CSC 125
Object Oriented Programming

Ch02_2_Elementary programming

Dr. Fadi Alzhouri

جامعة الخليج
للعلوم والتكنولوجيا

Gulf University
for Science & Technology

SINCE 2002

Formatting output: Escape sequence

- An escape sequence in programming is a series of characters that represents a special character or format.
- Escape sequences typically start with a backslash (\) followed by one or more characters.

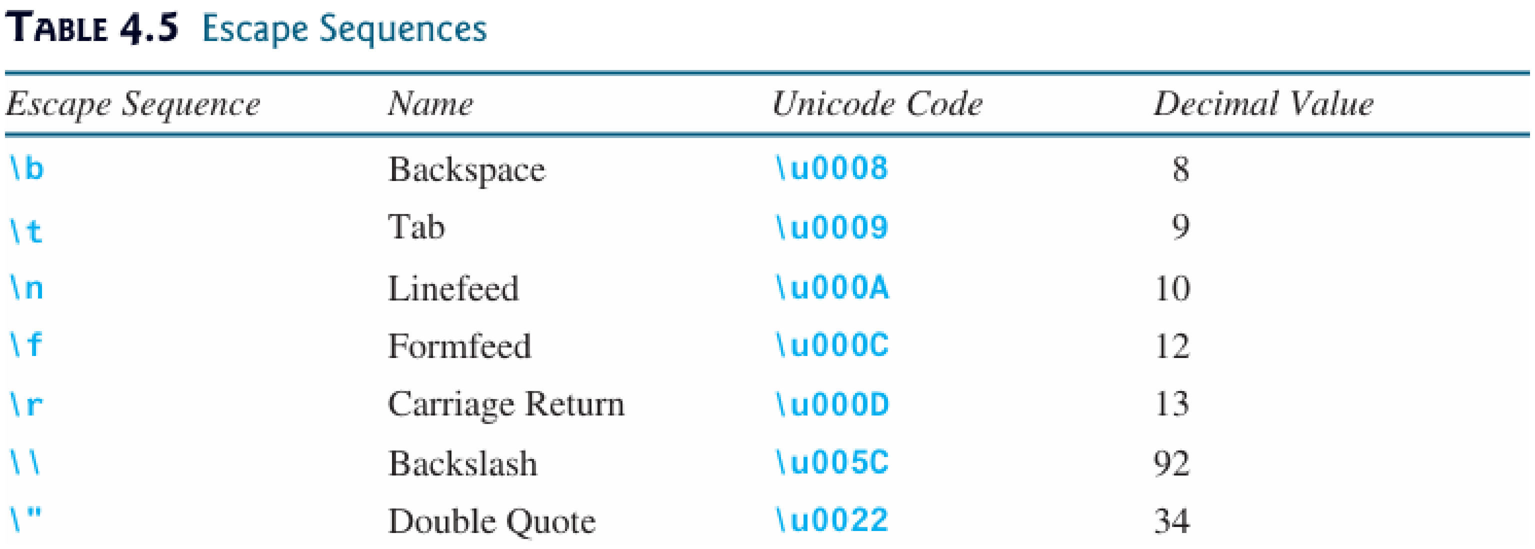


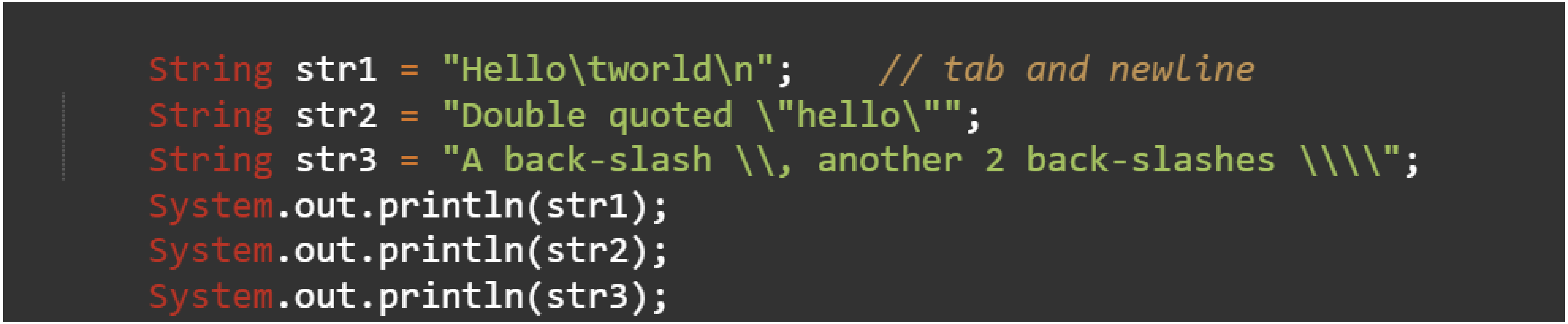
TABLE 4.5 Escape Sequences

Escape Sequence	*Name*	*Unicode Code*	*Decimal Value*
`\b`	Backspace	`\u0008`	8
`\t`	Tab	`\u0009`	9
`\n`	Linefeed	`\u000A`	10
`\f`	Formfeed	`\u000C`	12
`\r`	Carriage Return	`\u000D`	13
`\\`	Backslash	`\u005C`	92
`\"`	Double Quote	`\u0022`	34

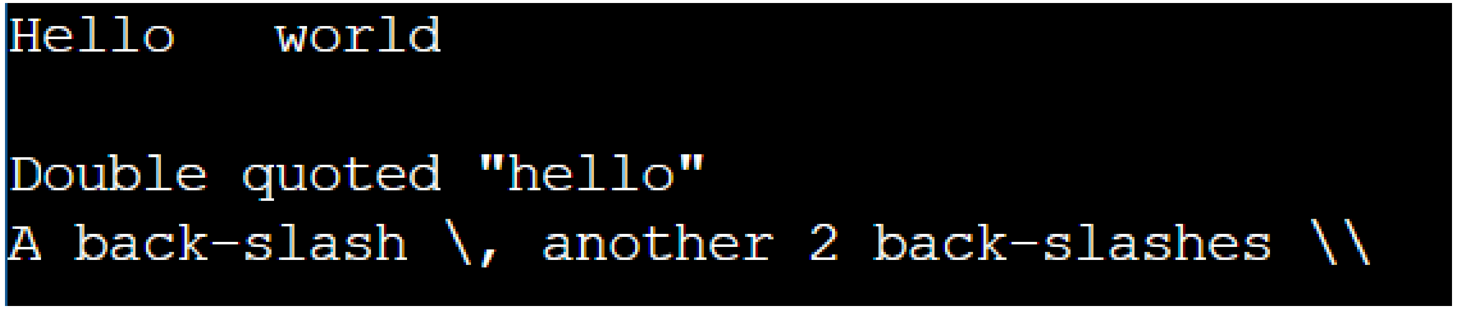
commonly-used escape sequences

Formatting output: Escape sequence (cont.)

- Example:



```
String str1 = "Hello\tworld\n";    // tab and newline
String str2 = "Double quoted \"hello\"";
String str3 = "A back-slash \\, another 2 back-slashes \\\\";
System.out.println(str1);
System.out.println(str2);
System.out.println(str3);
```



```
Hello   world

Double quoted "hello"
A back-slash \, another 2 back-slashes \\
```

End-of-Line (EOL)

- Newline (0AH) and Carriage Return (0DH) are used to represent the escape sequences `\n` and `\r`, respectively.
- They are used as line delimiters (or end-of-line) in text files.
- Unix and macOS (modern versions) use `\n` (0AH) as the EOL character.
- Windows uses a combination of `\r\n` (0D 0AH) as the EOL sequence.

Arithmetic Operators

- The operators for numeric data types include the standard arithmetic operators.

Operator	Mode	Usage	Description	Examples
+	Binary	x + y	Addition	1 + 2 = 3
	Unary	+x	Unary positive	1.1 + 2 = 3.1
-	Binary	x - y	Subtraction	1 - 2 = -1
	Unary	-x	Unary negate	1.1 - 2 = -0.9
*	Binary	x * y	Multiplication	2 * 3 = 6 2 * 3.0 = 6.0
/	Binary	x / y	Division	2/ 4 = 0 2 / 4.0 = 0.5
%	Binary	x % y	Modulus (Remainder)	2 % 3 = 2 2 % 3.0 = 2.0 7 % 3.0 = 1.0

Arithmetic Operators (cont.)

