Elementary Java Programming

CSGE601021 Dasar-Dasar Pemrograman 2

Fakultas Ilmu Komputer Universitas Indonesia

Menu

- Basics
- Selections
- Loops
- Math, Character, Strings

```
import passport from 'passport';
import LocalStrategy from 'passport-local';
import { Strategy as JWTStrategy, ExtractJwt } from 'passport-jwt';
import request from 'superagent';

import User from '../models/user.model';
import constants from '../config/constants';
import { createUser } from '../helpers/auth.helper';

/**
 * Local Strategy Auth
 */
const localOpts = { usernameField: 'username' };

const localLogin = new LocalStrategy(
  localOpts,
  async (username, password, done) => {
    try {
      const user = await User.query().where('username', username);

      if (user.length === 0) {
        const userData = {
          username,
          password,
        };
        const createdUser = await createUser(userData);
        return done(null, createdUser);
      } else if (!user[0].authenticateUser(password)) {
        return done(null, false);
      }
      return done(null, user[0]);
    } catch (e) {
      return done(null, false);
    }
  },
);

/**
 * JWT Strategy Auth
 */
const jwtOpts = {
  // Telling Passport to check authorization headers for JWT
  jwtFromRequest: ExtractJwt.fromAuthHeaderWithScheme('JWT'),
  // Telling Passport where to find the secret
  secretOrKey: constants.JWT_SECRET,
};

const jwtLogin = new JWTStrategy(jwtOpts, async (payload, done) => {
  try {
    console.log(payload);
    const user = await User.query().where('user_uuid', payload.user_uuid);
    console.log(user[0].toJSON());

    if (user.length === 0 || !user) {
      return done(null, false);
    }

    return done(null, user[0]);
  } catch (e) {
    console.log(e);
    return done(e, false);
 0  - 1.8k auth.js  Javascript-IDE  ⓪①②③④  Git-feature/fullSchedule
                                                                 unix | 1: 0  Top
```

The Basics

Basics

- Identifiers
- Variable declaration and initialization
- Numeric literal
- Constant variables
- Naming conventions
- Input from the console using **Scanner**
- Numeric operators
- Precedence of operators
- Augmented assignment operators
- Increment and Decrement Operators
- Cast the value of one type to another type

Python vs Java

- Python

```
someVariable = 42
someVariable = 'Hello, world'
```

$\underbrace{\texttt{someVariable}}_{}$

This "name" will 'identify' the variable

- Java

```
int someVariable = 42;
String someVariable2 = "hello";
```

Identifiers

- Identifies variables, methods, classes.
- sequence of characters that consist of letters, digits, underscores (_), and dollar signs ($).
- must start with a letter, an underscore (_), or a dollar sign ($).
- cannot start with a digit.
- cannot be a reserved word. (See Appendix A, "Java Keywords,").
- cannot be `true, false, or null.`
- can be of any length.
- Try to rename radius with this name below. Then, which name can cause error?

