دانشگاه صنعتی امیر کبیر
(پلی تکنیک تهران)

دانشکده مهندسی کامپیوتر
و فناوری اطلاعات

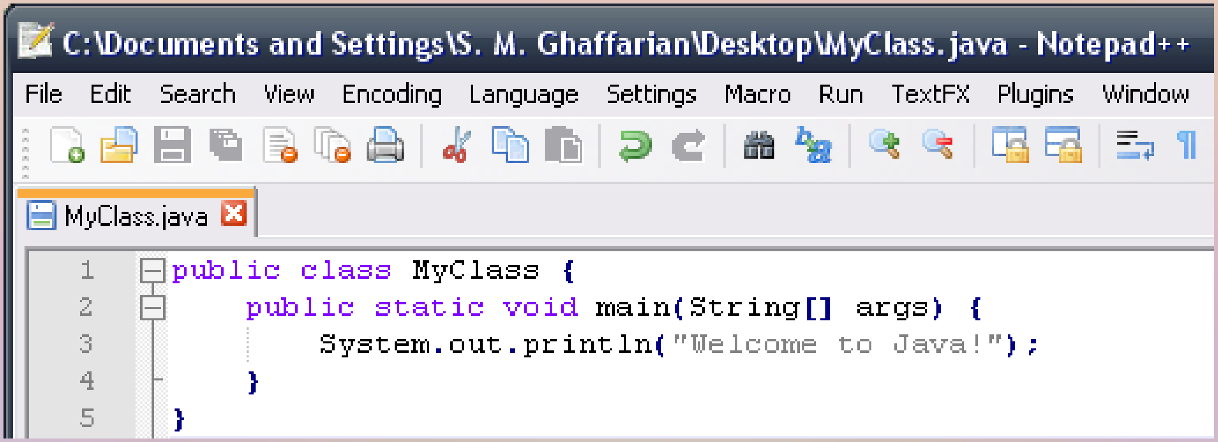
برنامه نویسی پیشرفته

برنامه نویسی ساخت‌یافته با Java

زمستان ۱۳۹۶

Java Basics

- Each Java file includes a **public class** with the same name as the file-name:


C:\Documents and Settings\S. M. Ghaffarian\Desktop\MyClass.java - Notepad++
File Edit Search View Encoding Language Settings Macro Run TextFX Plugins Window
MyClass.java

```
public class MyClass {
    public static void main(String[] args) {
        System.out.println("Welcome to Java!");
    }
}
```

- Just like the C language, the **main method** is the program's starting point.

Java Basics (continued ...)

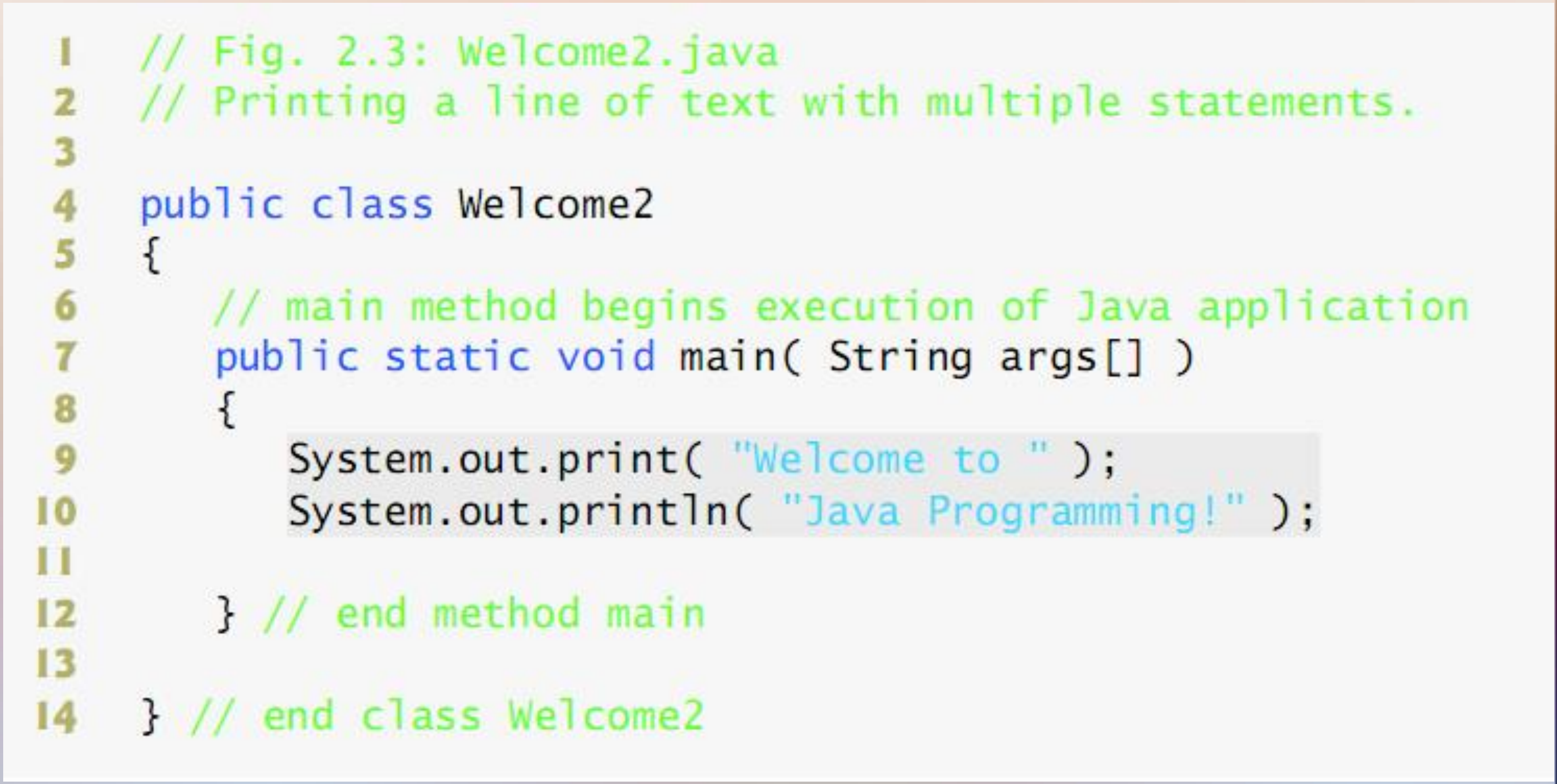
- Java is a C based language and so the syntax of Java programs is very similar to the syntax of C / C++

- The primitive data types are mostly similar :
  - byte, int, short, long, float, double, boolean, char

- Control statements are also mostly the same :
  - if, else, switch-case, while, for, do-while, continue, break

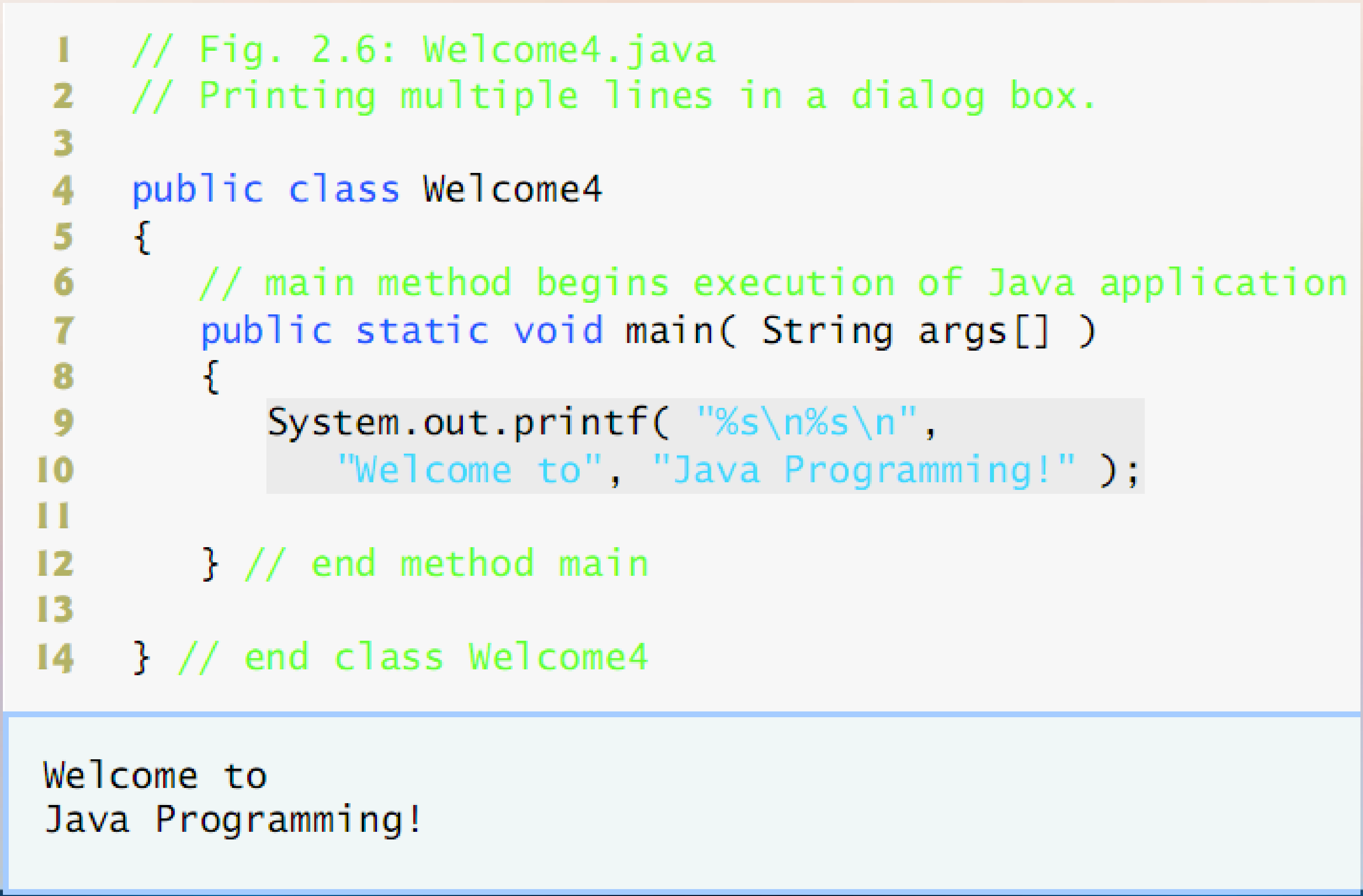
- Syntax of Java methods is also similar to C functions

print and println Methods



```java
// Fig. 2.3: Welcome2.java
// Printing a line of text with multiple statements.

public class Welcome2
{
   // main method begins execution of Java application
   public static void main( String args[] )
   {
      System.out.print( "Welcome to " );
      System.out.println( "Java Programming!" );

   } // end method main

} // end class Welcome2
```

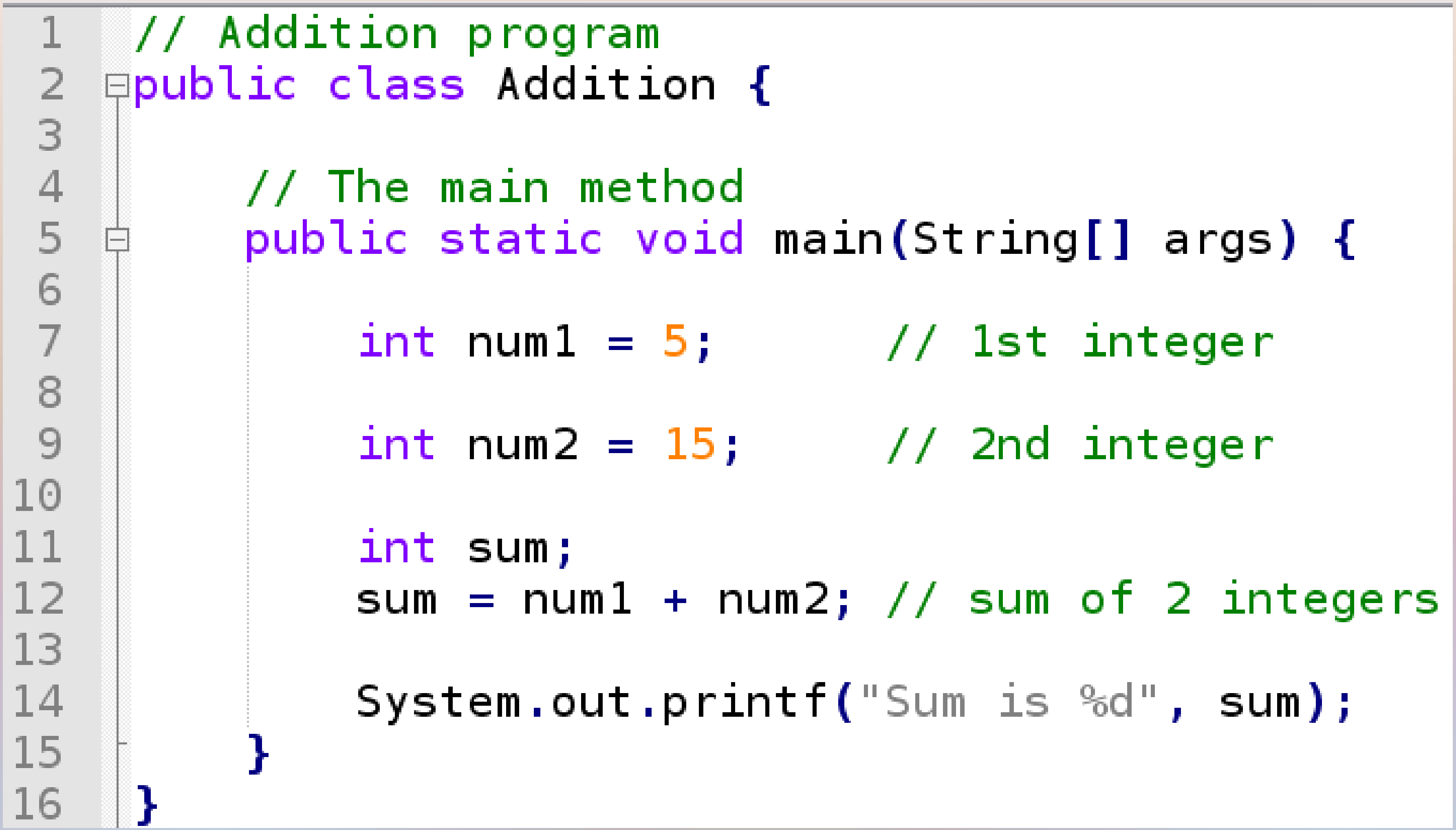
Good Old printf !!



```java
// Fig. 2.6: Welcome4.java
// Printing multiple lines in a dialog box.

public class Welcome4
{
   // main method begins execution of Java application
   public static void main( String args[] )
   {
      System.out.printf( "%s\n%s\n",
         "Welcome to", "Java Programming!" );

   } // end method main

} // end class Welcome4
```

