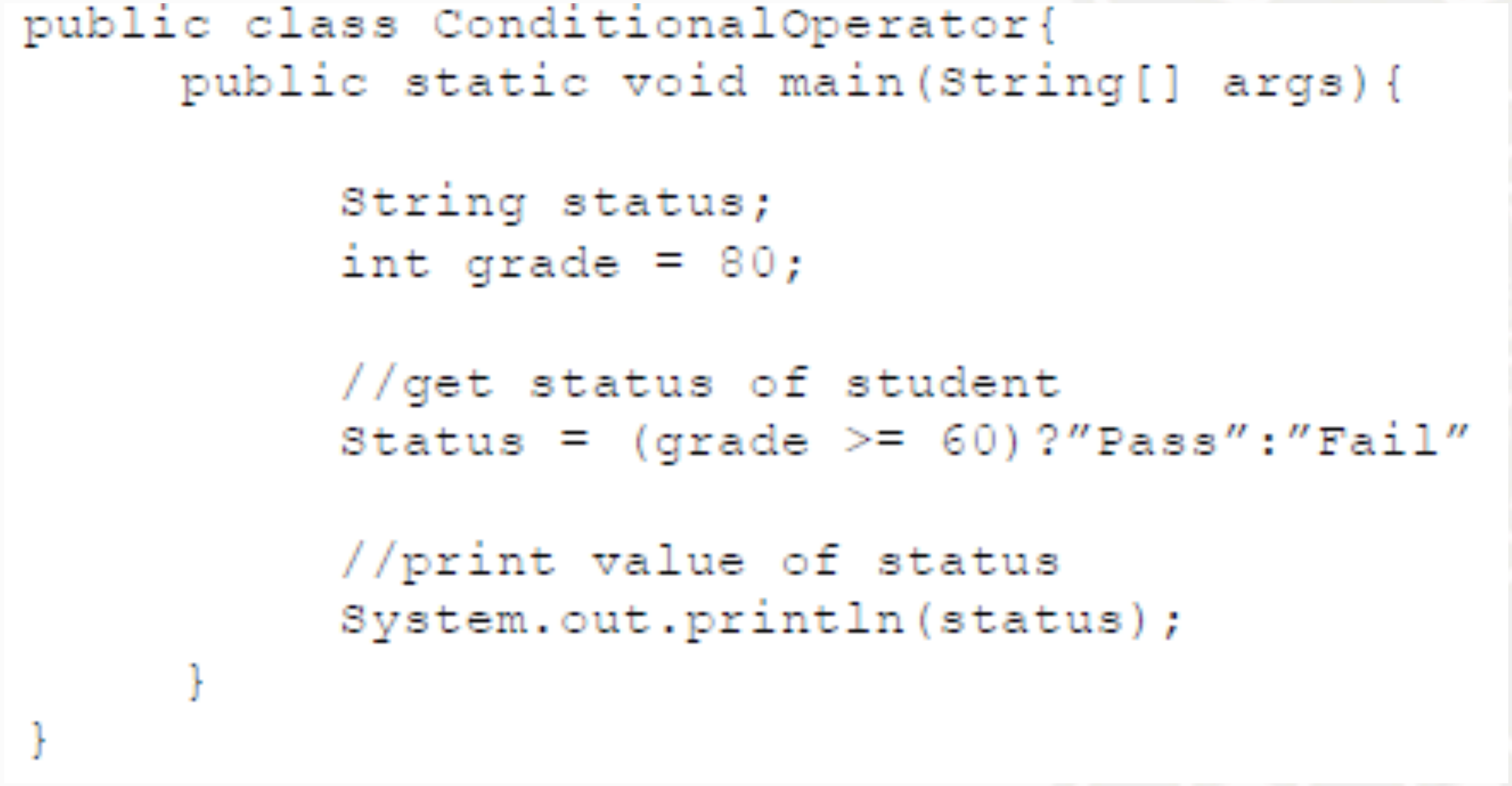
expr	Result
TRUE	FALSE
FALSE	TRUE

Conditional Operator (?:)

- The ?: is a ternary operator since it takes three arguments that together form a conditional expression
- Syntax:

  expr1 ? expr2 : expr3

- Idea behind ?: is that if *expr1* is *true*, value of *expr2* is returned, otherwise, *expr3* is returned



```
public class ConditionalOperator{
     public static void main(String[] args){

          String status;
          int grade = 80;

          //get status of student
          Status = (grade >= 60)?”Pass”:”Fail”

          //print value of status
          System.out.println(status);
     }
}
```