  a. `_radius`   b. `Radius1`   c. `1radius`   d. `for`

Variable Declaration and Initialization

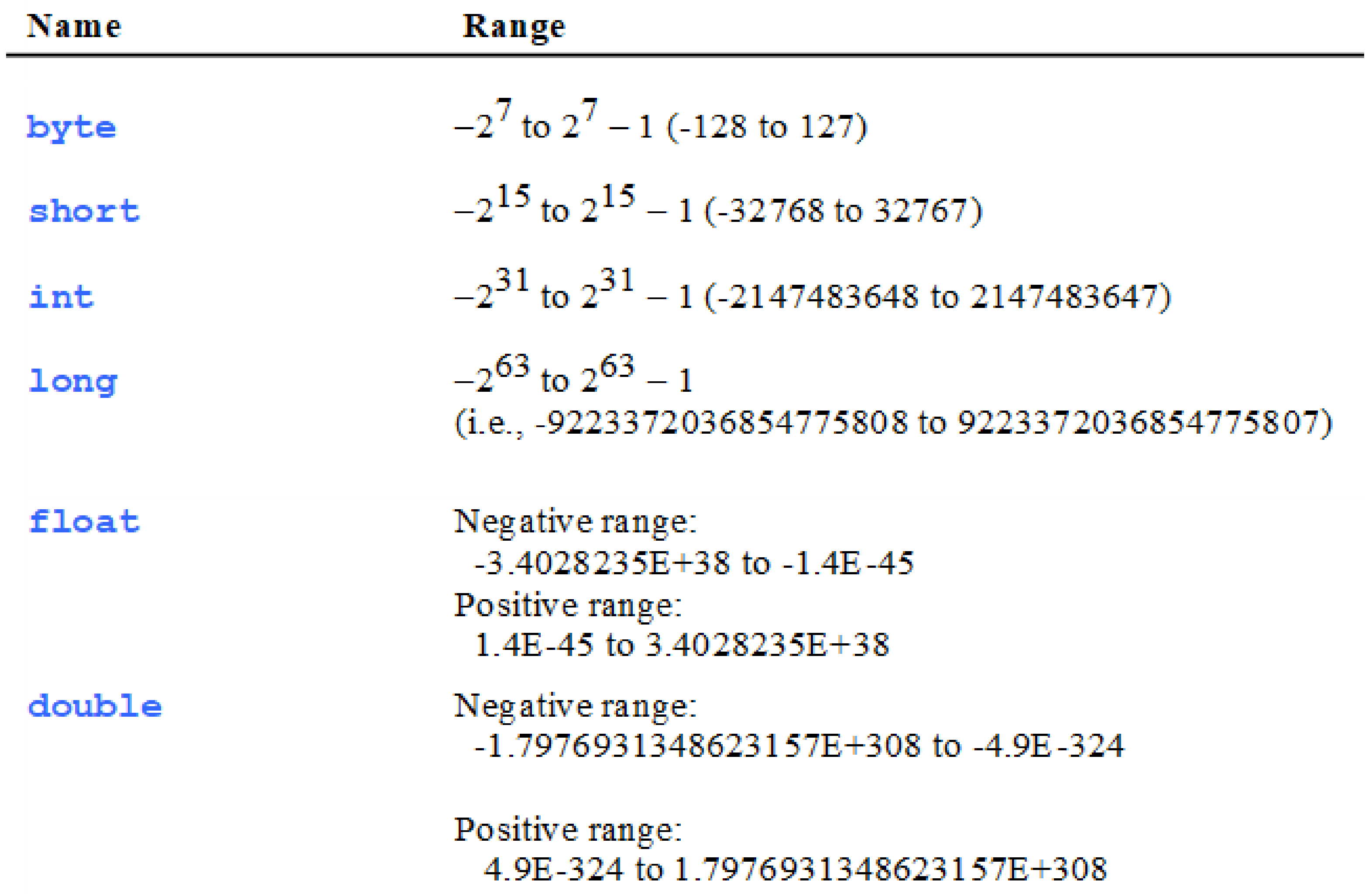
- Do separately… (declare first, then initialize!)

```
double radius;
radius = 2.5;
```

- Or in one step

$$\underbrace{\texttt{double}}_{\text{Data type}}\ \texttt{radius = 2.5;}$$

Numerical Data Types



Name	Range
byte	-2^{7} to $2^{7} - 1$ (-128 to 127)
short	-2^{15} to $2^{15} - 1$ (-32768 to 32767)
int	-2^{31} to $2^{31} - 1$ (-2147483648 to 2147483647)
long	-2^{63} to $2^{63} - 1$ (i.e., -9223372036854775808 to 9223372036854775807)
float	Negative range: -3.4028235E+38 to -1.4E-45 Positive range: 1.4E-45 to 3.4028235E+38
double	Negative range: -1.7976931348623157E+308 to -4.9E-324 Positive range: 4.9E-324 to 1.7976931348623157E+308

Try this...

Change data type of radius and area from double to int and use a radius of 2^14 = 16384.

```
int radius;
int area;
radius = 16384;
area = radius * radius * 3;
```

Then change the radius value to 2^15. What happens?