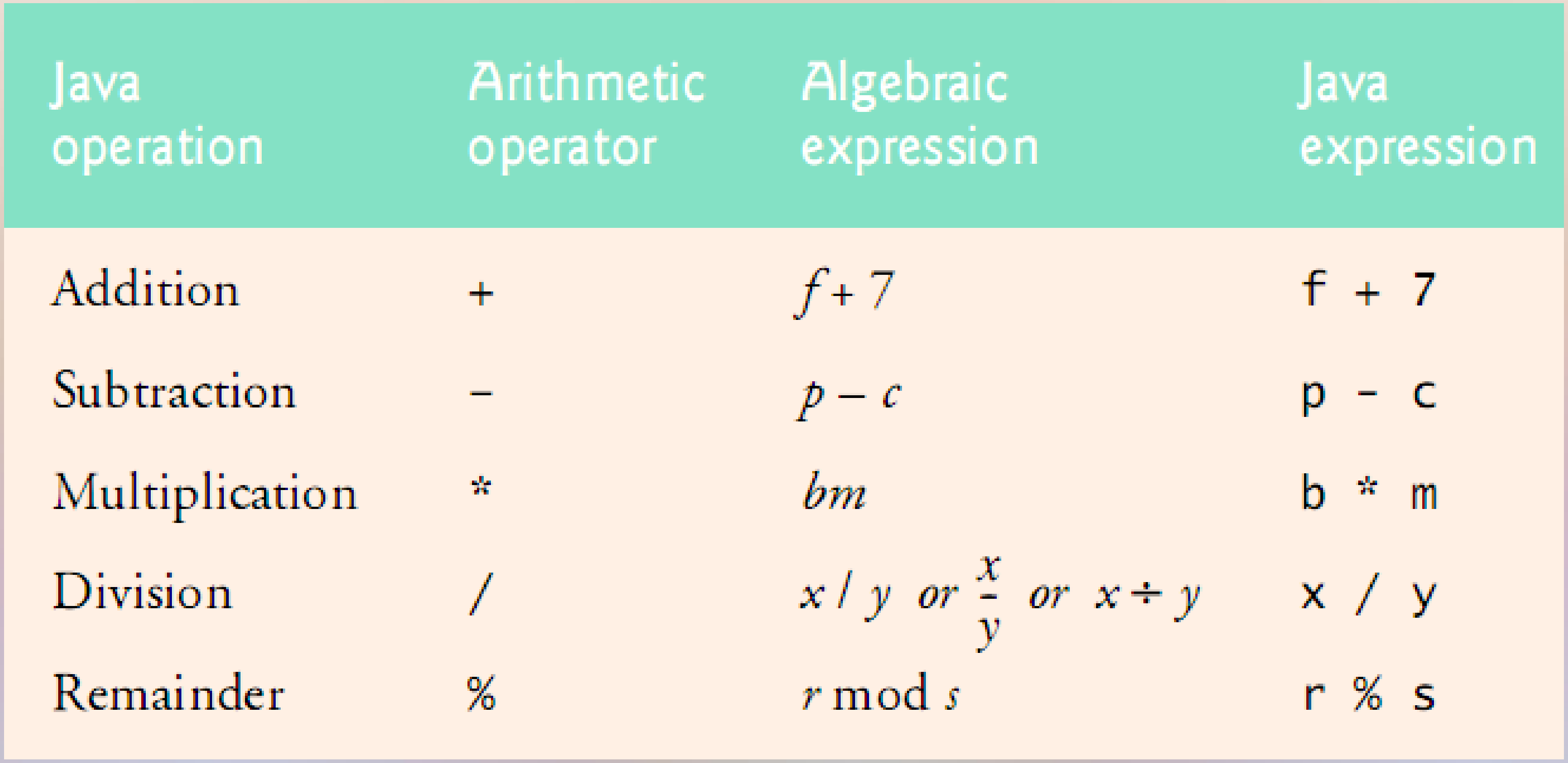
```
Welcome to
Java Programming!
```

Simple Arithmetic Example



```java
// Addition program
public class Addition {

    // The main method
    public static void main(String[] args) {

        int num1 = 5;      // 1st integer

        int num2 = 15;     // 2nd integer

        int sum;
        sum = num1 + num2; // sum of 2 integers

        System.out.printf("Sum is %d", sum);
    }
}
```

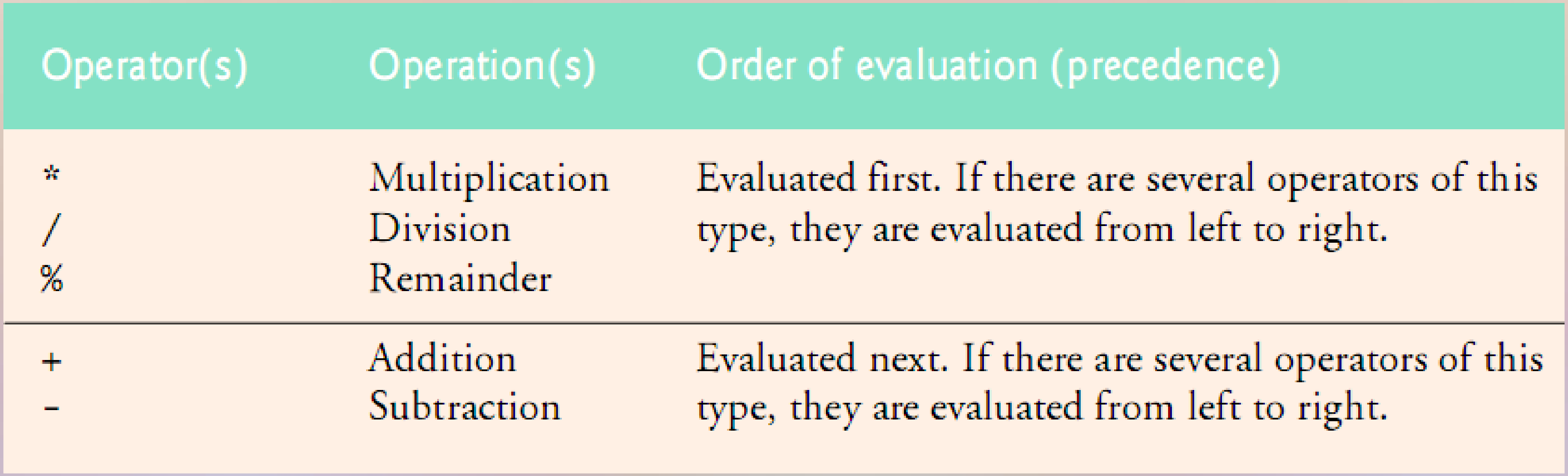
Arithmetic Operators



Java operation	Arithmetic operator	Algebraic expression	Java expression
Addition	+	$f + 7$	`f + 7`
Subtraction	–	$p - c$	`p - c`
Multiplication	*	bm	`b * m`
Division	/	x / y *or* $\frac{x}{y}$ *or* $x \div y$	`x / y`
Remainder	%	$r \bmod s$	`r % s`

Arithmetic Operators (continued ...)

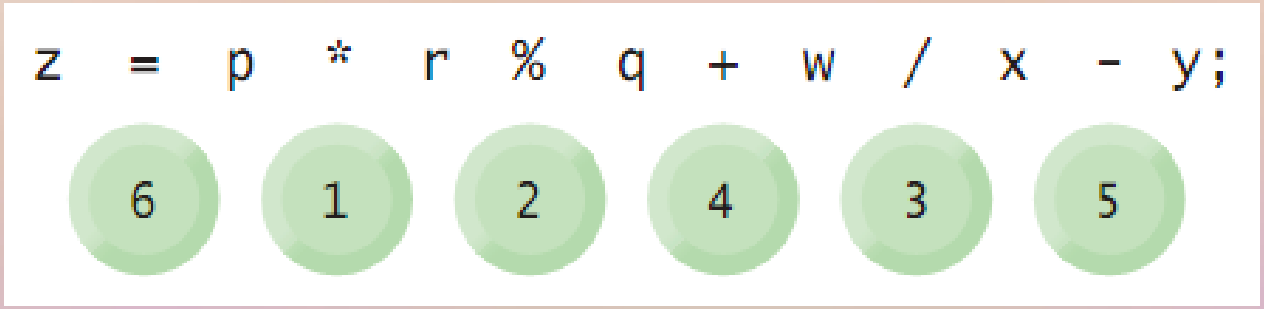
Operator Precedence

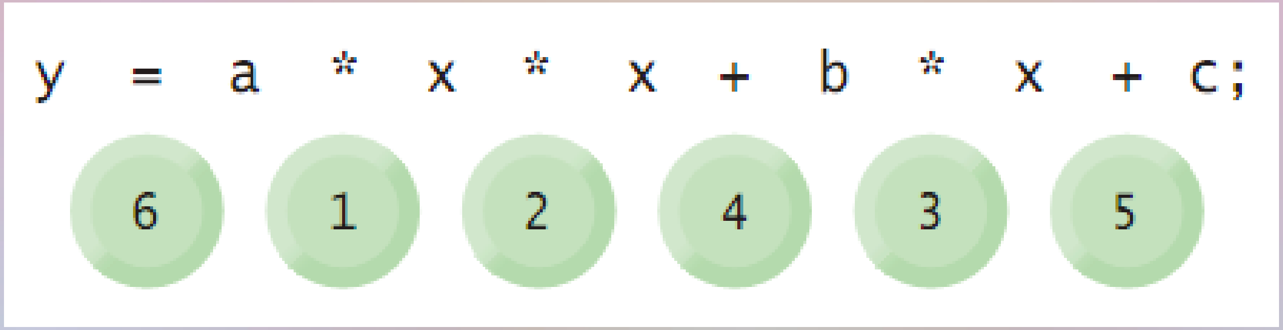


Operator(s)	Operation(s)	Order of evaluation (precedence)
* / %	Multiplication Division Remainder	Evaluated first. If there are several operators of this type, they are evaluated from left to right.
+ -	Addition Subtraction	Evaluated next. If there are several operators of this type, they are evaluated from left to right.

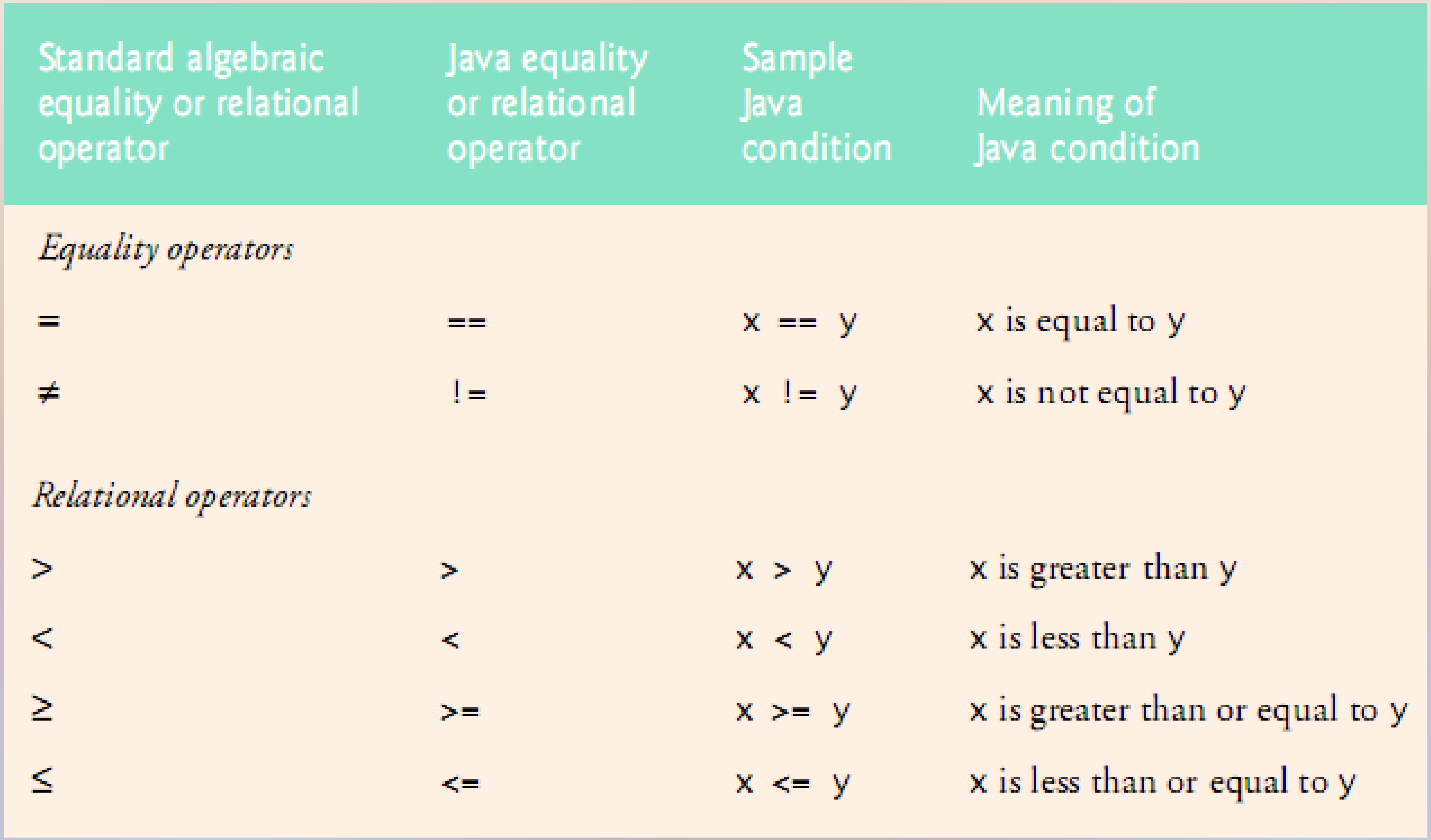
Arithmetic Operators (continued ...)