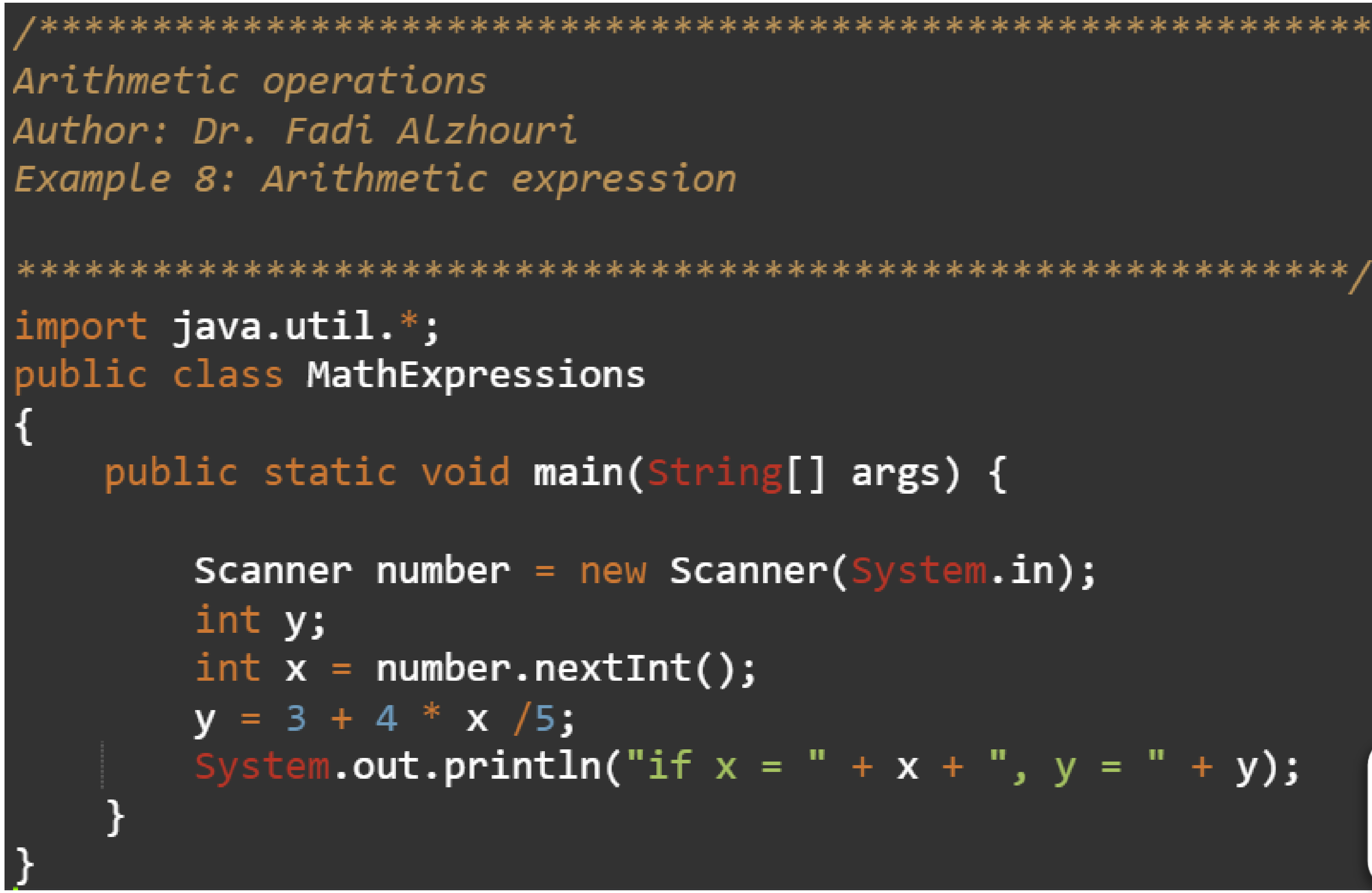
- Division (/): when both operands (numerator & denominator) of a division are integers, the result of the division is integer, and the fractional part is truncated.
- To get a double-point (real) result, one of the operands must be a double number.
  - Example: 3 / 2.0 = 1.0
- Modulus (%): the remainder is negative only if the dividend is negative.
- When both operators (dividend & divisor) are integers, the remainder is integer, while if one of them is double, the remainder is a double.

Arithmetic Expressions in Java

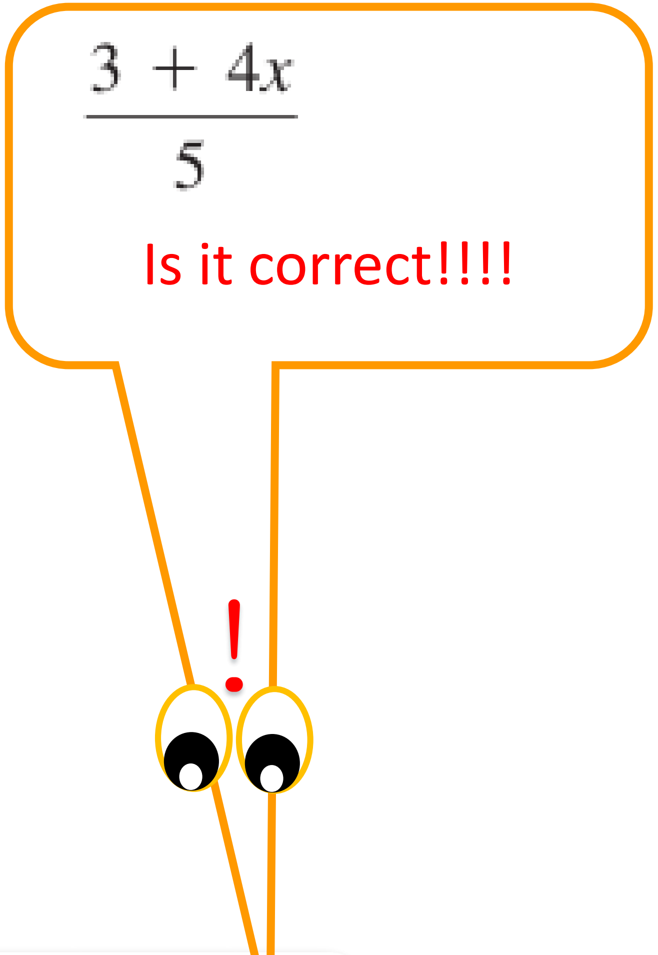
- Write a program to calculate the following expression: $$\frac{3 + 4x}{5}$$

Arithmetic Expressions in Java

- Write a program to calculate the following expression:

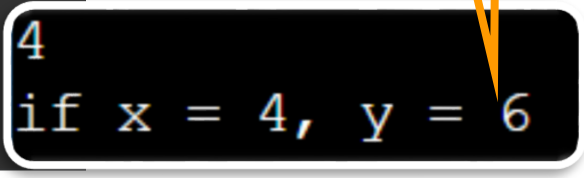


```
/*************************************************************
Arithmetic operations
Author: Dr. Fadi Alzhouri
Example 8: Arithmetic expression

*************************************************************/
import java.util.*;
public class MathExpressions
{
    public static void main(String[] args) {

        Scanner number = new Scanner(System.in);
        int y;
        int x = number.nextInt();
        y = 3 + 4 * x /5;
        System.out.println("if x = " + x + ", y = " + y);
    }
}
```



$$\frac{3 + 4x}{5}$$

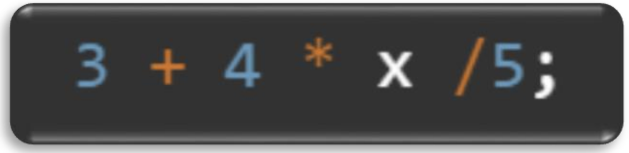
Is it correct!!!!



```
4
if x = 4, y = 6
```

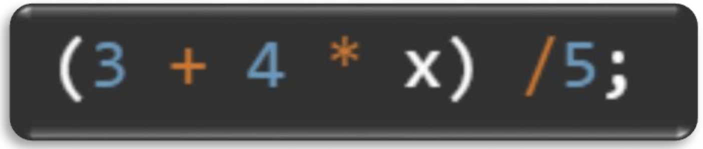
Arithmetic Expressions in Java (cont.)

- $\frac{3 + 4x}{5} \neq$



```
3 + 4 * x /5;
```

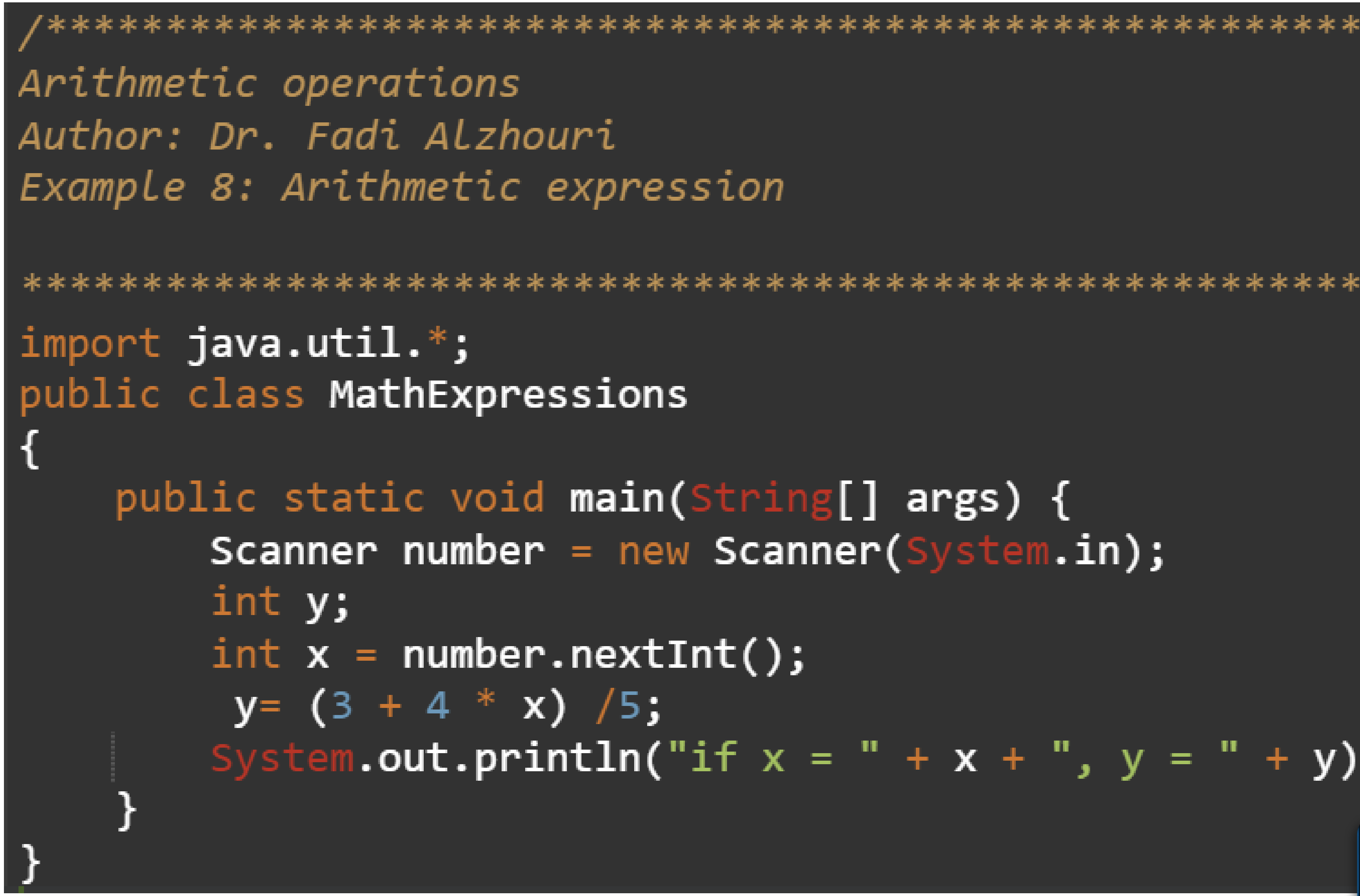
- $\frac{3 + 4x}{5} =$



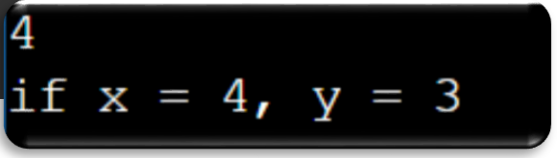
```
(3 + 4 * x) /5;
```

- Java evaluates arithmetic expressions based on operator precedence and associativity rules.