Number Literals

- A compilation error if the literal were too large for the variable to hold.
  - The statement byte b = 1000 would cause a compilation error, because 1000 cannot be stored in a variable of the byte type.
- By default, a floating-point literal is treated as a double type value.
  - E.g. 5.0 is considered a double value, not a float value. You can make a number a float by appending the letter f or F, and make a number a double by appending the letter d or D. For example, you can use 100.2f or 100.2F for a float number, and 100.2d or 100.2D for a double number.

double vs. float

The double type values are more accurate than the float type values. For example,

```
System.out.println("1.0 / 3.0 is " + 1.0 / 3.0);
```

displays `1.0 / 3.0 is 0.`$\underbrace{\texttt{3333333333333333}}_{\text{16 digits}}$

```
System.out.println("1.0F / 3.0F is " + 1.0F / 3.0F);
```

displays `1.0F / 3.0F is 0.`$\underbrace{\texttt{3333333}}_{\text{7 digits}}\texttt{4}$

Naming Conventions

Variables and methods:	Use lowercase. If the name consists of several words, concatenate all in one, use lowercase for the first word, and capitalize the first letter of each subsequent word in the name. For example, the variables radius and area, and the method `computeArea`.
Classes:	Capitalize the first letter of each word in the name. For example, the class name `ComputeArea`.
Constants:	Capitalize all letters in constants, and use underscores to connect words. For example, the constant `PI` and `MAX_VALUE`

coding

Code: Compute Area of A Circle

https://liveexample.pearsoncmg.com/html/ComputeArea.html

Try this…

1. Rename radius with your own variable name (e.g. jarijari)
2. Change the value of pi (∏ = 3.14…) to 3. What type of the output?
3. Rewrite ∏ as 3.14, then try to calculate perimeter, too… (2∏r)

coding

Constant Variable with "final" keyword

- YOAO – You only assign once: the value is only assigned once in a lifetime ☺
- Try this ...
  - Use a variable PI as a constant of 3.14. Then, compile & run.
    ```
    final double PI = 3.14;
    area = radius * radius * PI;
    ```
  - Try to reset the value of PI after compute the area.
    ```
    PI = 3.141;
    ```
  - What happens?

Reading Input from the Console

1. Create a Scanner object.

```
    Scanner input = new Scanner(System.in);
```

2. Use the method nextDouble() to obtain to a double value. For example,

```
    System.out.print("Enter a double value: ");
    Scanner input = new Scanner(System.in);
    double d = input.nextDouble();
```

Reading Numbers from the Keyboard

```
Scanner input = new Scanner(System.in);
int value = input.nextInt();
```

Method	Description
`nextByte()`	reads an integer of the `byte` type.
`nextShort()`	reads an integer of the `short` type.
`nextInt()`	reads an integer of the `int` type.
`nextLong()`	reads an integer of the `long` type.
`nextFloat()`	reads a number of the `float` type.
`nextDouble()`	reads a number of the `double` type.

Code: Compute Area of A Circle with Console Input

https://liveexample.pearsoncmg.com/html/ComputeAreaWithConsoleInput.html

Try this…

1. Give input of integer (e.g. 3).
2. Replace method nextDouble() with nextInt(), but give input of double (e.g. 2.5).
3. Recall about the number literal, 100.2d or 100.2D can be used to assign a double number. Try to give input of 100.2d ! What happens?

Examples

- **Displaying Time**
  Write a program that obtains minutes and remaining seconds from seconds.
  https://liveexample.pearsoncmg.com/html/DisplayTime.html