- Two Examples of operator precedence:



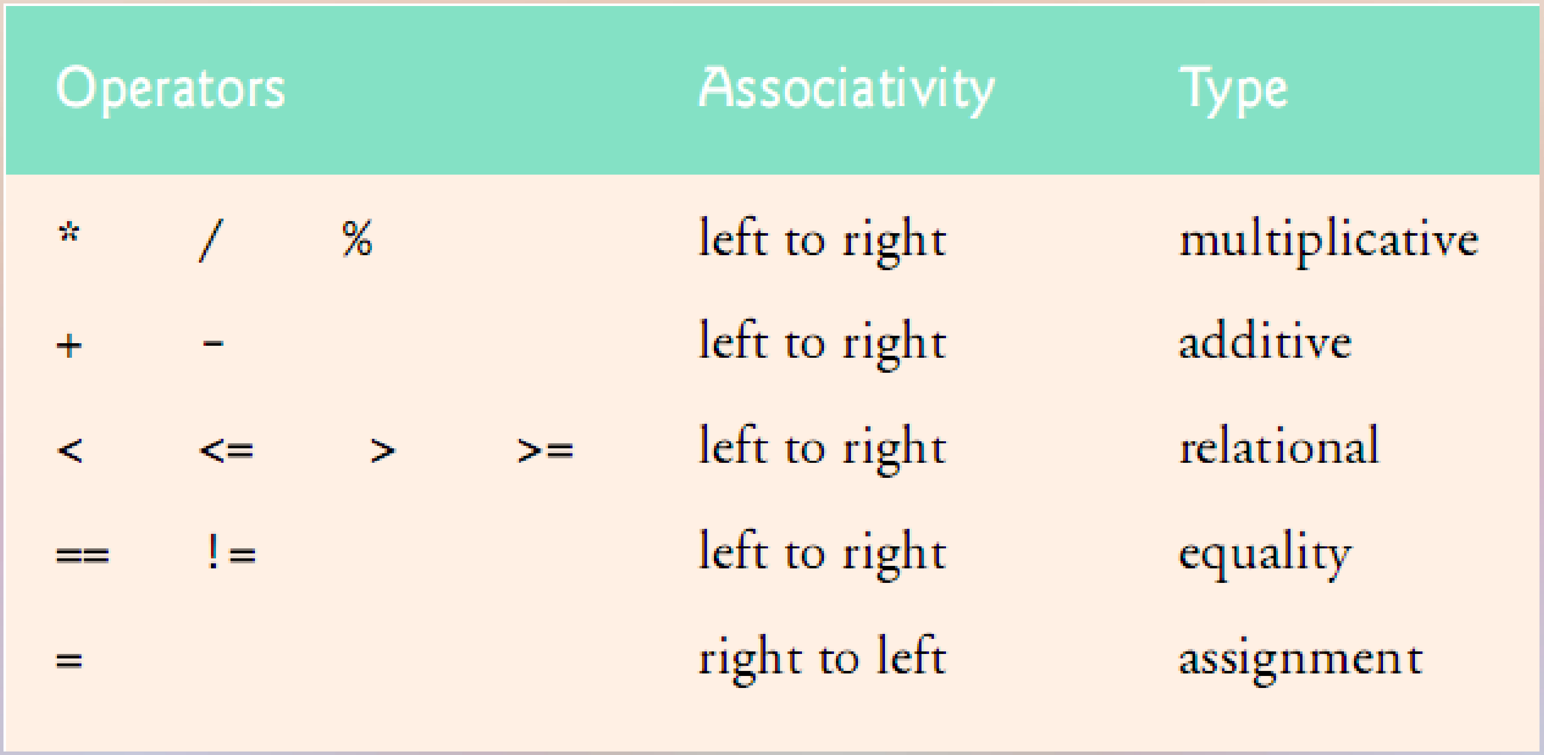


Relational Operators



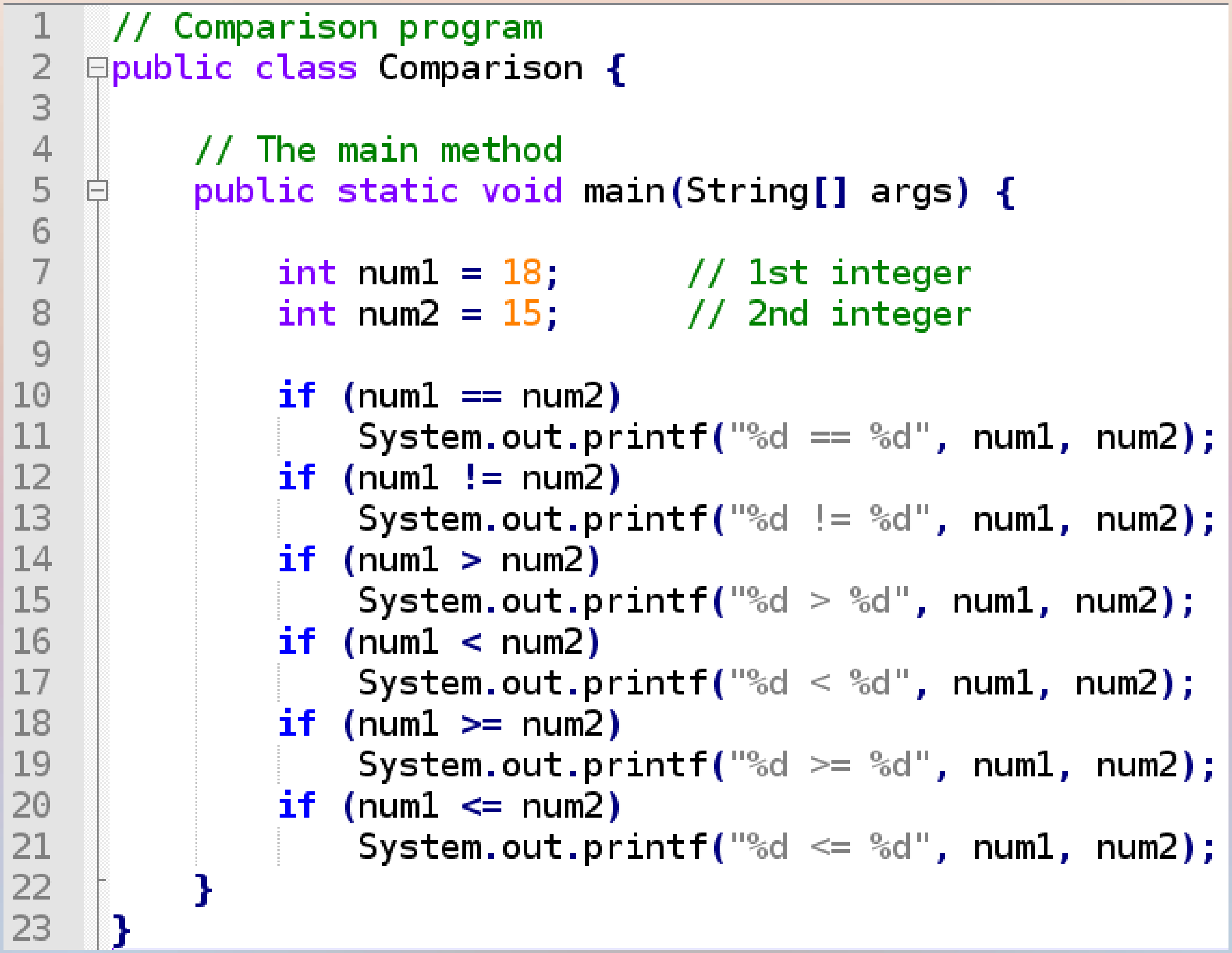
Standard algebraic equality or relational operator	Java equality or relational operator	Sample Java condition	Meaning of Java condition
Equality operators			
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y
Relational operators			
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
≥	>=	x >= y	x is greater than or equal to y
≤	<=	x <= y	x is less than or equal to y

Precedence & Associativity of Operators



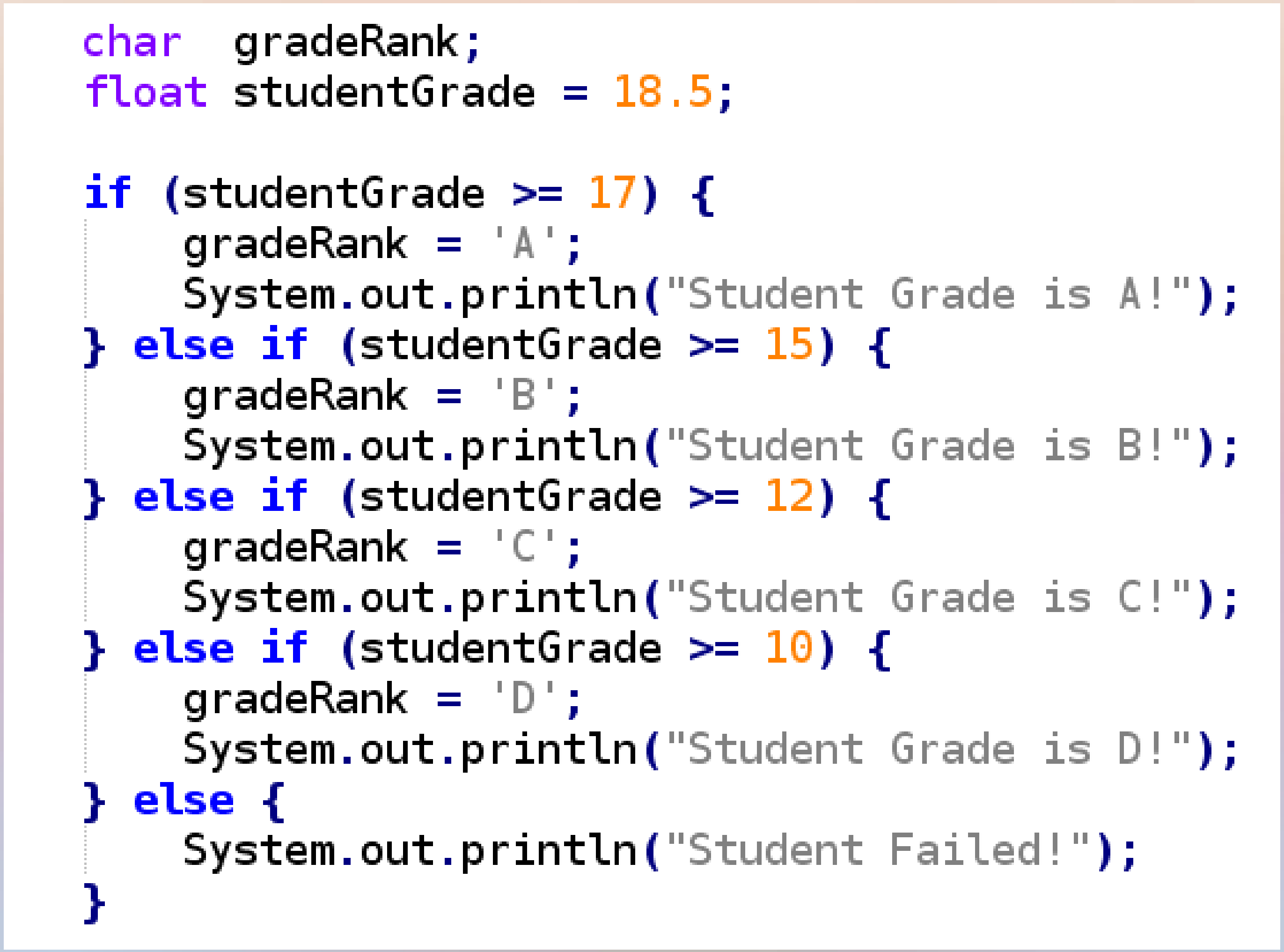
Operators	Associativity	Type
* / %	left to right	multiplicative
+ -	left to right	additive
< <= > >=	left to right	relational
== !=	left to right	equality
=	right to left	assignment