Arithmetic Expressions in Java (cont.)



```
/***********************************************************
Arithmetic operations
Author: Dr. Fadi Alzhouri
Example 8: Arithmetic expression

************************************************************/
import java.util.*;
public class MathExpressions
{
    public static void main(String[] args) {
        Scanner number = new Scanner(System.in);
        int y;
        int x = number.nextInt();
         y= (3 + 4 * x) /5;
        System.out.println("if x = " + x + ", y = " + y);
    }
}
```



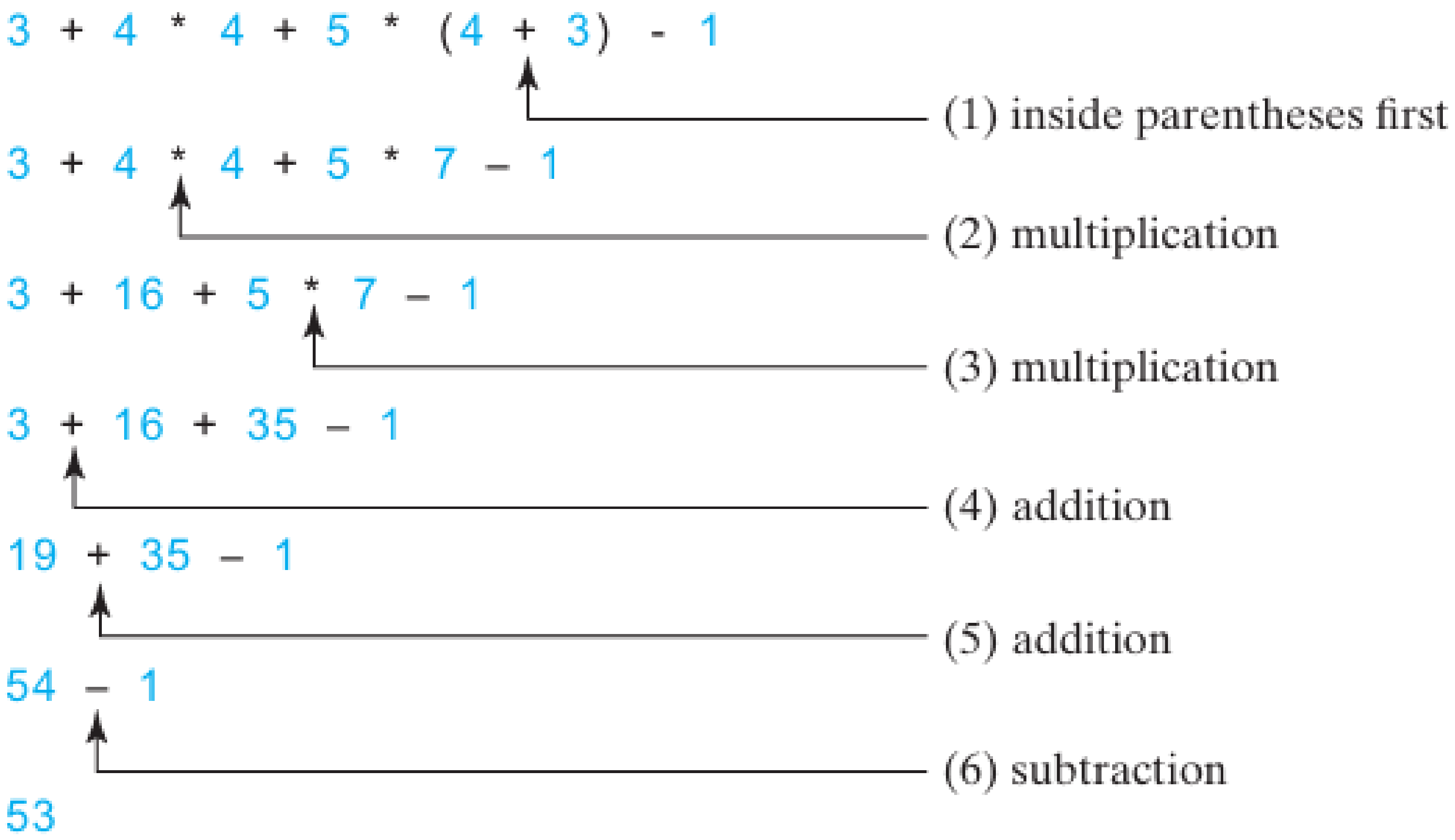
```
4
if x = 4, y = 3
```

Operators Precedence

1. Parentheses () have the highest precedence and can be used to change the order of evaluation.
2. Unary '-' (negate) and '+' (positive) have next higher precedence.
3. The multiplication (*), division (/) and modulus (%) have the same precedence.
4. Addition (+) and subtraction (-)
5. Within the same precedence level (i.e., addition/subtraction and multiplication/division/modulus), the expression is evaluated from left to right (called left-associative).

Operators Precedence (cont.)

Example:



3 + 4 * 4 + 5 * (4 + 3) - 1

(1) inside parentheses first

3 + 4 * 4 + 5 * 7 – 1

(2) multiplication

3 + 16 + 5 * 7 – 1

(3) multiplication

3 + 16 + 35 – 1

(4) addition

19 + 35 – 1

(5) addition

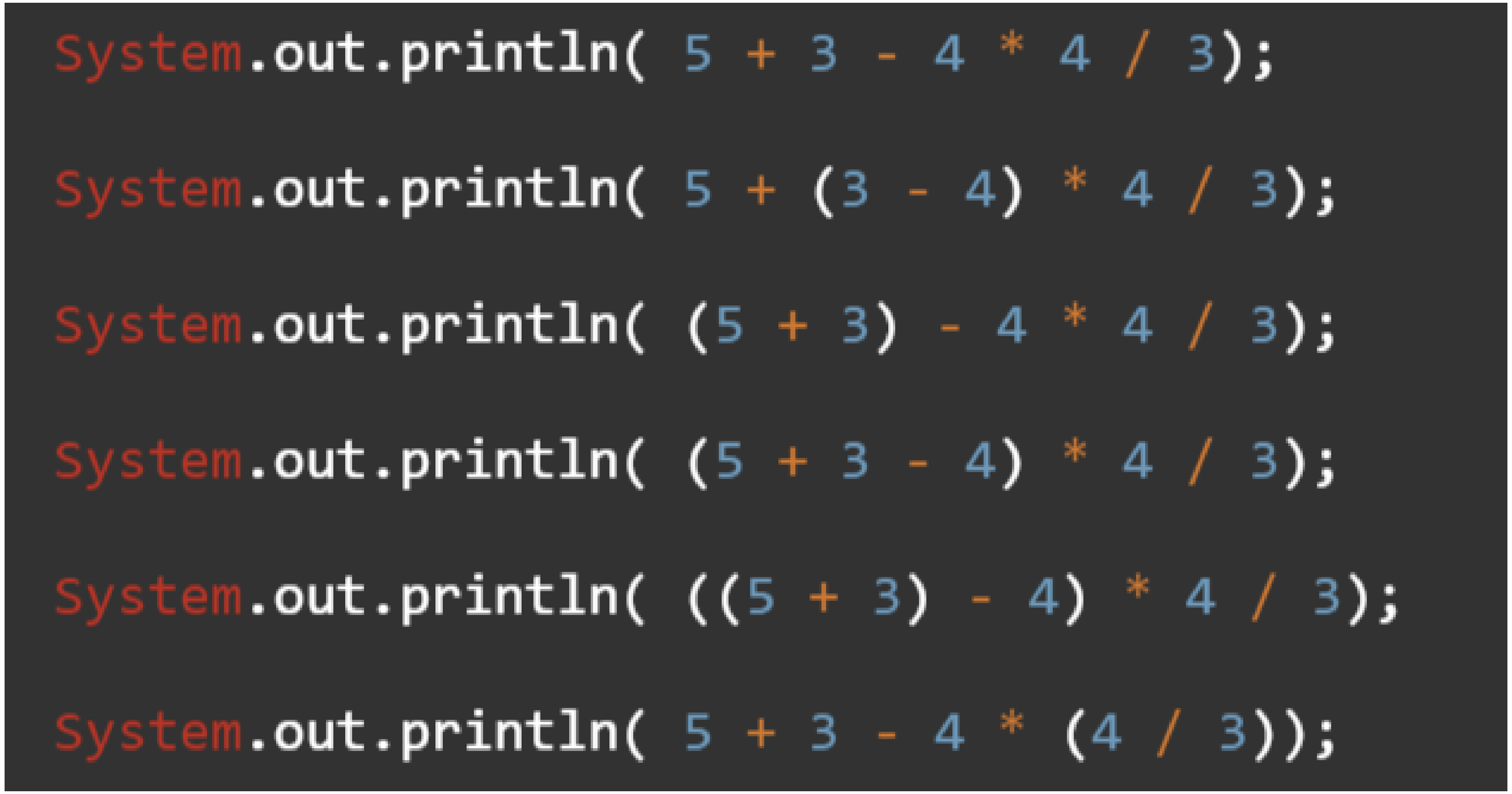
54 – 1

(6) subtraction

53

Operators Precedence (cont.)