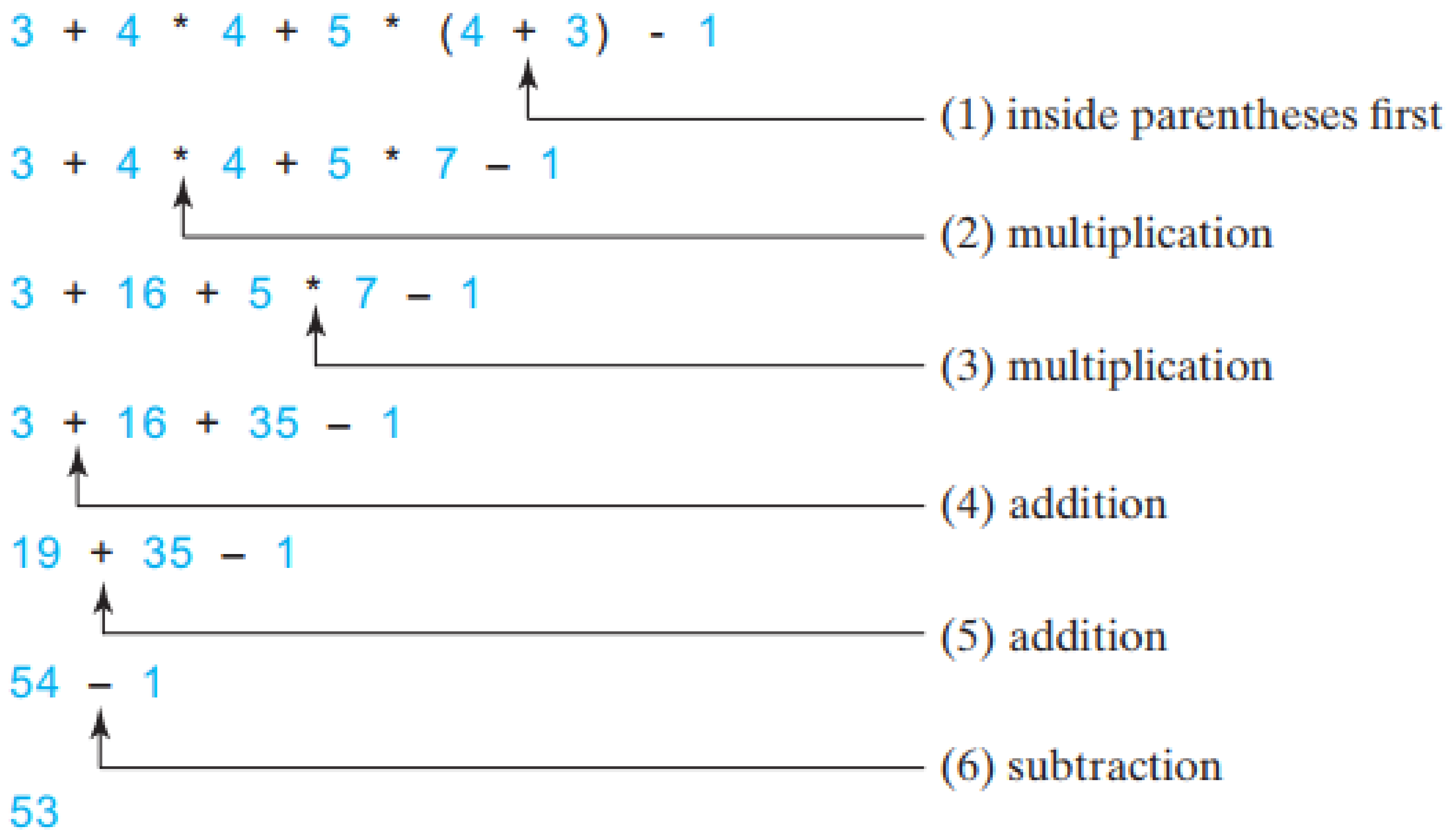
- **Converting Temperatures**
  Write a program that converts a Fahrenheit degree to Celsius using the formula:
  https://liveexample.pearsoncmg.com/html/FahrenheitToCelsius.html