Simple Example Program



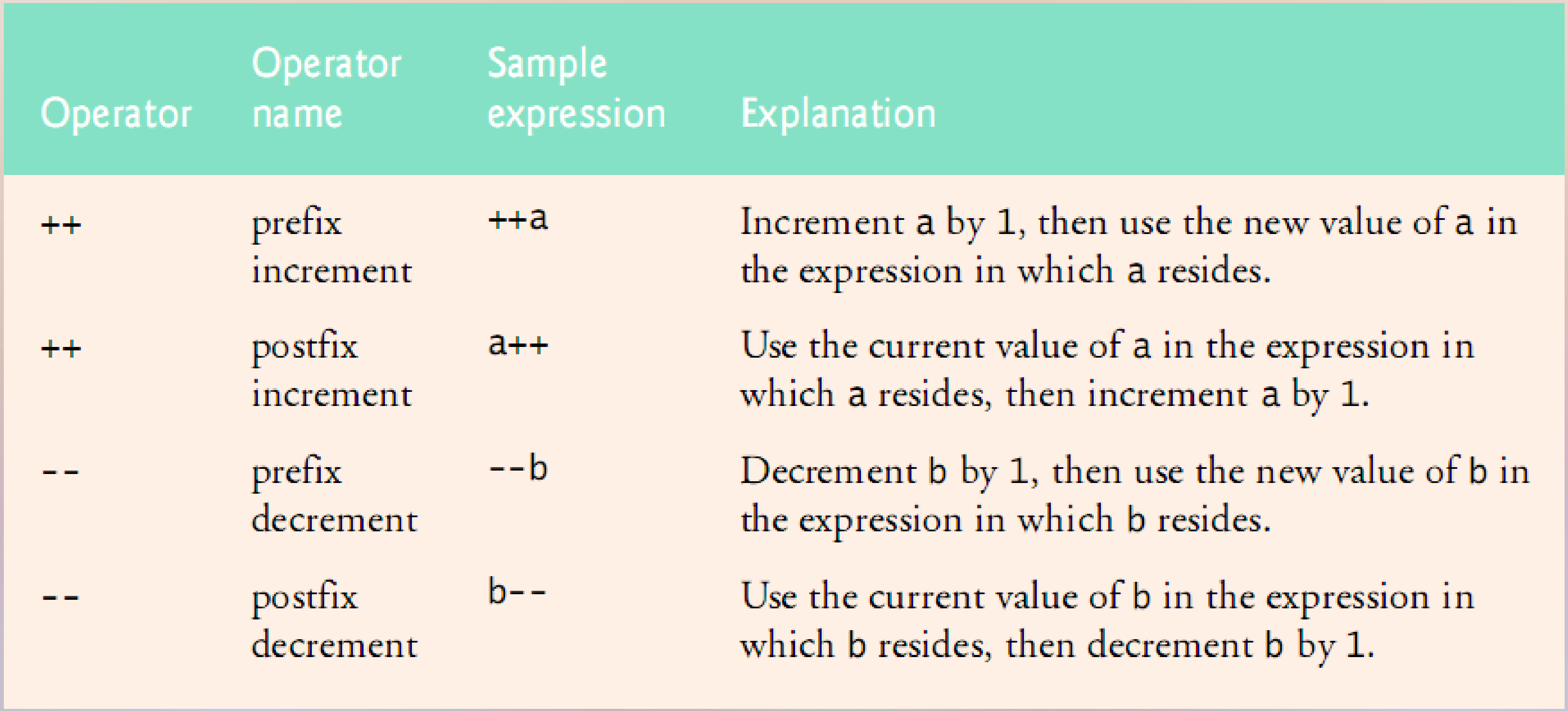
```java
// Comparison program
public class Comparison {

    // The main method
    public static void main(String[] args) {

        int num1 = 18;      // 1st integer
        int num2 = 15;      // 2nd integer

        if (num1 == num2)
            System.out.printf("%d == %d", num1, num2);
        if (num1 != num2)
            System.out.printf("%d != %d", num1, num2);
        if (num1 > num2)
            System.out.printf("%d > %d", num1, num2);
        if (num1 < num2)
            System.out.printf("%d < %d", num1, num2);
        if (num1 >= num2)
            System.out.printf("%d >= %d", num1, num2);
        if (num1 <= num2)
            System.out.printf("%d <= %d", num1, num2);
    }
}
```

if-else Control Statements



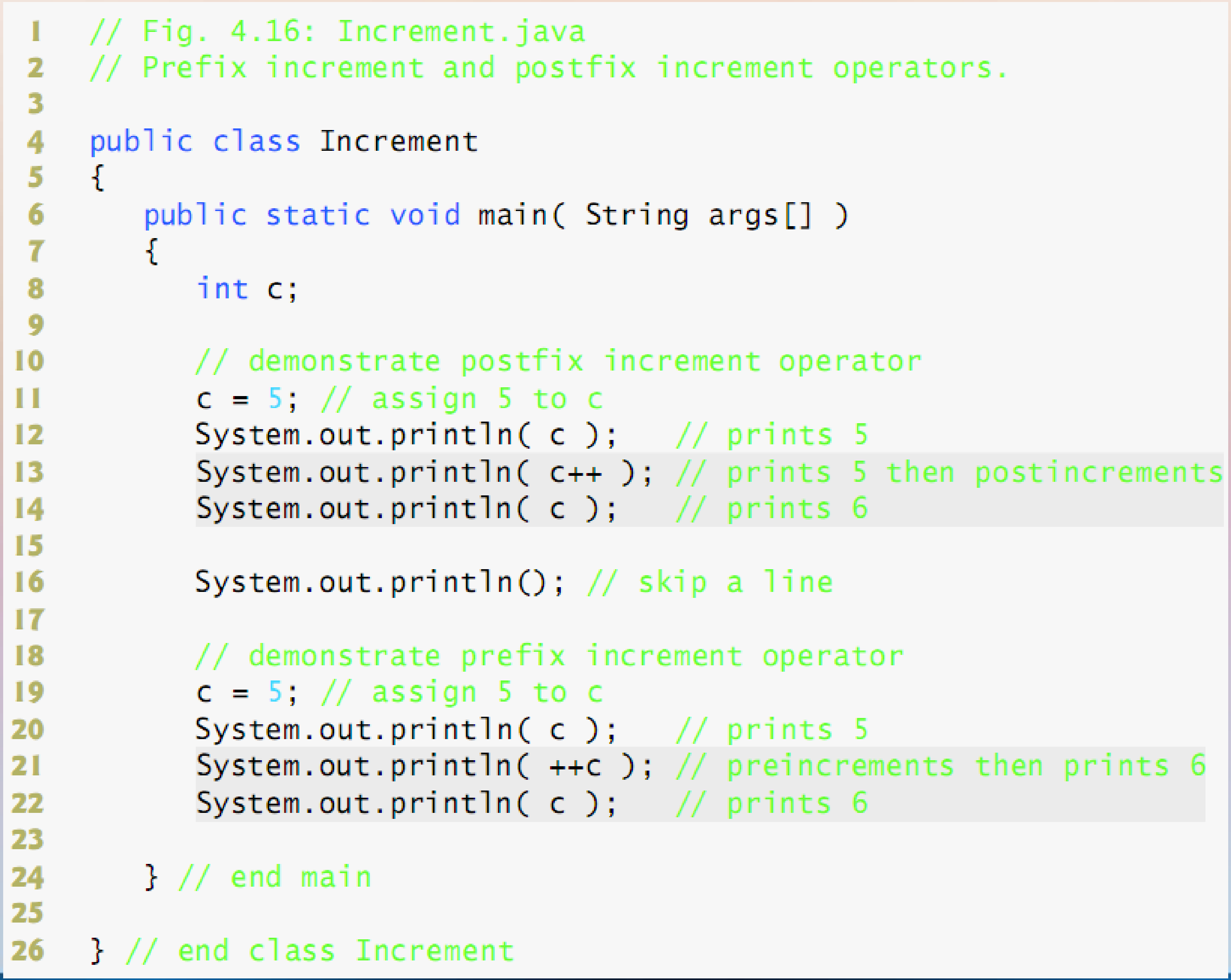
```
char  gradeRank;
float studentGrade = 18.5;

if (studentGrade >= 17) {
    gradeRank = 'A';
    System.out.println("Student Grade is A!");
} else if (studentGrade >= 15) {
    gradeRank = 'B';
    System.out.println("Student Grade is B!");
} else if (studentGrade >= 12) {
    gradeRank = 'C';
    System.out.println("Student Grade is C!");
} else if (studentGrade >= 10) {
    gradeRank = 'D';
    System.out.println("Student Grade is D!");
} else {
    System.out.println("Student Failed!");
}
```

Increment & Decrement Operators



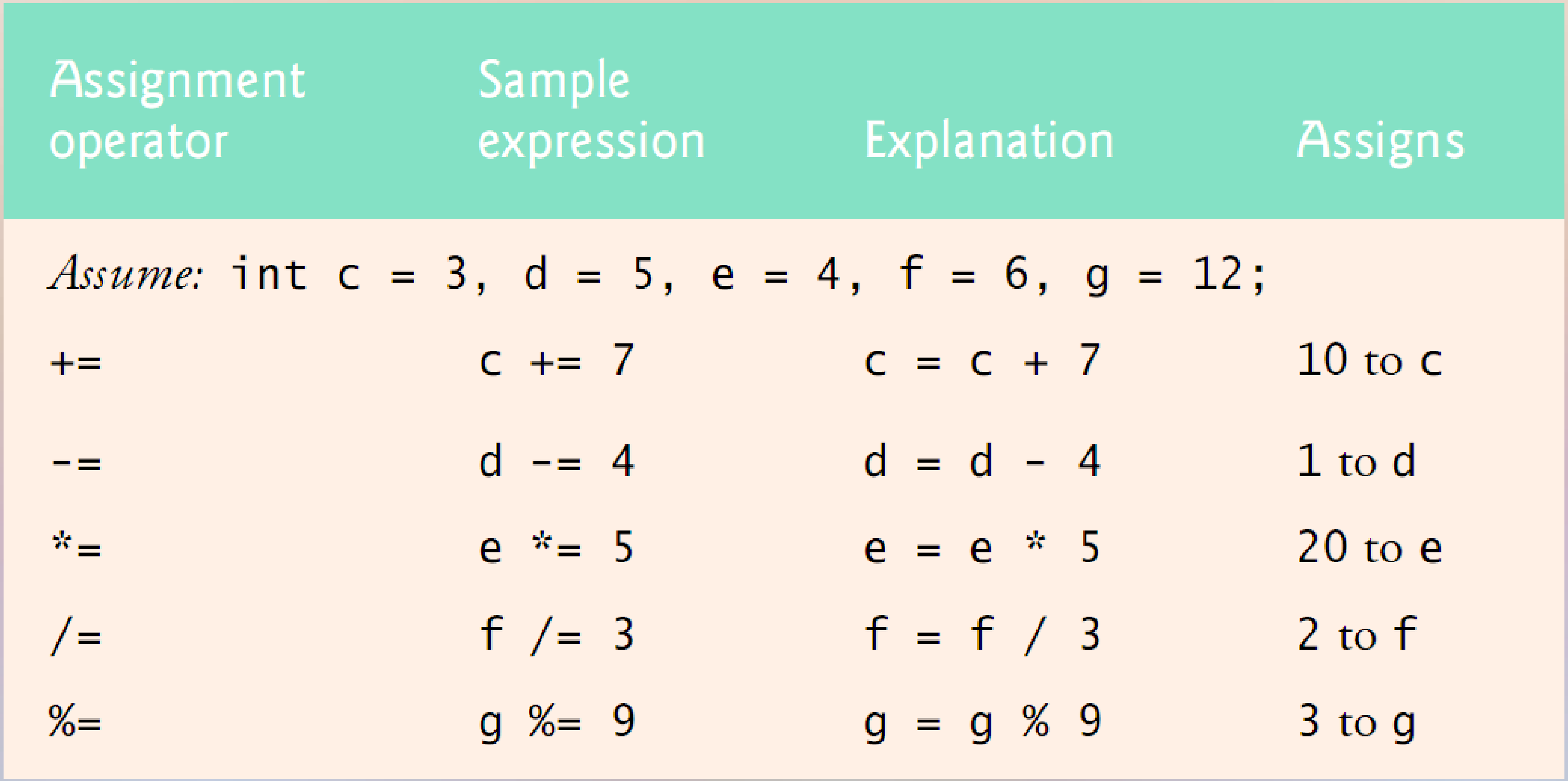
Operator	Operator name	Sample expression	Explanation
`++`	prefix increment	`++a`	Increment `a` by `1`, then use the new value of `a` in the expression in which `a` resides.
`++`	postfix increment	`a++`	Use the current value of `a` in the expression in which `a` resides, then increment `a` by `1`.
`--`	prefix decrement	`--b`	Decrement `b` by `1`, then use the new value of `b` in the expression in which `b` resides.
`--`	postfix decrement	`b--`	Use the current value of `b` in the expression in which `b` resides, then decrement `b` by `1`.

The Difference …



```java
// Fig. 4.16: Increment.java
// Prefix increment and postfix increment operators.

public class Increment
{
   public static void main( String args[] )
   {
      int c;

      // demonstrate postfix increment operator
      c = 5; // assign 5 to c
      System.out.println( c );   // prints 5
      System.out.println( c++ ); // prints 5 then postincrements
      System.out.println( c );   // prints 6

      System.out.println(); // skip a line

      // demonstrate prefix increment operator
      c = 5; // assign 5 to c
      System.out.println( c );   // prints 5
      System.out.println( ++c ); // preincrements then prints 6
      System.out.println( c );   // prints 6

   } // end main

} // end class Increment
```

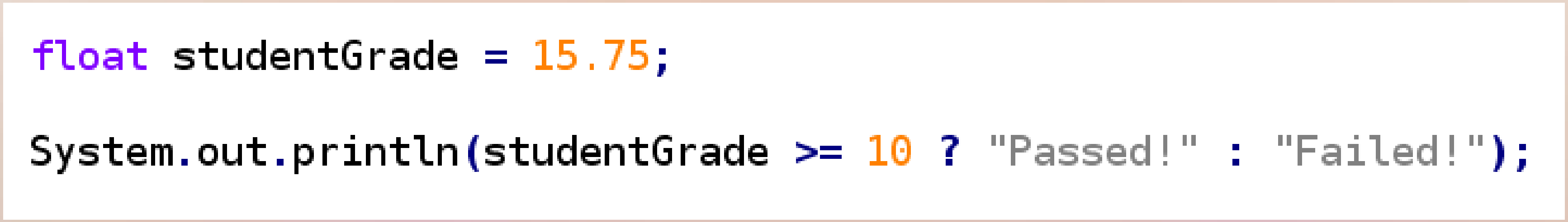
Arithmatic Compound Assignment Operators



Assignment operator	Sample expression	Explanation	Assigns
Assume: `int c = 3, d = 5, e = 4, f = 6, g = 12;`			
`+=`	`c += 7`	`c = c + 7`	10 to c
`-=`	`d -= 4`	`d = d - 4`	1 to d
`*=`	`e *= 5`	`e = e * 5`	20 to e
`/=`	`f /= 3`	`f = f / 3`	2 to f
`%=`	`g %= 9`	`g = g % 9`	3 to g