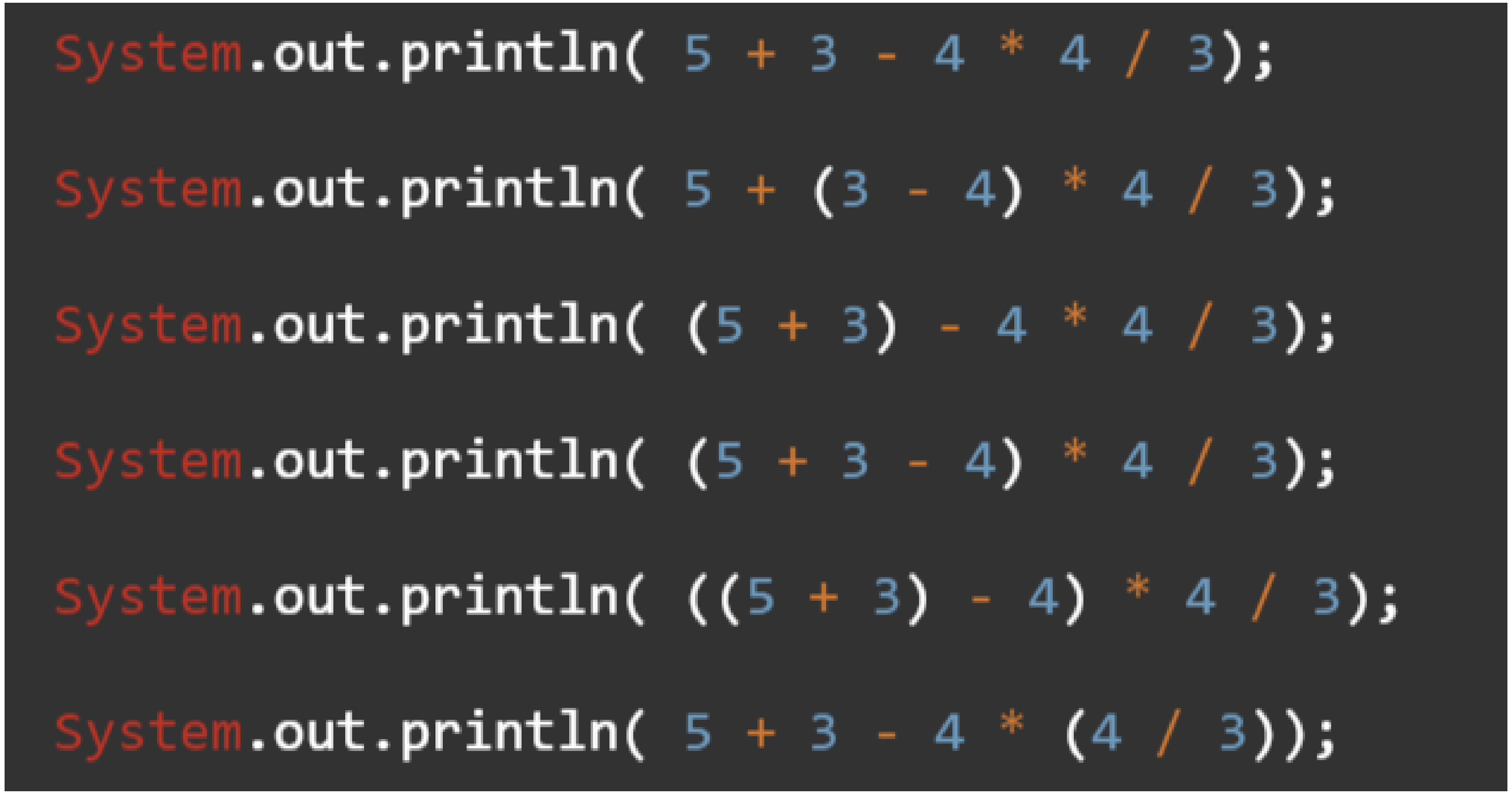
Exercise: evaluate each expression



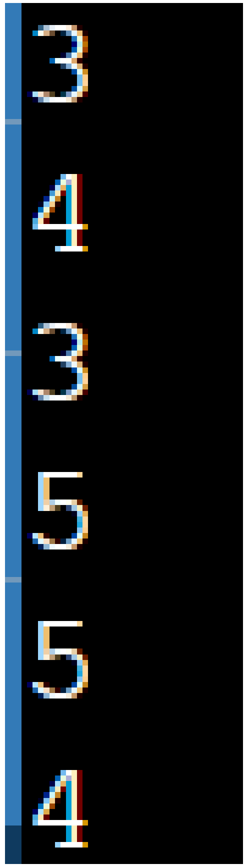
```
System.out.println( 5 + 3 - 4 * 4 / 3);

System.out.println( 5 + (3 - 4) * 4 / 3);

System.out.println( (5 + 3) - 4 * 4 / 3);

System.out.println( (5 + 3 - 4) * 4 / 3);

System.out.println( ((5 + 3) - 4) * 4 / 3);

System.out.println( 5 + 3 - 4 * (4 / 3));
```

Operators Precedence (cont.)

Exercise: evaluate each expression



```java
System.out.println( 5 + 3 - 4 * 4 / 3);

System.out.println( 5 + (3 - 4) * 4 / 3);

System.out.println( (5 + 3) - 4 * 4 / 3);

System.out.println( (5 + 3 - 4) * 4 / 3);

System.out.println( ((5 + 3) - 4) * 4 / 3);

System.out.println( 5 + 3 - 4 * (4 / 3));
```



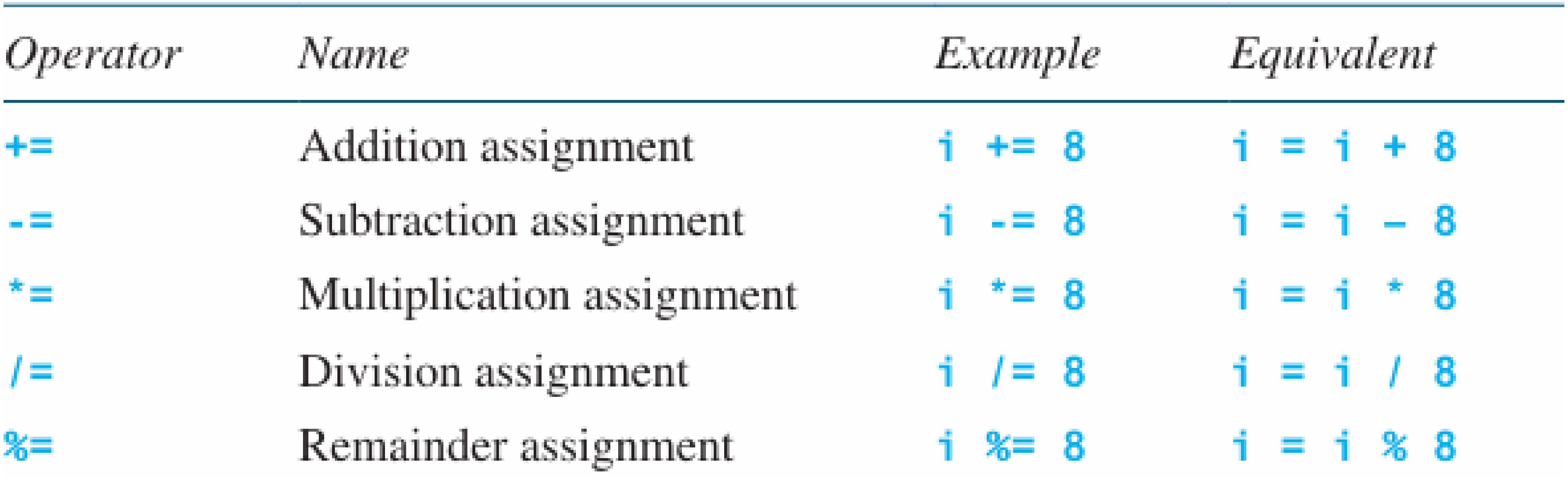
```
3
4
3
5
5
4
```

Compound Assignment Operators

- The operators +, -, *, /, and % can be combined with the assignment operator to form augmented operators.
- The current value of a variable is used, modified, then reassigned back to the same variable.
- `count = count + 1;` → `count += 1;`

Compound addition

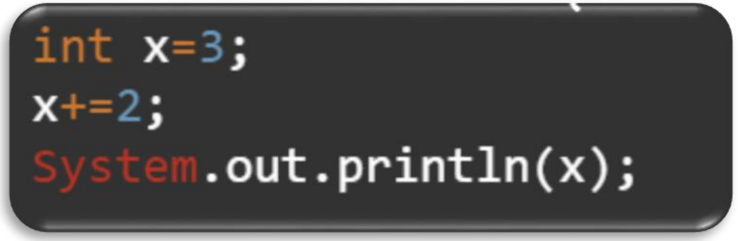
Compound Assignment Operators (cont.)



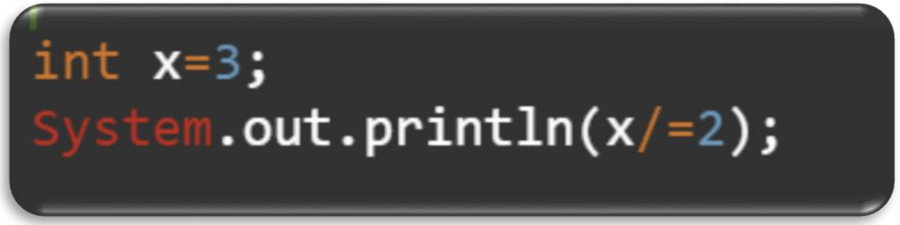
Operator	*Name*	*Example*	*Equivalent*
`+=`	Addition assignment	`i += 8`	`i = i + 8`
`-=`	Subtraction assignment	`i -= 8`	`i = i - 8`
`*=`	Multiplication assignment	`i *= 8`	`i = i * 8`
`/=`	Division assignment	`i /= 8`	`i = i / 8`
`%=`	Remainder assignment	`i %= 8`	`i = i % 8`

- Caution: there are no spaces in the compound assignment operators.
    - For example, + = should be +=.

Compound Assignment Operators (cont.)



```java
int x=3;
x+=2;
System.out.println(x);
```