$$C = (\frac{5}{9})(F - 32)$$

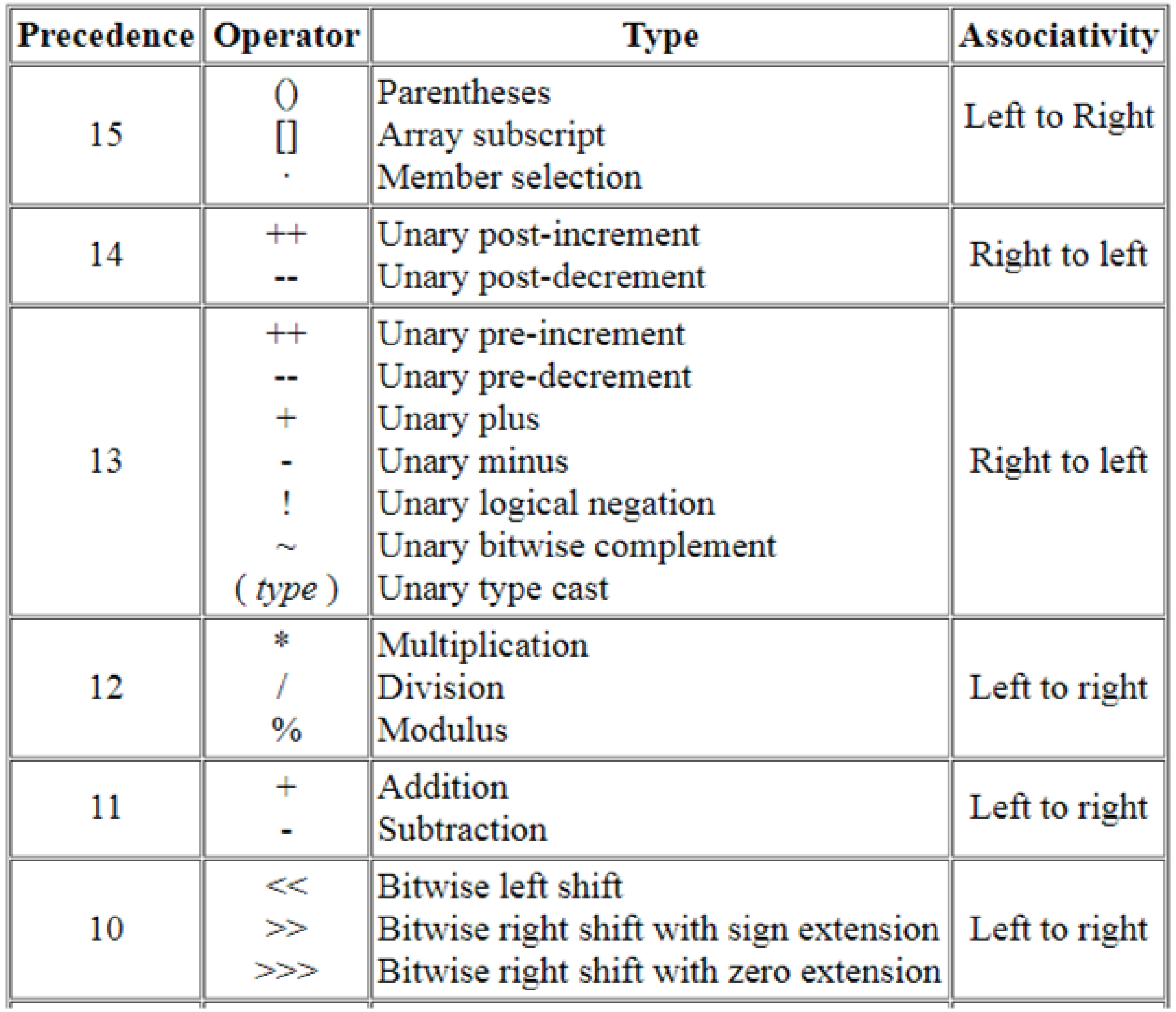
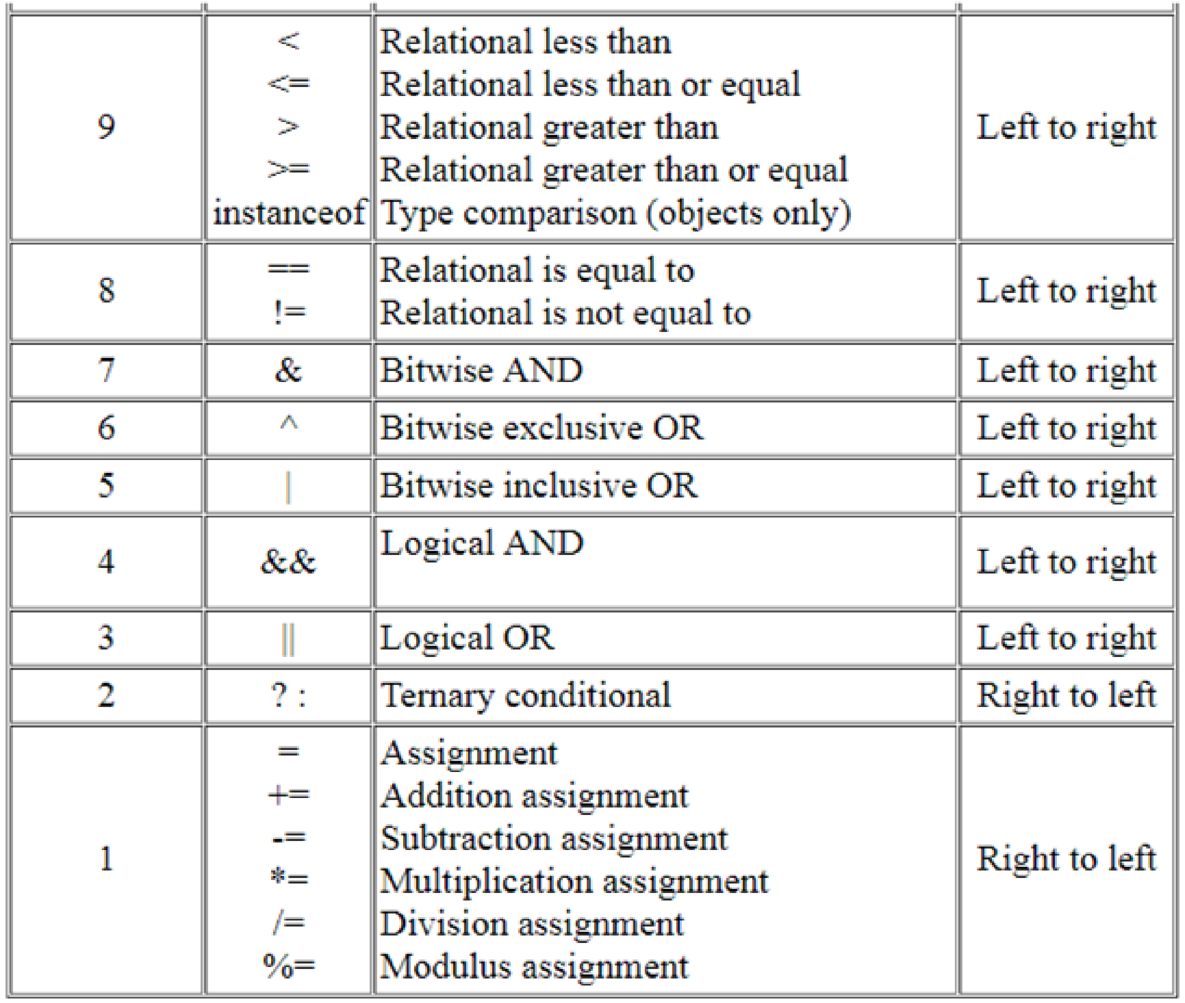
Numeric Operators