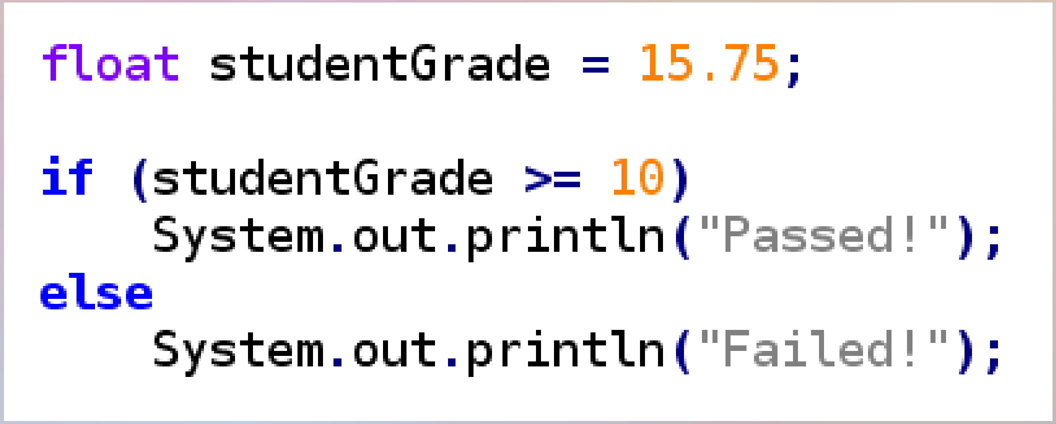
The Conditinal Operator

- **The Conditinal Operator (?:)**



```java
float studentGrade = 15.75;

System.out.println(studentGrade >= 10 ? "Passed!" : "Failed!");
```

- **is equal to …**



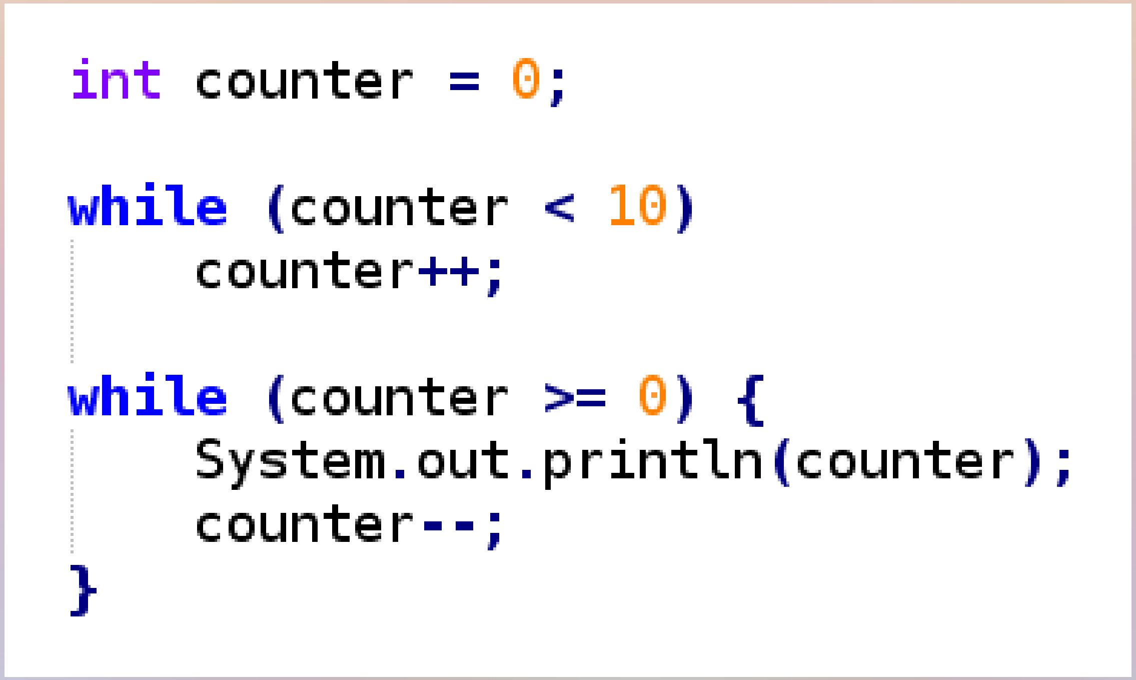
```java
float studentGrade = 15.75;

if (studentGrade >= 10)
    System.out.println("Passed!");
else
    System.out.println("Failed!");
```

Precedence & Associativity of Operators

Operators	Associativity	Type
`++` `--`	right to left	unary postfix
`++` `--` `+` `-` `(` *type* `)`	right to left	unary prefix
`*` `/` `%`	left to right	multiplicative
`+` `-`	left to right	additive
`<` `<=` `>` `>=`	left to right	relational
`==` `!=`	left to right	equality
`?:`	right to left	conditional
`=` `+=` `-=` `*=` `/=` `%=`	right to left	assignment

Repetition Control Statements

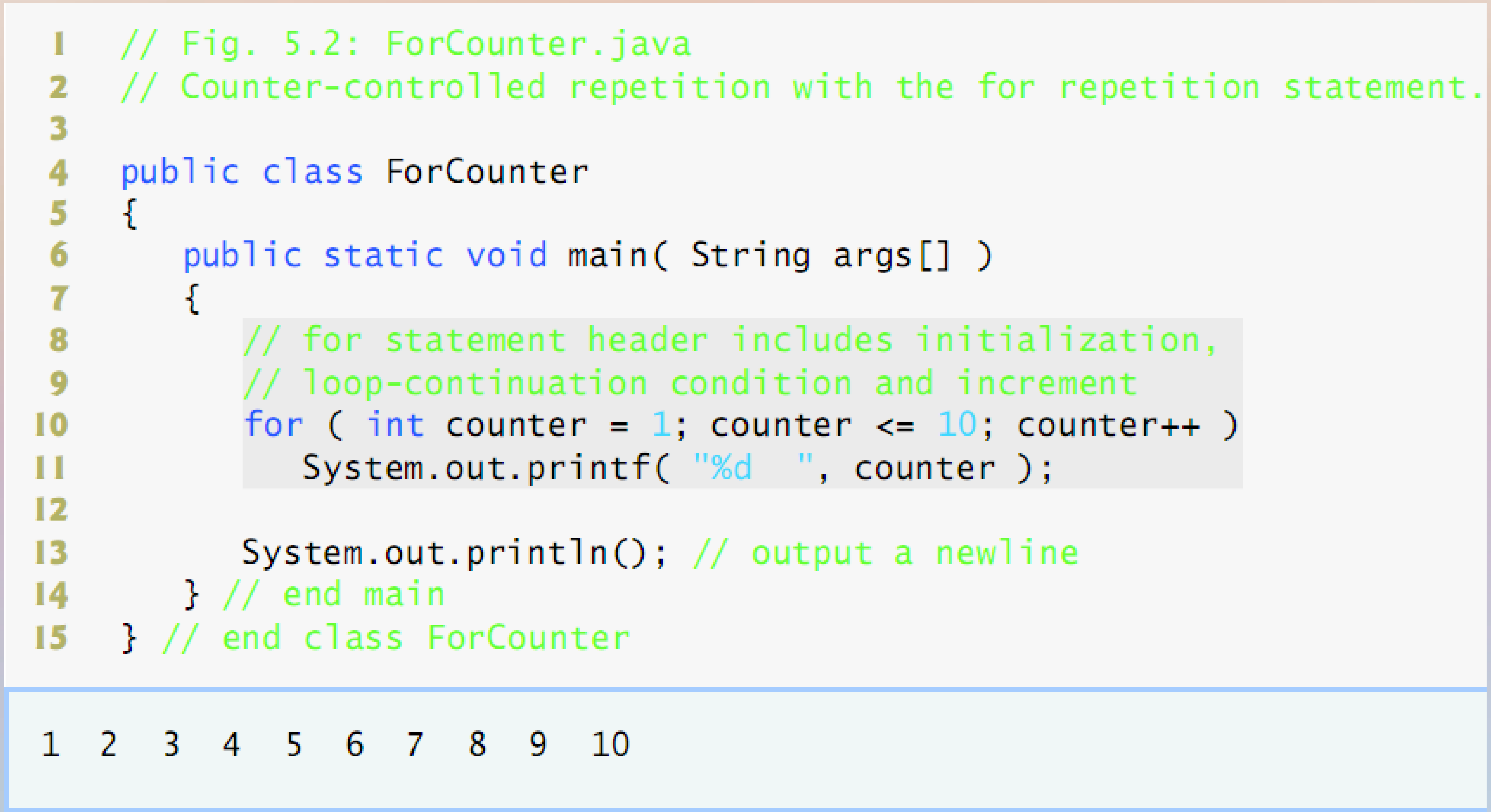
- while **Repetition Statement**



```
int counter = 0;

while (counter < 10)
    counter++;

while (counter >= 0) {
    System.out.println(counter);
    counter--;
}
```

Repetition Control Statements (continued ...)

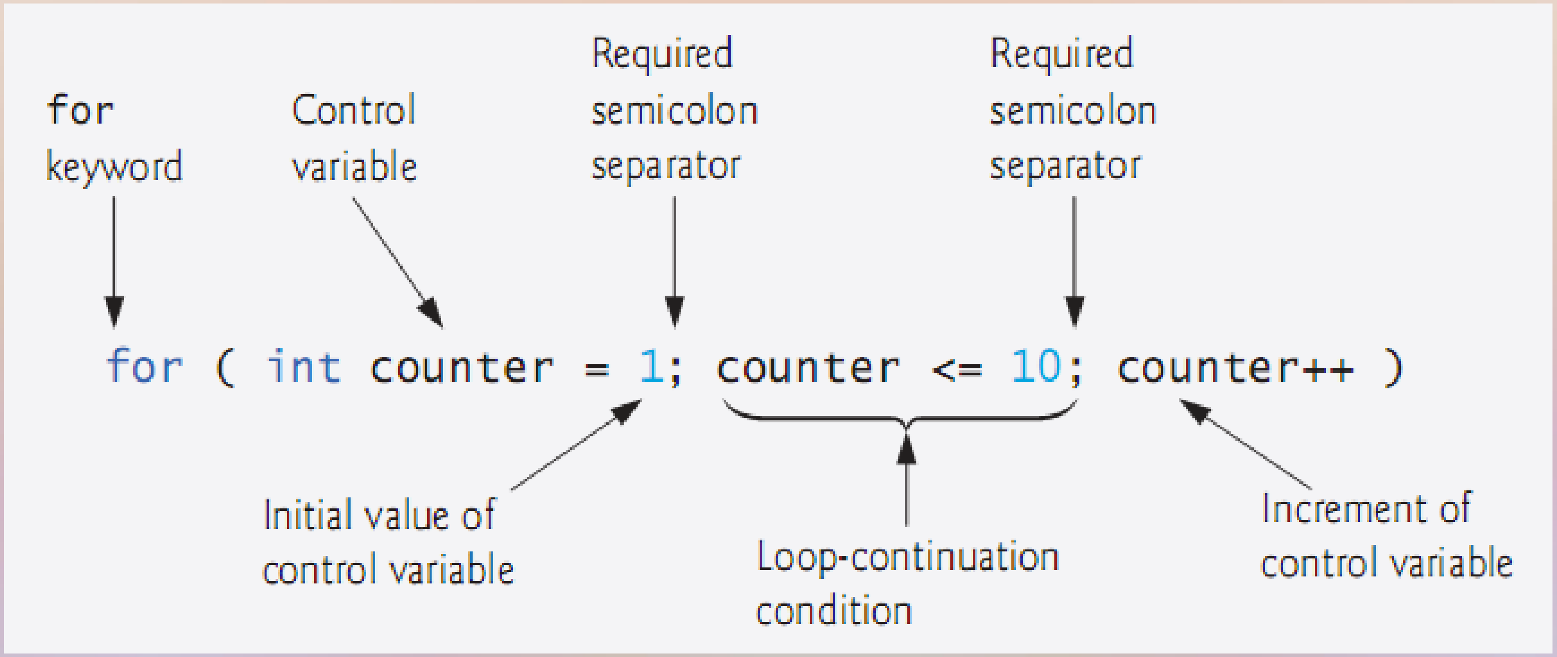
- for **Repetition Statement**



```java
// Fig. 5.2: ForCounter.java
// Counter-controlled repetition with the for repetition statement.

public class ForCounter
{
   public static void main( String args[] )
   {
      // for statement header includes initialization,
      // loop-continuation condition and increment
      for ( int counter = 1; counter <= 10; counter++ )
         System.out.printf( "%d  ", counter );

      System.out.println(); // output a newline
   } // end main
} // end class ForCounter
```

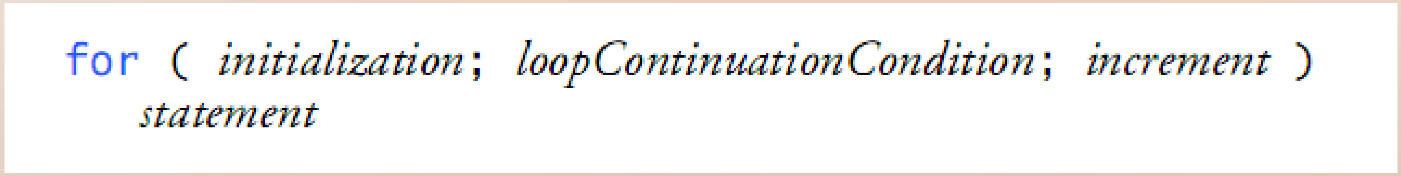
```
1  2  3  4  5  6  7  8  9  10
```

Repetition Control Statements (continued ...)


for keyword
Control variable
Required semicolon separator
Required semicolon separator
Initial value of control variable
Loop-continuation condition
Increment of control variable

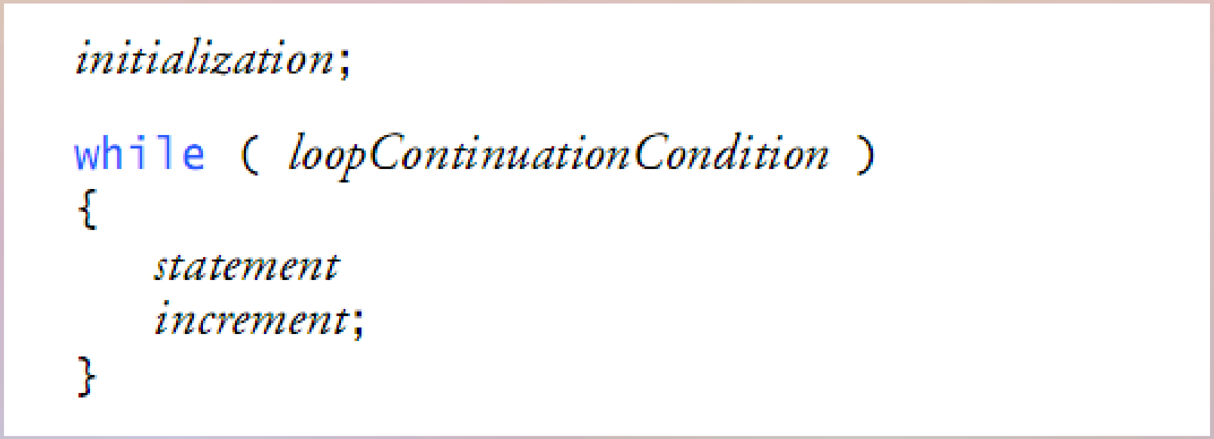
```
for ( int counter = 1; counter <= 10; counter++ )
```

Repetition Control Statements (continued ...)