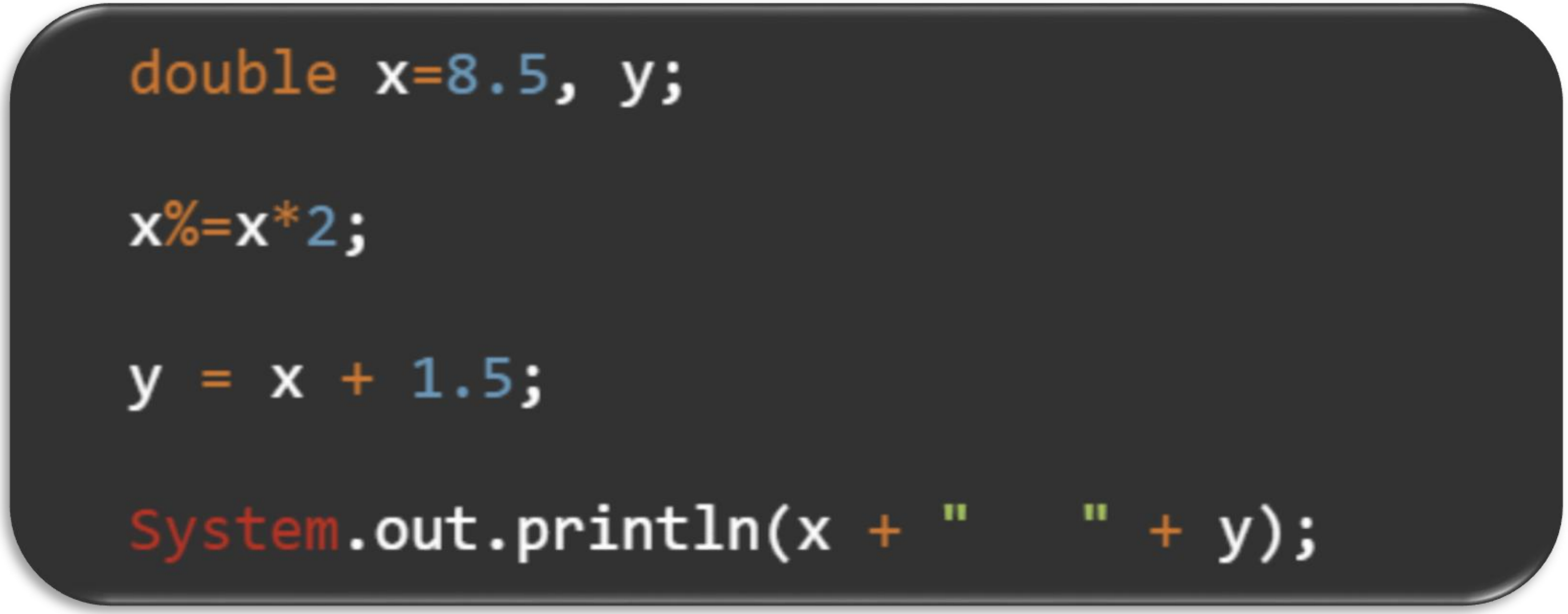


```java
int x=3;
System.out.println(x/=2);
```

- Caution: there are no spaces in the compound assignment operators.
  - For example, + = should be +=.

Compound Assignment Operators (cont.)

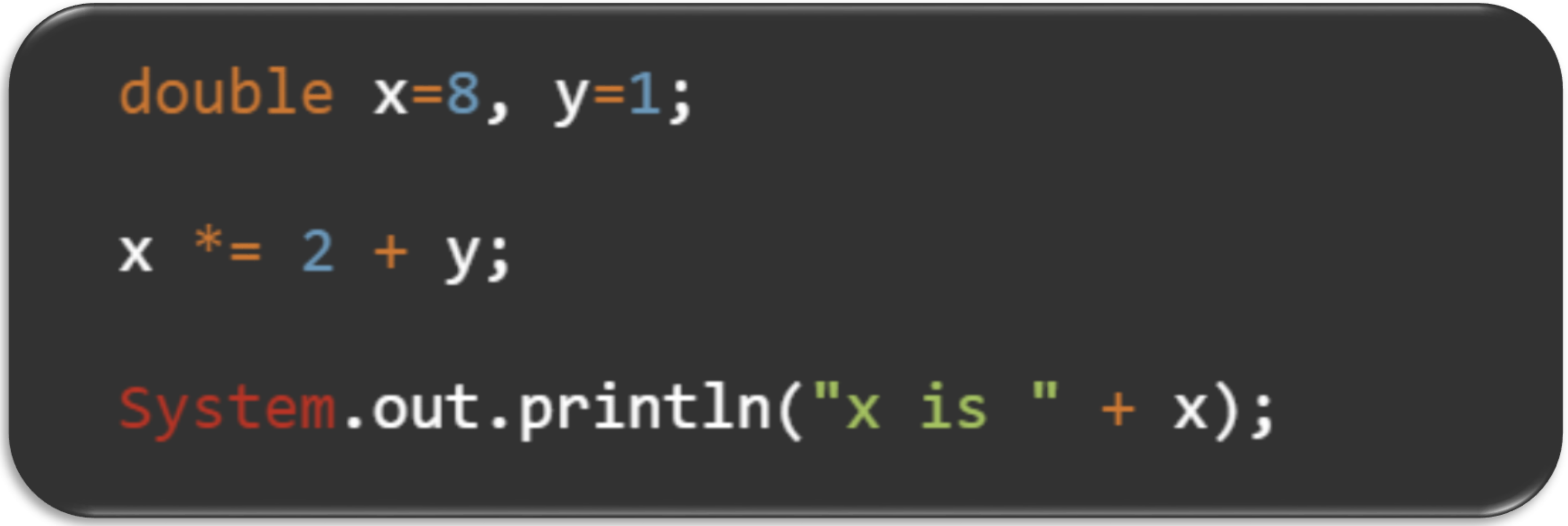
- Exercise 1:



```
double x=8.5, y;

x%=x*2;

y = x + 1.5;

System.out.println(x + "   " + y);
```

Compound Assignment Operators (cont.)

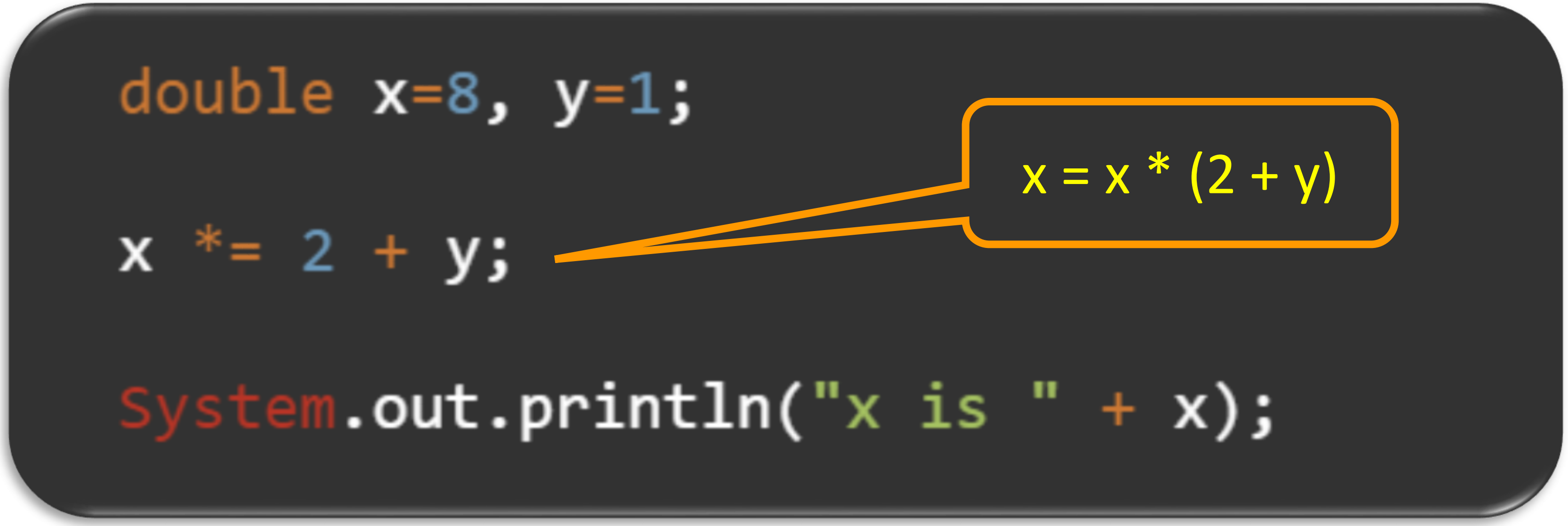
- Exercise 2:



```
double x=8, y=1;

x *= 2 + y;

System.out.println("x is " + x);
```

Compound Assignment Operators (cont.)

- Exercise 2:



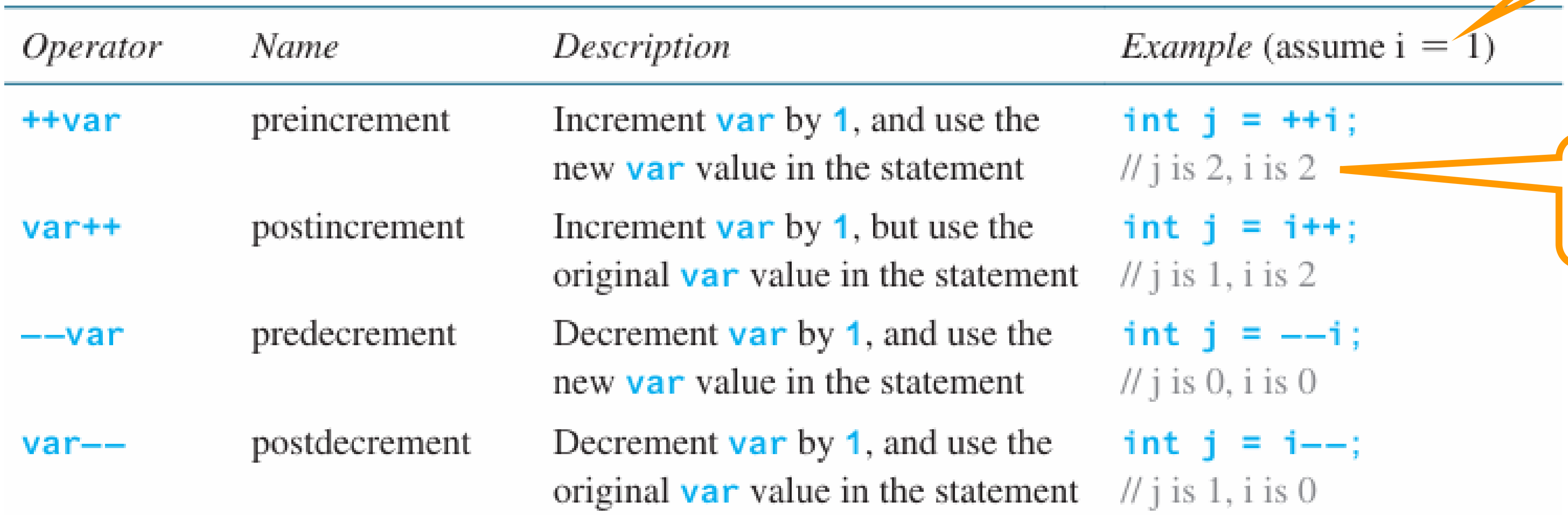
```
double x=8, y=1;

x *= 2 + y;

System.out.println("x is " + x);
```

x = x * (2 + y)

Increment and Decrement Operators

- The increment operator (+ +) increments a variable by 1.
- The decrement operator (- -) decrements a variable by 1.