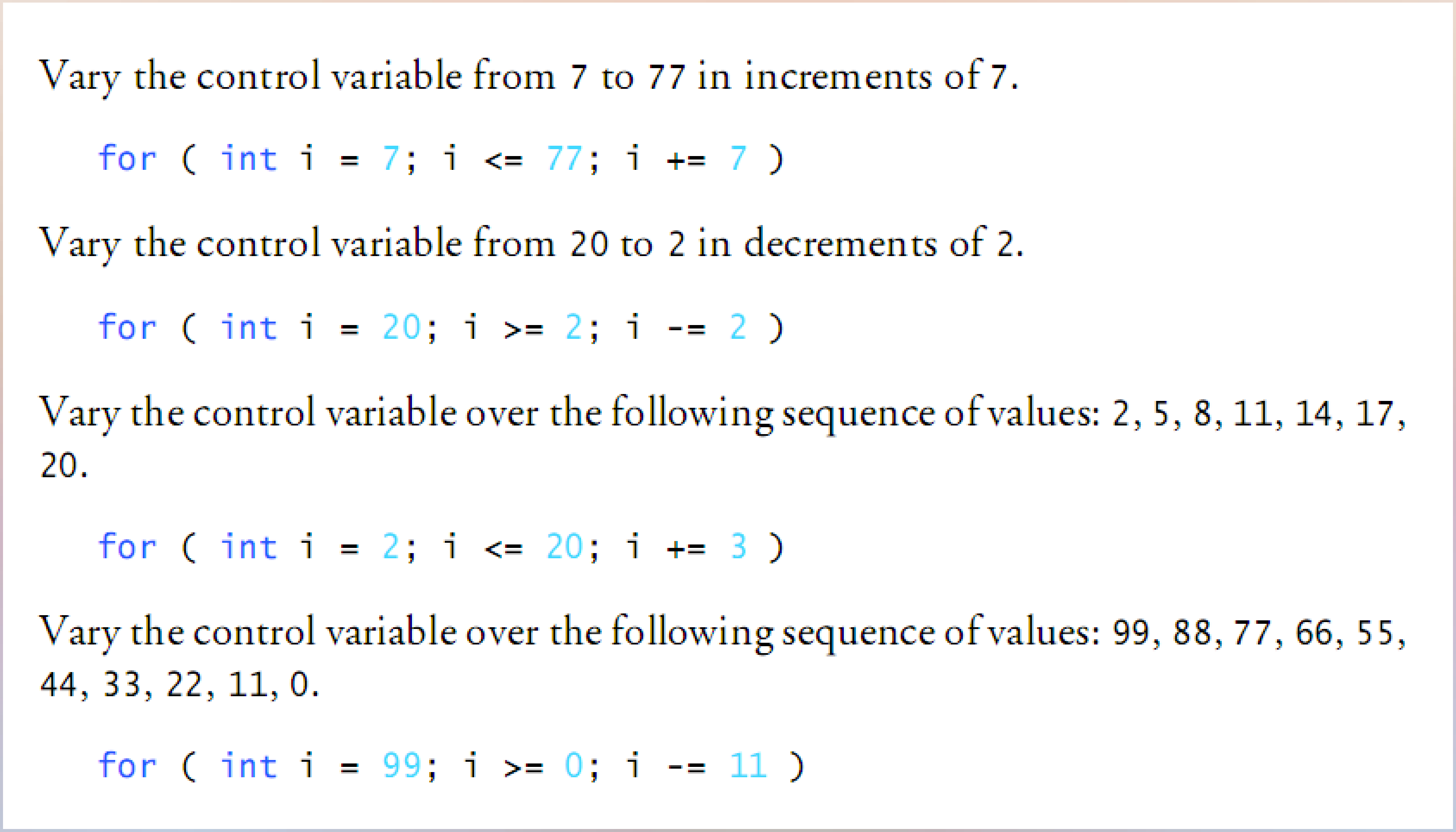
```
for ( initialization; loopContinuationCondition; increment )
   statement
```

is equal to …



```
initialization;

while ( loopContinuationCondition )
{
   statement
   increment;
}
```

Repetition Control Statements (continued ...)



Vary the control variable from 7 to 77 in increments of 7.

```
for ( int i = 7; i <= 77; i += 7 )
```

Vary the control variable from 20 to 2 in decrements of 2.

```
for ( int i = 20; i >= 2; i -= 2 )
```

Vary the control variable over the following sequence of values: 2, 5, 8, 11, 14, 17, 20.

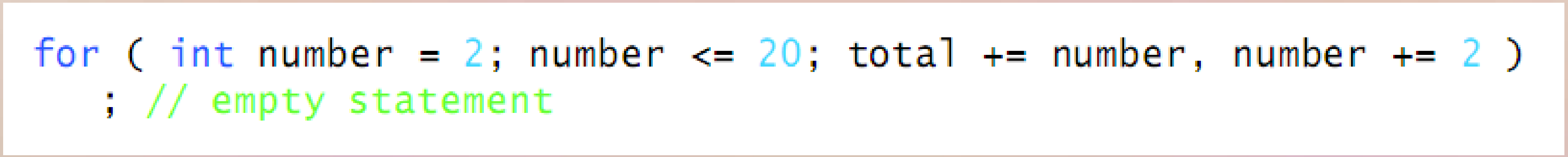
```
for ( int i = 2; i <= 20; i += 3 )
```

Vary the control variable over the following sequence of values: 99, 88, 77, 66, 55, 44, 33, 22, 11, 0.

```
for ( int i = 99; i >= 0; i -= 11 )
```

Repetition Control Statements (continued ...)

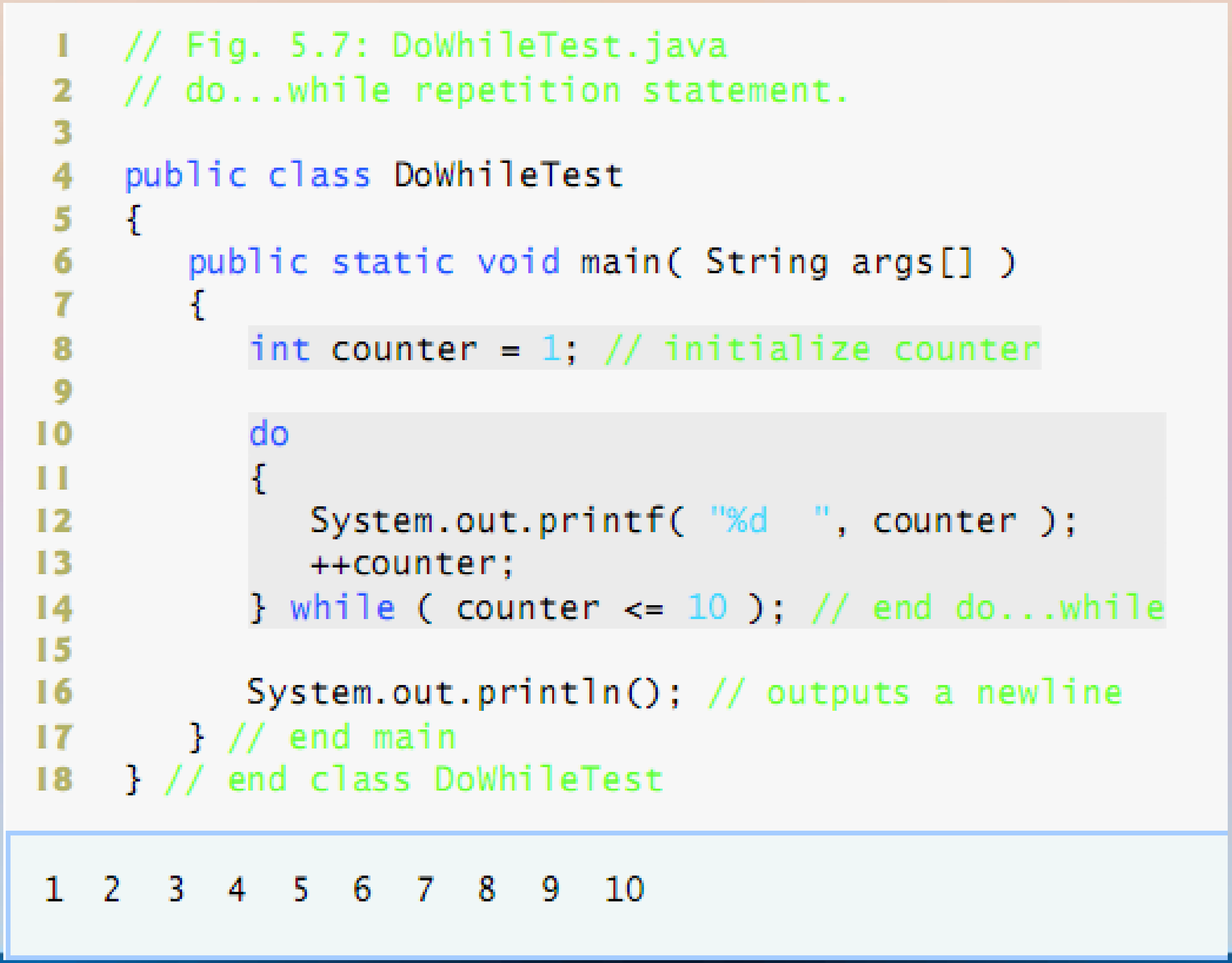
- **Summation of even numbers in the range of 2 to 20:**



```
for ( int number = 2; number <= 20; total += number, number += 2 )
   ; // empty statement
```

Repetition Control Statements (continued ...)

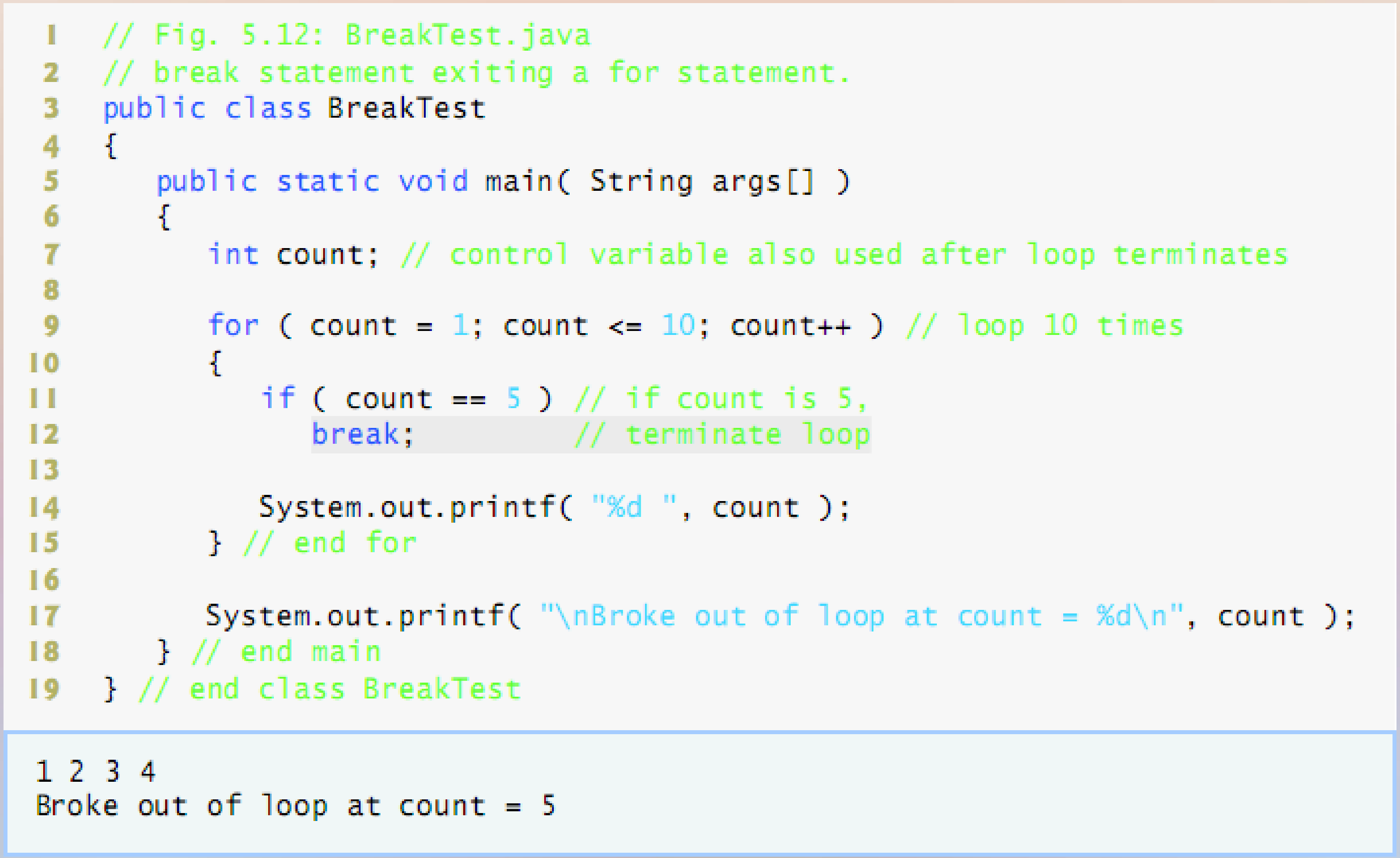
- do...while **Repetition Statement**



```java
// Fig. 5.7: DoWhileTest.java
// do...while repetition statement.

public class DoWhileTest
{
   public static void main( String args[] )
   {
      int counter = 1; // initialize counter

      do
      {
         System.out.printf( "%d  ", counter );
         ++counter;
      } while ( counter <= 10 ); // end do...while

      System.out.println(); // outputs a newline
   } // end main
} // end class DoWhileTest
```

```
1  2  3  4  5  6  7  8  9  10
```