TABLE 2.5 Increment and Decrement Operators



Operator	*Name*	*Description*	*Example* (assume i = 1)
`++var`	preincrement	Increment `var` by `1`, and use the new `var` value in the statement	`int j = ++i;` // j is 2, i is 2
`var++`	postincrement	Increment `var` by `1`, but use the original `var` value in the statement	`int j = i++;` // j is 1, i is 2
`--var`	predecrement	Decrement `var` by `1`, and use the new `var` value in the statement	`int j = --i;` // j is 0, i is 0
`var--`	postdecrement	Decrement `var` by `1`, and use the original `var` value in the statement	`int j = i--;` // j is 1, i is 0

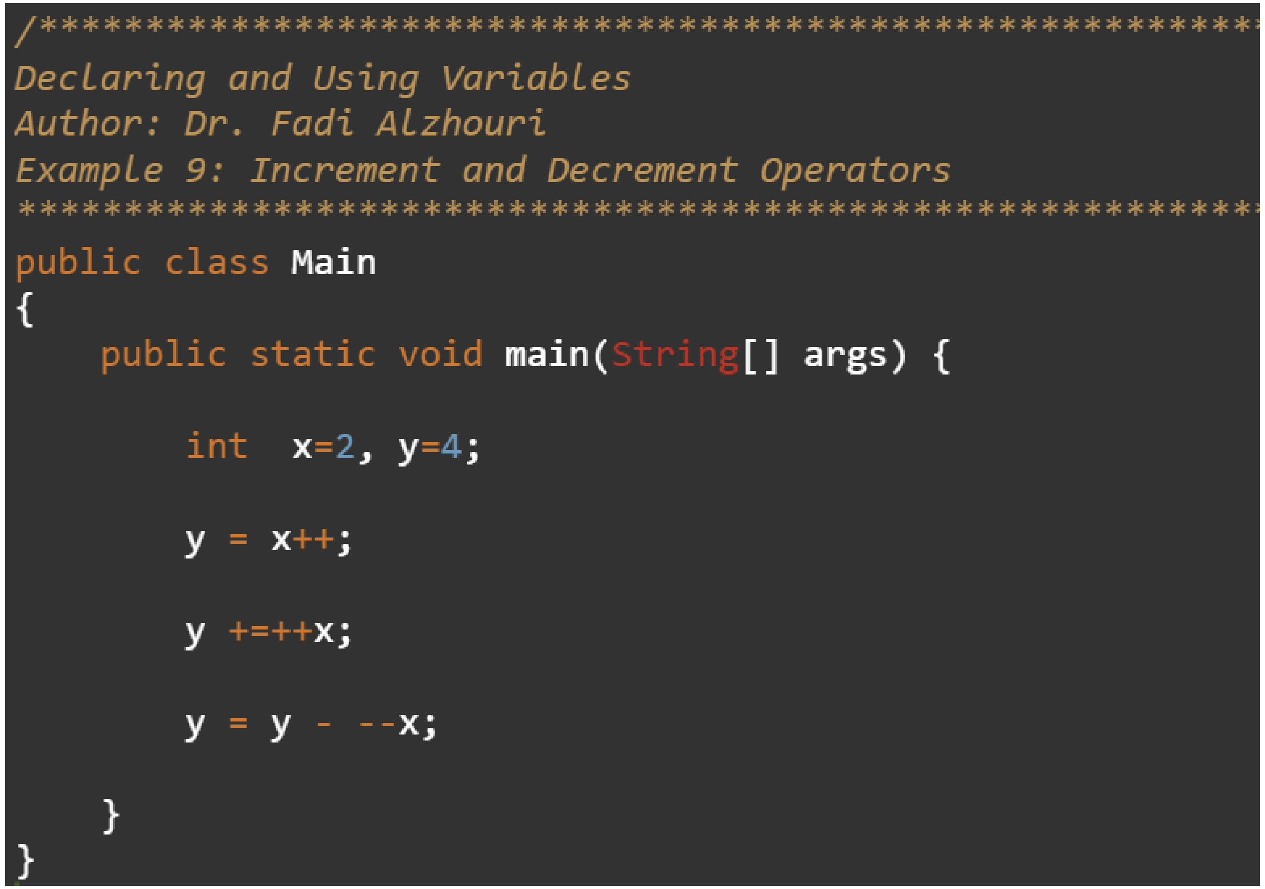
old value

new value

- Work with all primitive data types except Boolean.

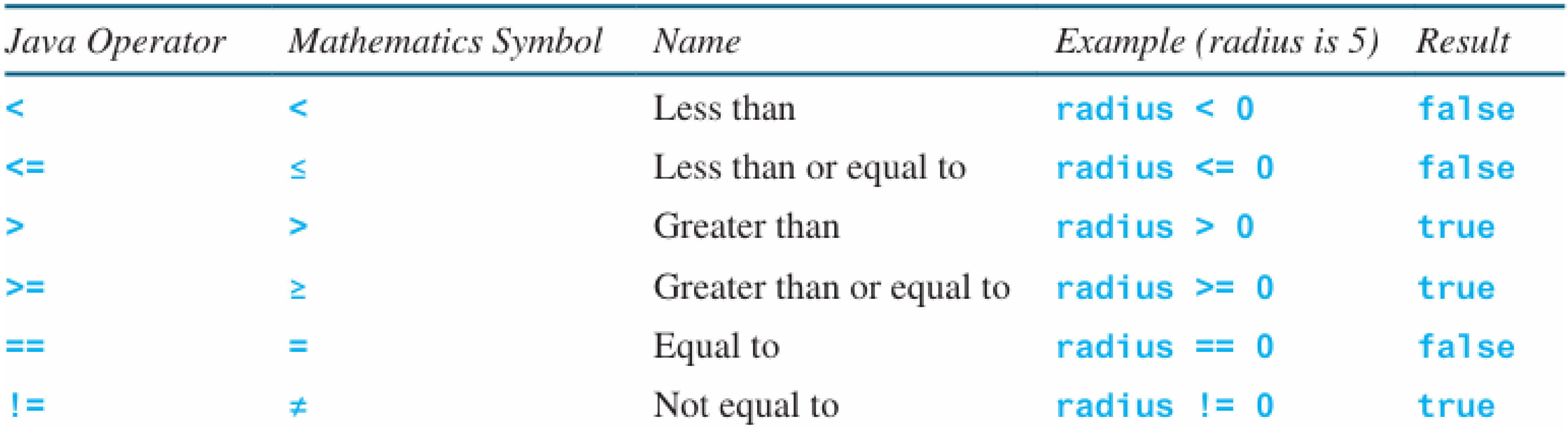
Increment and Decrement Operators (cont.)

- Exercise:



```
/*********************************************************
Declaring and Using Variables
Author: Dr. Fadi Alzhouri
Example 9: Increment and Decrement Operators
**********************************************************
public class Main
{
    public static void main(String[] args) {

        int  x=2, y=4;

        y = x++;

        y +=++x;

        y = y - --x;

    }
}
```

Relational (comparison) Operators (cont.)

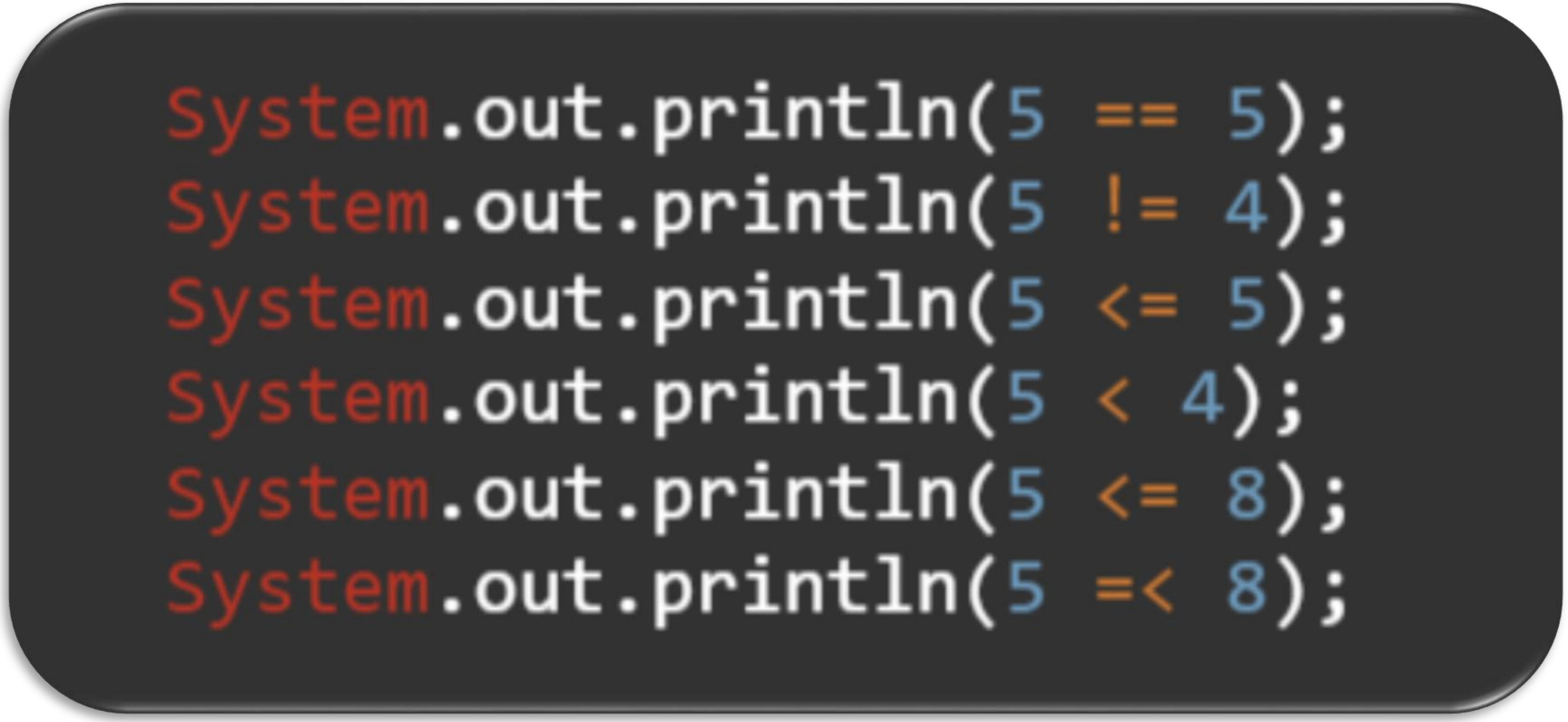


Java Operator	*Mathematics Symbol*	*Name*	*Example (radius is 5)*	*Result*
`<`	<	Less than	`radius < 0`	`false`
`<=`	≤	Less than or equal to	`radius <= 0`	`false`
`>`	>	Greater than	`radius > 0`	`true`
`>=`	≥	Greater than or equal to	`radius >= 0`	`true`
`==`	=	Equal to	`radius == 0`	`false`
`!=`	≠	Not equal to	`radius != 0`	`true`

- The equality testing operator is two equal signs (==), not a single equal sign (=).
- The equal sign (=) symbol is for assignment.