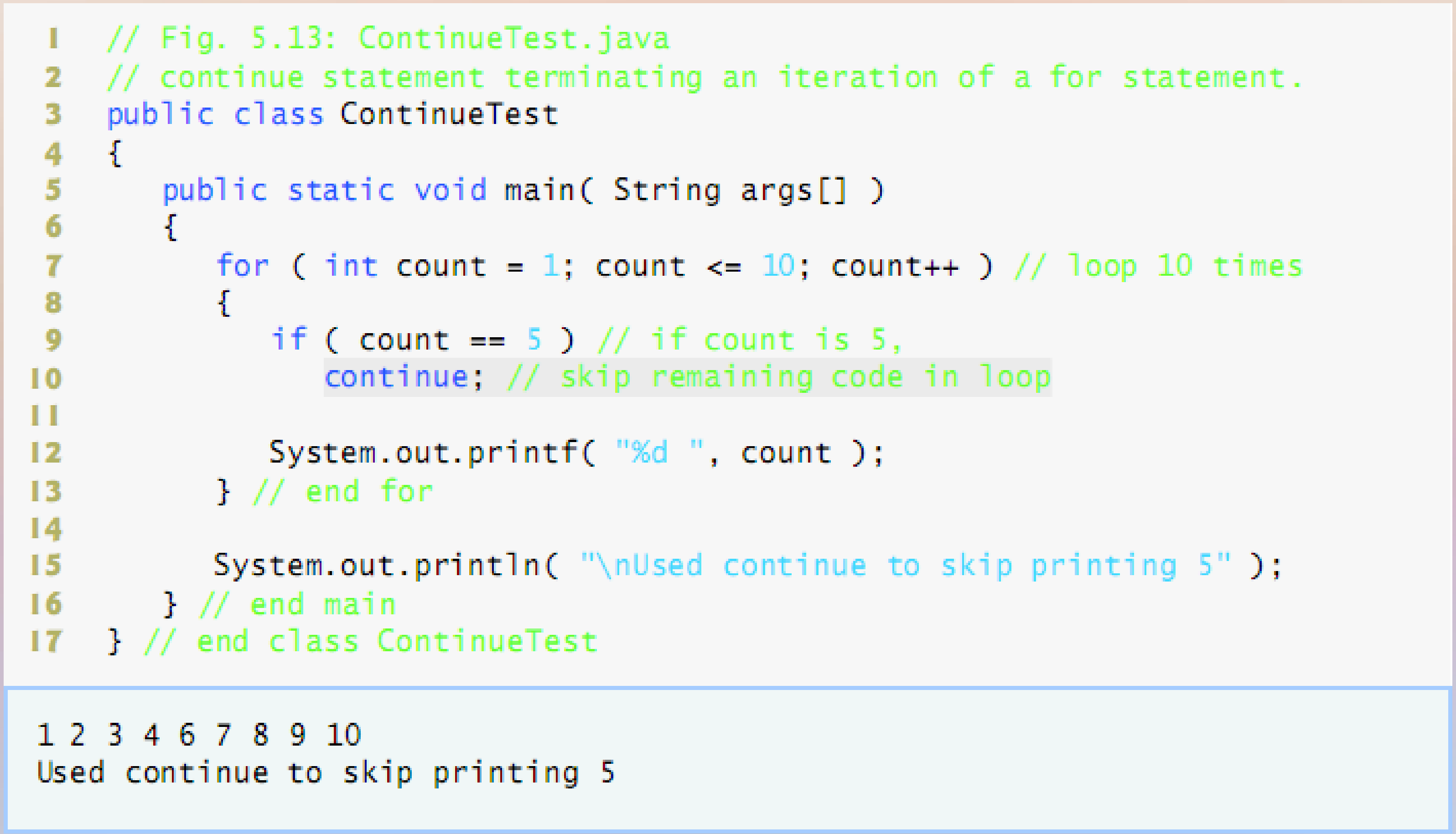
break Statement



```java
// Fig. 5.12: BreakTest.java
// break statement exiting a for statement.
public class BreakTest
{
   public static void main( String args[] )
   {
      int count; // control variable also used after loop terminates

      for ( count = 1; count <= 10; count++ ) // loop 10 times
      {
         if ( count == 5 ) // if count is 5,
            break;         // terminate loop

         System.out.printf( "%d ", count );
      } // end for

      System.out.printf( "\nBroke out of loop at count = %d\n", count );
   } // end main
} // end class BreakTest
```

```
1 2 3 4
Broke out of loop at count = 5
```

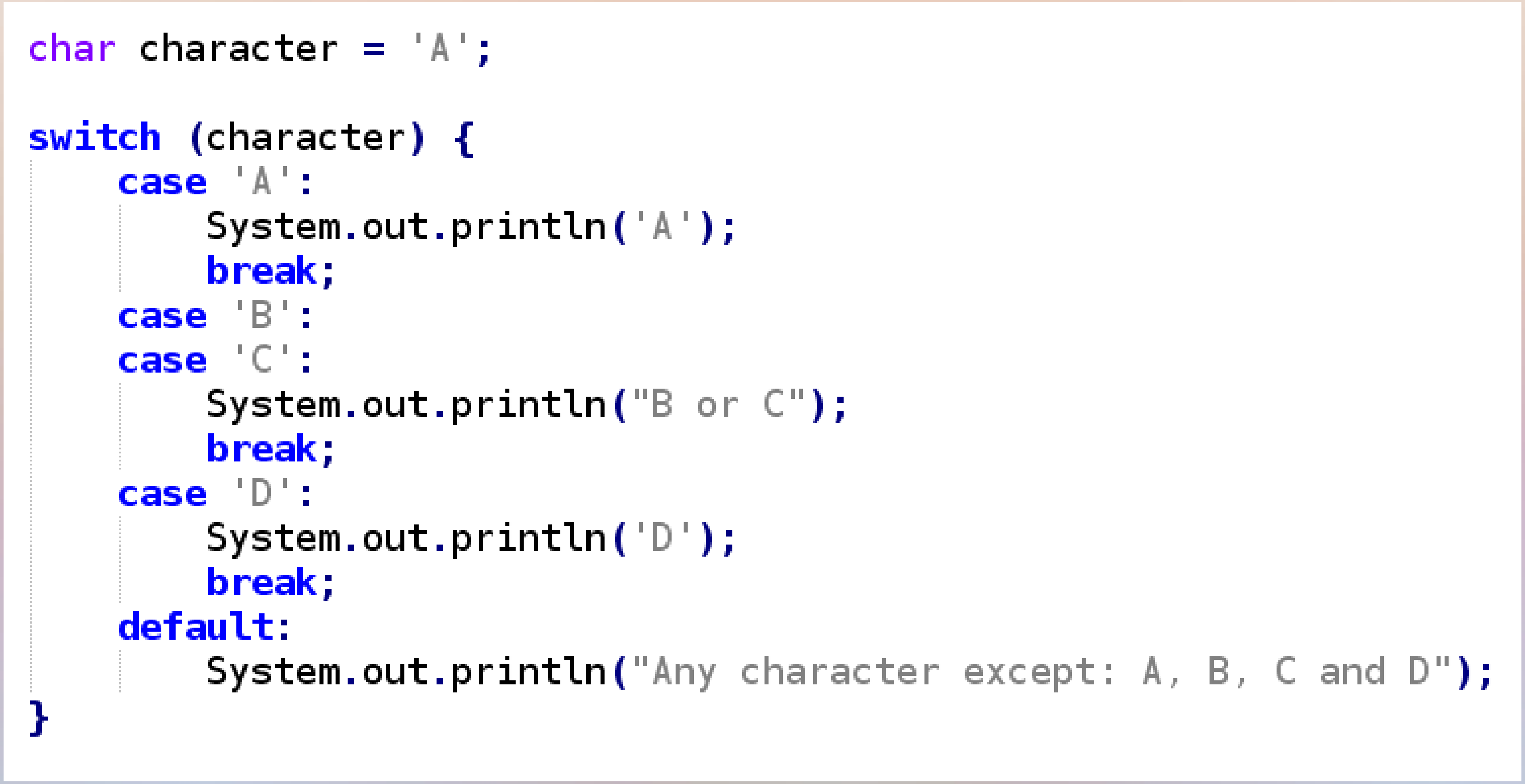
continue Statement



```java
// Fig. 5.13: ContinueTest.java
// continue statement terminating an iteration of a for statement.
public class ContinueTest
{
   public static void main( String args[] )
   {
      for ( int count = 1; count <= 10; count++ ) // loop 10 times
      {
         if ( count == 5 ) // if count is 5,
            continue; // skip remaining code in loop

         System.out.printf( "%d ", count );
      } // end for

      System.out.println( "\nUsed continue to skip printing 5" );
   } // end main
} // end class ContinueTest
```

```
1 2 3 4 6 7 8 9 10
Used continue to skip printing 5
```

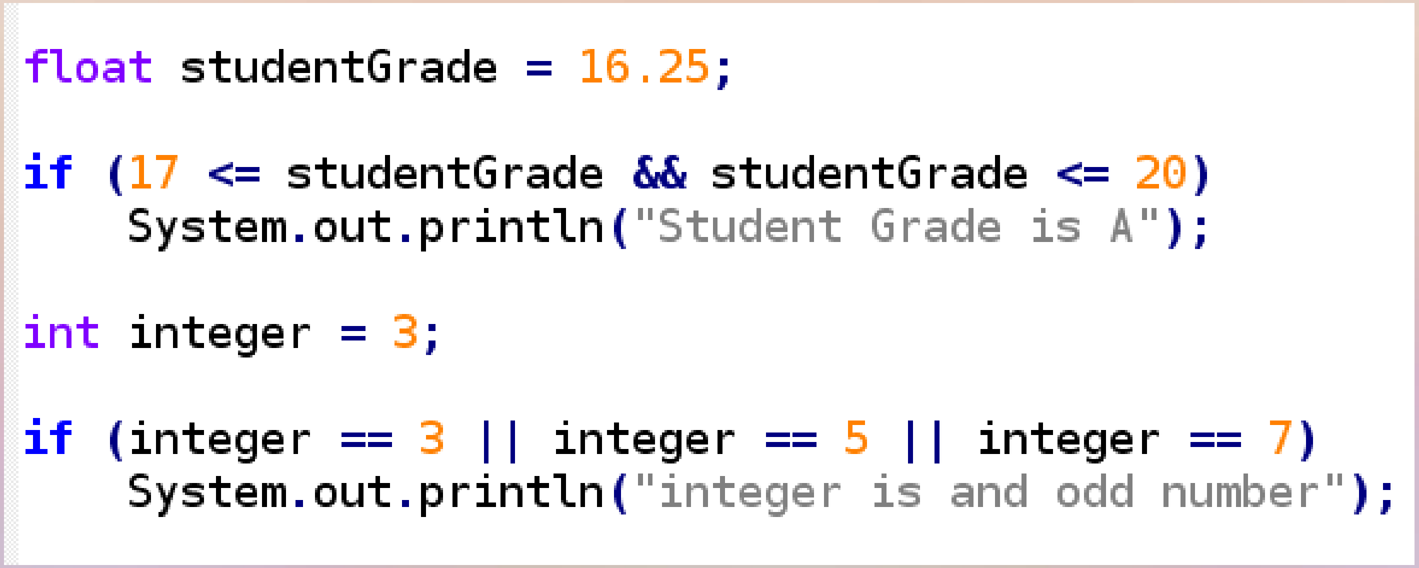
switch Multiple-Selection Statement



```java
char character = 'A';

switch (character) {
    case 'A':
        System.out.println('A');
        break;
    case 'B':
    case 'C':
        System.out.println("B or C");
        break;
    case 'D':
        System.out.println('D');
        break;
    default:
        System.out.println("Any character except: A, B, C and D");
}
```

Logical Operators

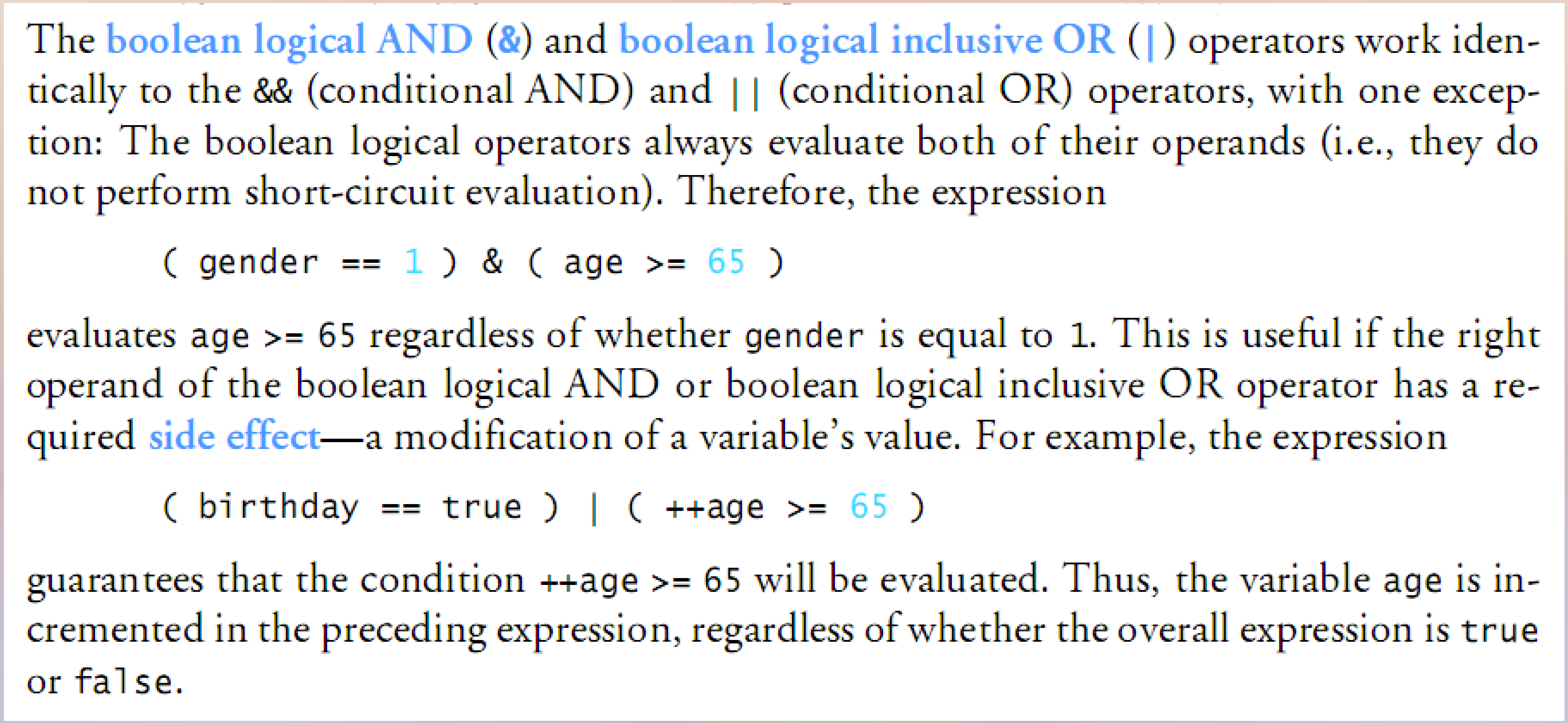
- **Conditional AND and OR operators**



```
float studentGrade = 16.25;

if (17 <= studentGrade && studentGrade <= 20)
    System.out.println("Student Grade is A");

int integer = 3;

if (integer == 3 || integer == 5 || integer == 7)
    System.out.println("integer is and odd number");
```

Logical Operators (continued ...)