Relational (comparison) Operators (cont.)

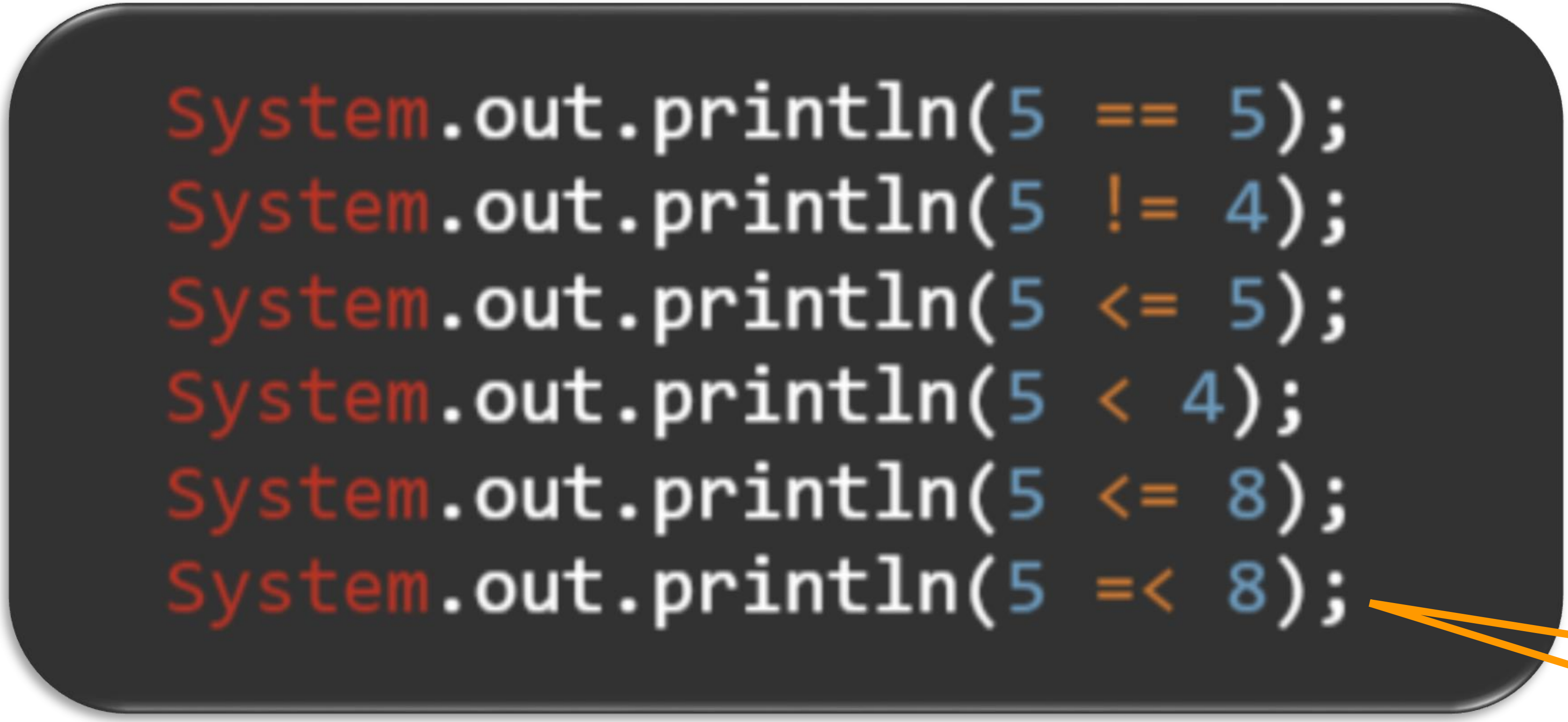
Exercise:



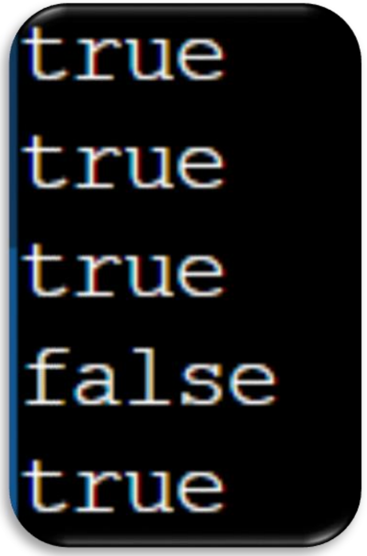
```
System.out.println(5 == 5);
System.out.println(5 != 4);
System.out.println(5 <= 5);
System.out.println(5 < 4);
System.out.println(5 <= 8);
System.out.println(5 =< 8);
```

Relational (comparison) Operators (cont.)

Exercise:



```
System.out.println(5 == 5);
System.out.println(5 != 4);
System.out.println(5 <= 5);
System.out.println(5 < 4);
System.out.println(5 <= 8);
System.out.println(5 =< 8);
```

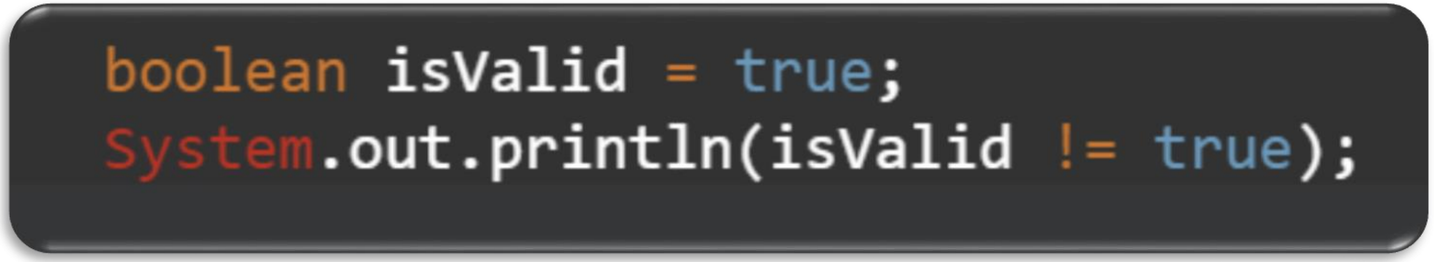


```
true
true
true
false
true
```

error

Relational (comparison) Operators (cont.)

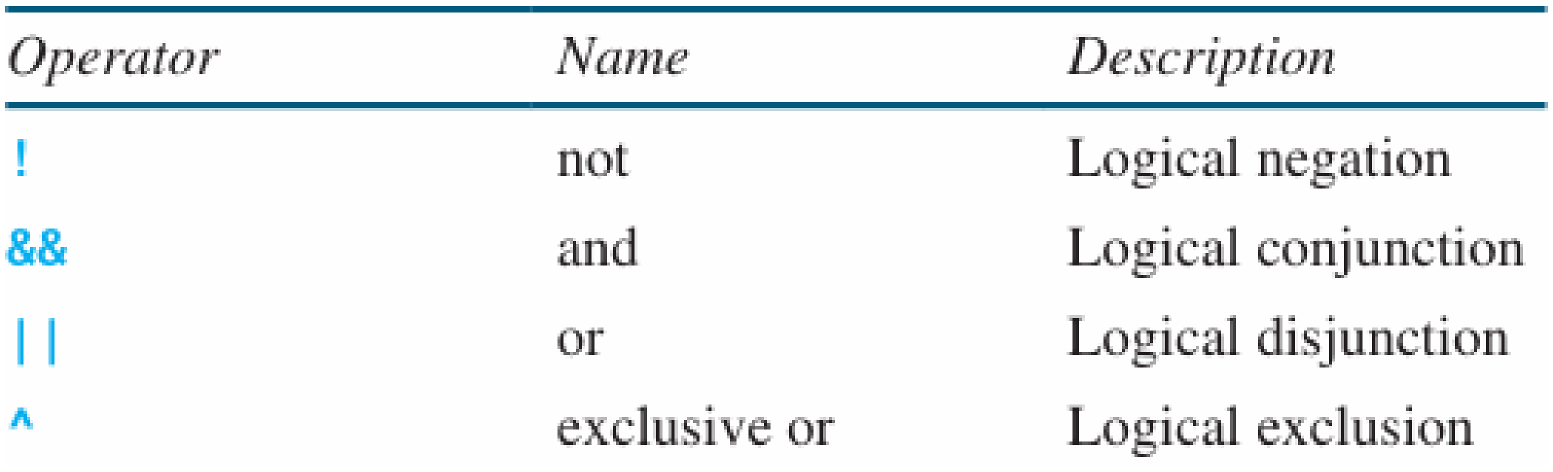
Exercise 2:



```
boolean isValid = true;
System.out.println(isValid != true);
```

Logical Operators

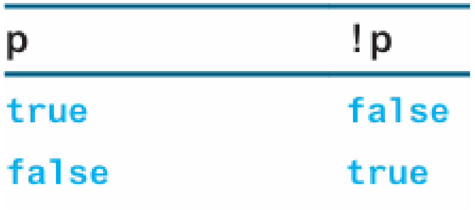
- The logical operators can be used to create a compound boolean expression.
- It operates on boolean operands only, in descending order of precedence



Operator	*Name*	*Description*
!	not	Logical negation
&&	and	Logical conjunction
||	or	Logical disjunction
^	exclusive or	Logical exclusion

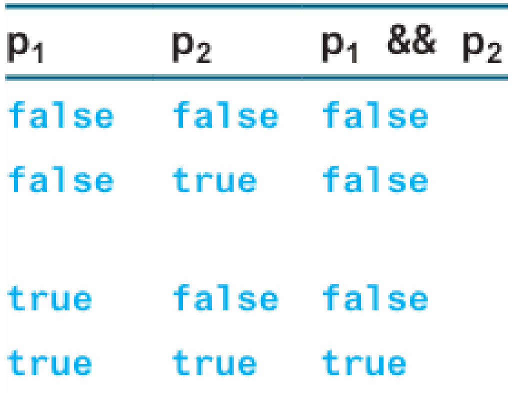
Logical Operators (cont.)