- **Logical AND and OR operators**



The boolean logical AND (`&`) and boolean logical inclusive OR (`|`) operators work identically to the `&&` (conditional AND) and `||` (conditional OR) operators, with one exception: The boolean logical operators always evaluate both of their operands (i.e., they do not perform short-circuit evaluation). Therefore, the expression

```
( gender == 1 ) & ( age >= 65 )
```

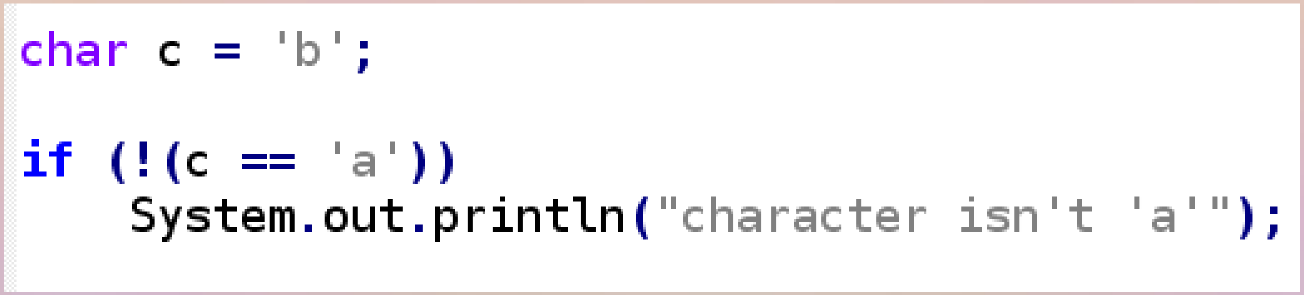
evaluates `age >= 65` regardless of whether `gender` is equal to `1`. This is useful if the right operand of the boolean logical AND or boolean logical inclusive OR operator has a required side effect—a modification of a variable's value. For example, the expression

```
( birthday == true ) | ( ++age >= 65 )
```

guarantees that the condition `++age >= 65` will be evaluated. Thus, the variable `age` is incremented in the preceding expression, regardless of whether the overall expression is `true` or `false`.

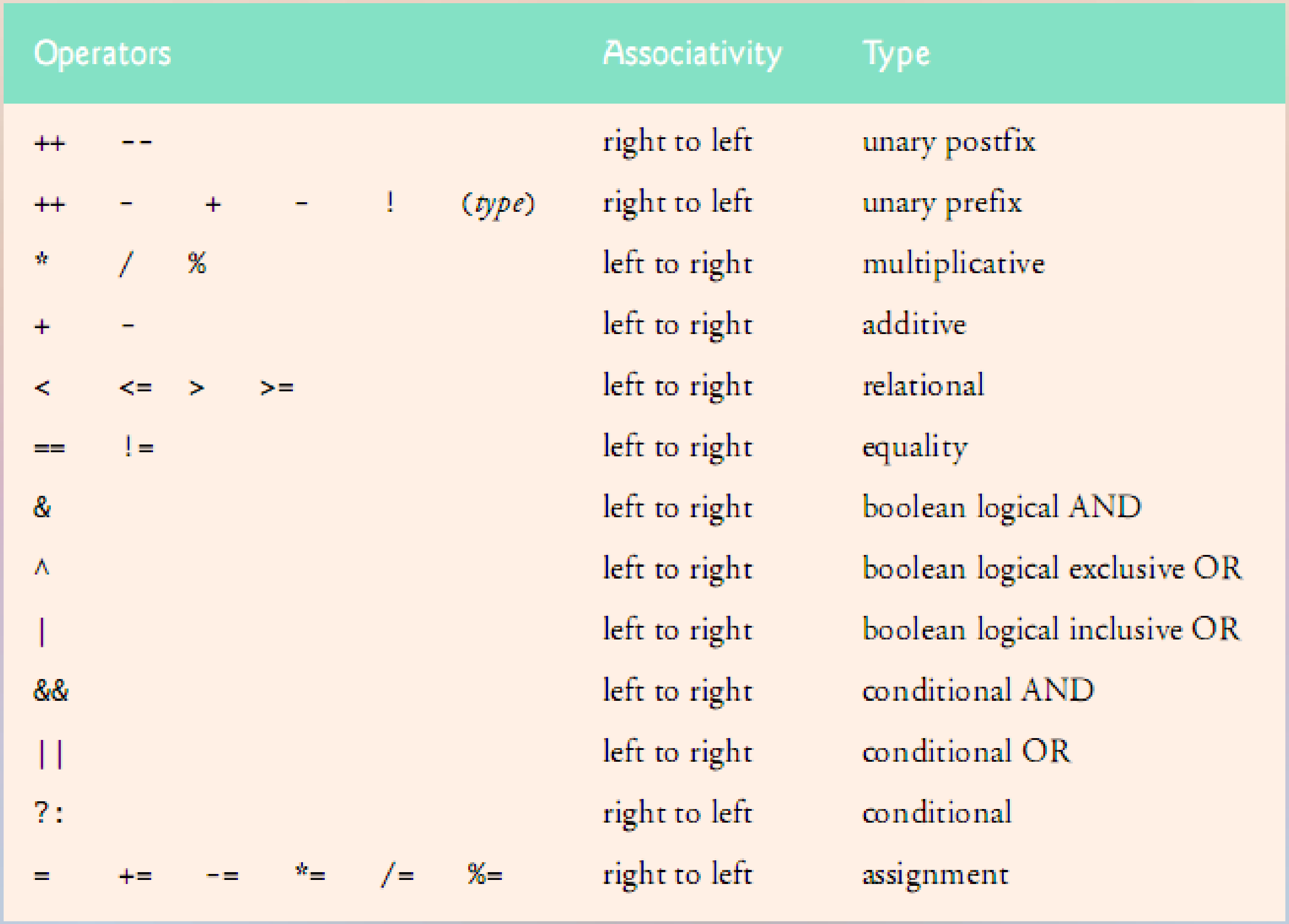
Logical Operators (continued ...)

- **Logical Negation Operator**



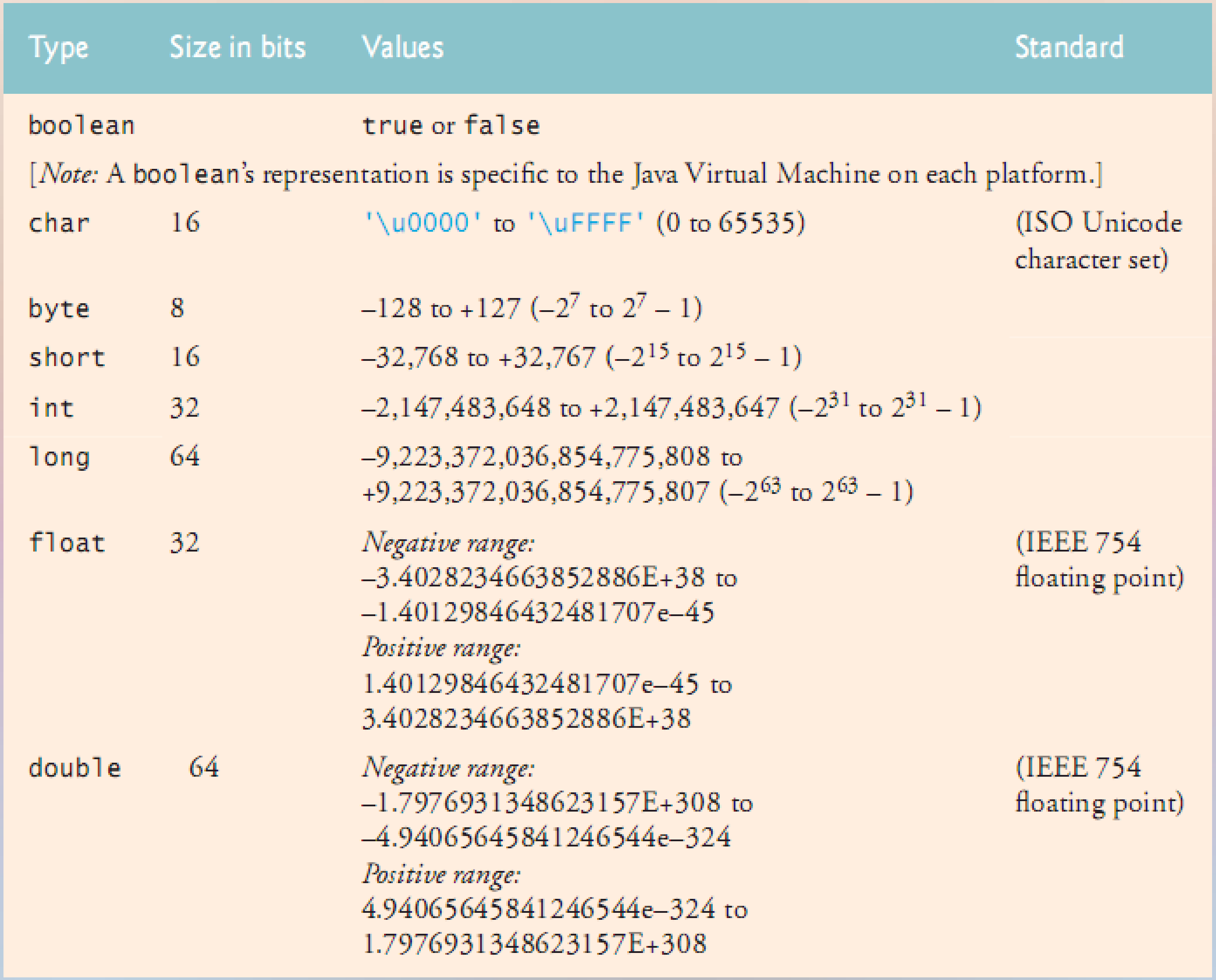
```
char c = 'b';

if (!(c == 'a'))
    System.out.println("character isn't 'a'");
```

Precedence & Associativity of Operators



Operators	Associativity	Type
`++ --`	right to left	unary postfix
`++ - + - !` *(type)*	right to left	unary prefix
`* / %`	left to right	multiplicative
`+ -`	left to right	additive
`< <= > >=`	left to right	relational
`== !=`	left to right	equality
`&`	left to right	boolean logical AND
`^`	left to right	boolean logical exclusive OR
`|`	left to right	boolean logical inclusive OR
`&&`	left to right	conditional AND
`||`	left to right	conditional OR
`?:`	right to left	conditional
`= += -= *= /= %=`	right to left	assignment

Primitive Data-Types



Type	Size in bits	Values	Standard
`boolean`		`true` or `false`	
[*Note:* A `boolean`'s representation is specific to the Java Virtual Machine on each platform.]			
`char`	16	`'\u0000'` to `'\uFFFF'` (0 to 65535)	(ISO Unicode character set)
`byte`	8	–128 to +127 (-2^7 to $2^7 - 1$)	
`short`	16	–32,768 to +32,767 (-2^{15} to $2^{15} - 1$)	
`int`	32	–2,147,483,648 to +2,147,483,647 (-2^{31} to $2^{31} - 1$)	
`long`	64	–9,223,372,036,854,775,808 to +9,223,372,036,854,775,807 (-2^{63} to $2^{63} - 1$)	
`float`	32	*Negative range:* –3.4028234663852886E+38 to –1.40129846432481707e–45 *Positive range:* 1.40129846432481707e–45 to 3.4028234663852886E+38	(IEEE 754 floating point)
`double`	64	*Negative range:* –1.7976931348623157E+308 to –4.94065645841246544e–324 *Positive range:* 4.94065645841246544e–324 to 1.7976931348623157E+308	(IEEE 754 floating point)

Code Aesthetics

- Indent the code inside a block   (4x spaces or 1x tab)
- Put a space on both sides of every operator
- Start the name of every variable with lower-case letters
- Start the name of every class with upper-case letters
- Use Camel-case letters for all names

Java Coding Conventions

- **Sun MicroSystems original Java coding conventions:**
- www.oracle.com/technetwork/java/codeconventions-150003.pdf

- **Google's Java coding conventions:**
- https://google.github.io/styleguide/javaguide.html

- **Twitter's Java coding conventions:**
- github.com/twitter/commons/blob/master/src/java/com/twitter/common/styleguide.md

References

- **Deitel's Java How to Program (7th Edition)**
  - Chapter 2
  - Chapter 4
  - Chapter 5

شعر امروز

نادان دلش خوش است به تدبیر ناخدا

غافل که ناخدا هم ازین تخته پاره‌هاست