- Truth tables for all logical operators



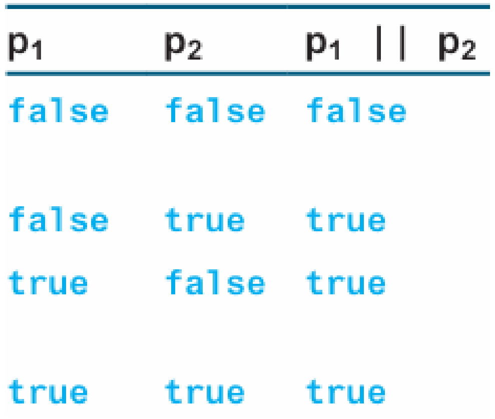
p	!p
true	false
false	true

Not



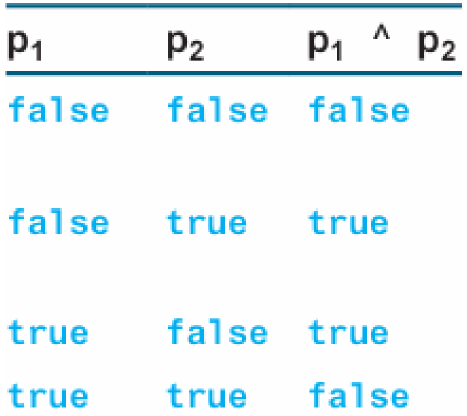
p_1	p_2	p_1 && p_2
false	false	false
false	true	false
true	false	false
true	true	true

And



p_1	p_2	p_1 || p_2
false	false	false
false	true	true
true	false	true
true	true	true

Or

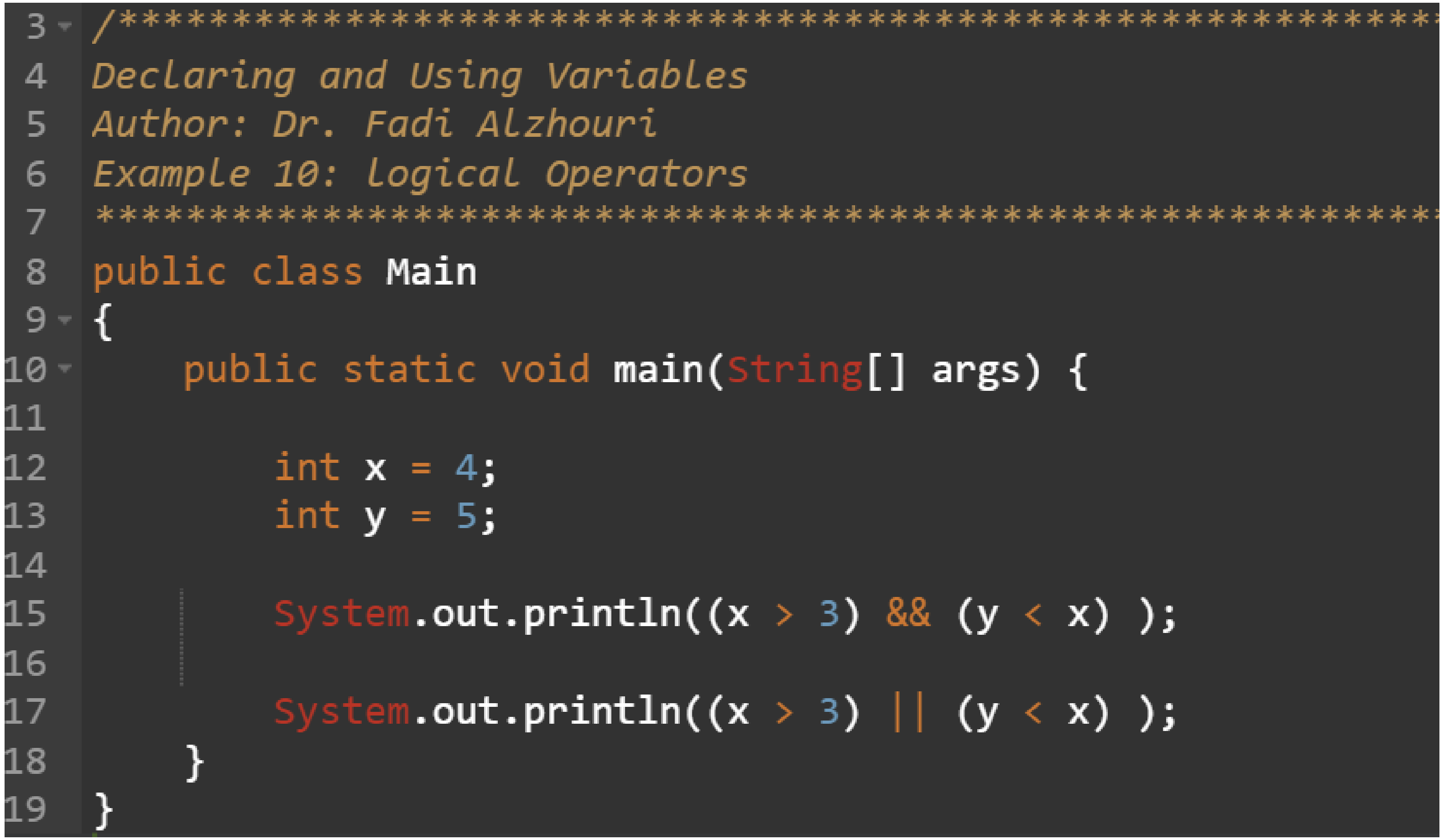


p_1	p_2	p_1 ^ p_2
false	false	false
false	true	true
true	false	true
true	true	false

XOR

Logical Operators (cont.)

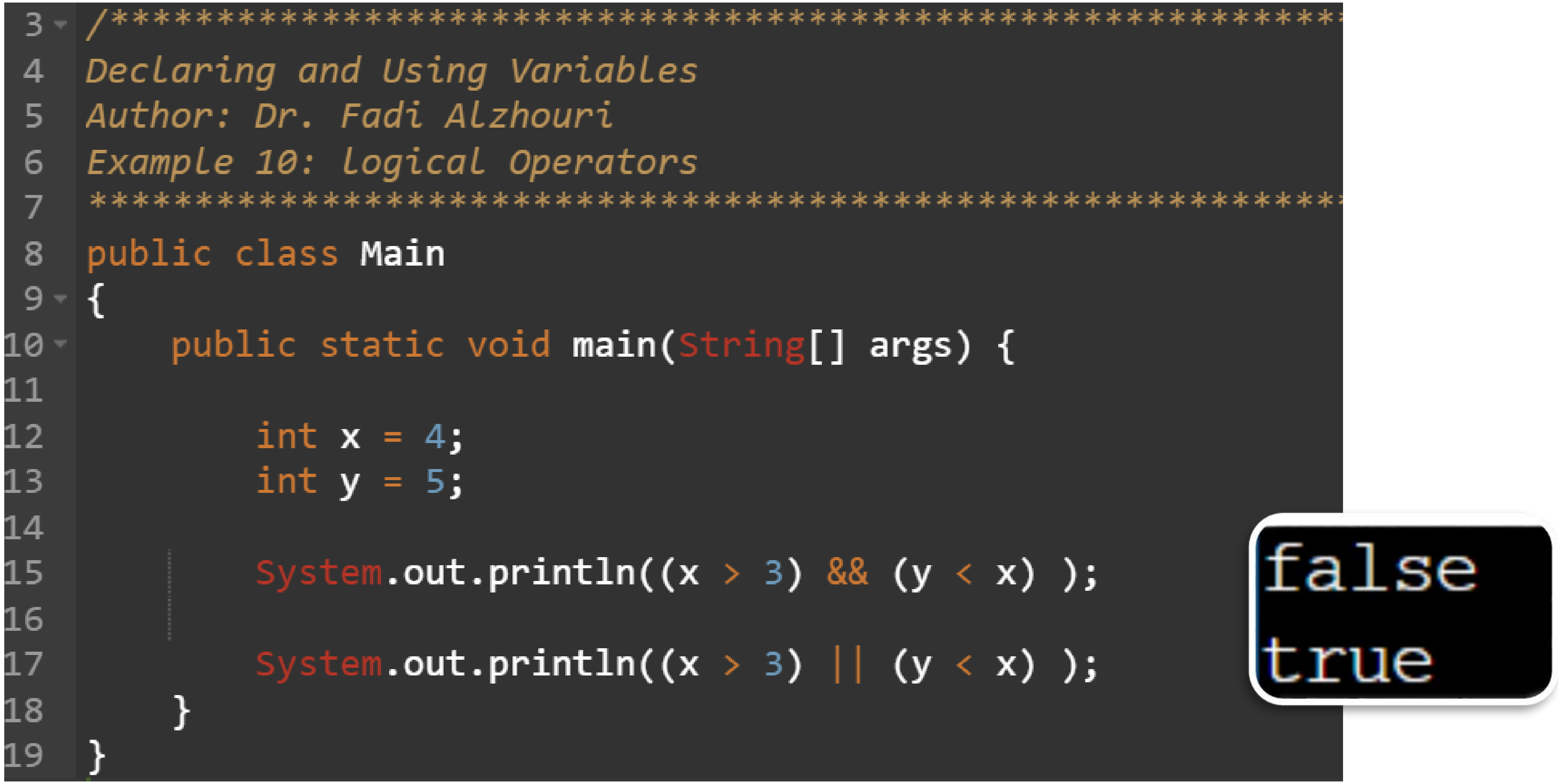
- Exercise:



```java
/**********************************************************
Declaring and Using Variables
Author: Dr. Fadi Alzhouri
Example 10: logical Operators
**********************************************************
public class Main
{
    public static void main(String[] args) {

        int x = 4;
        int y = 5;

        System.out.println((x > 3) && (y < x) );

        System.out.println((x > 3) || (y < x) );
    }
}
```

Logical Operators (cont.)

- Exercise:



```
/**********************************************************
Declaring and Using Variables
Author: Dr. Fadi Alzhouri
Example 10: logical Operators
***********************************************************
public class Main
{
    public static void main(String[] args) {

        int x = 4;
        int y = 5;

        System.out.println((x > 3) && (y < x) );

        System.out.println((x > 3) || (y < x) );
    }
}
```

```
false
true
```

References

- **Introduction to Java Programming, Brief Version, Global Edition, 11th edition,** Published by Pearson (June 21, 2018) © 2018