Name	Meaning	Example	Result
+	Addition	34 + 1	35
-	Subtraction	34.0 – 0.1	33.9
*	Multiplication	300 * 30	9000
/	Division	1.0 / 2.0	0.5
%	Remainder	20 % 3	2

Precedence of Operators



Precedence of Operators

Precedence	Operator	Type	Associativity
15	() [] .	Parentheses Array subscript Member selection	Left to Right
14	++ --	Unary post-increment Unary post-decrement	Right to left
13	++ -- + - ! ~ (*type*)	Unary pre-increment Unary pre-decrement Unary plus Unary minus Unary logical negation Unary bitwise complement Unary type cast	Right to left
12	* / %	Multiplication Division Modulus	Left to right
11	+ -	Addition Subtraction	Left to right
10	<< >> >>>	Bitwise left shift Bitwise right shift with sign extension Bitwise right shift with zero extension	Left to right
9	< <= > >= instanceof	Relational less than Relational less than or equal Relational greater than Relational greater than or equal Type comparison (objects only)	Left to right
8	== !=	Relational is equal to Relational is not equal to	Left to right
7	&	Bitwise AND	Left to right
6	^	Bitwise exclusive OR	Left to right
5	|	Bitwise inclusive OR	Left to right
4	&&	Logical AND	Left to right
3	||	Logical OR	Left to right
2	? :	Ternary conditional	Right to left
1	= += -= *= /= %=	Assignment Addition assignment Subtraction assignment Multiplication assignment Division assignment Modulus assignment	Right to left

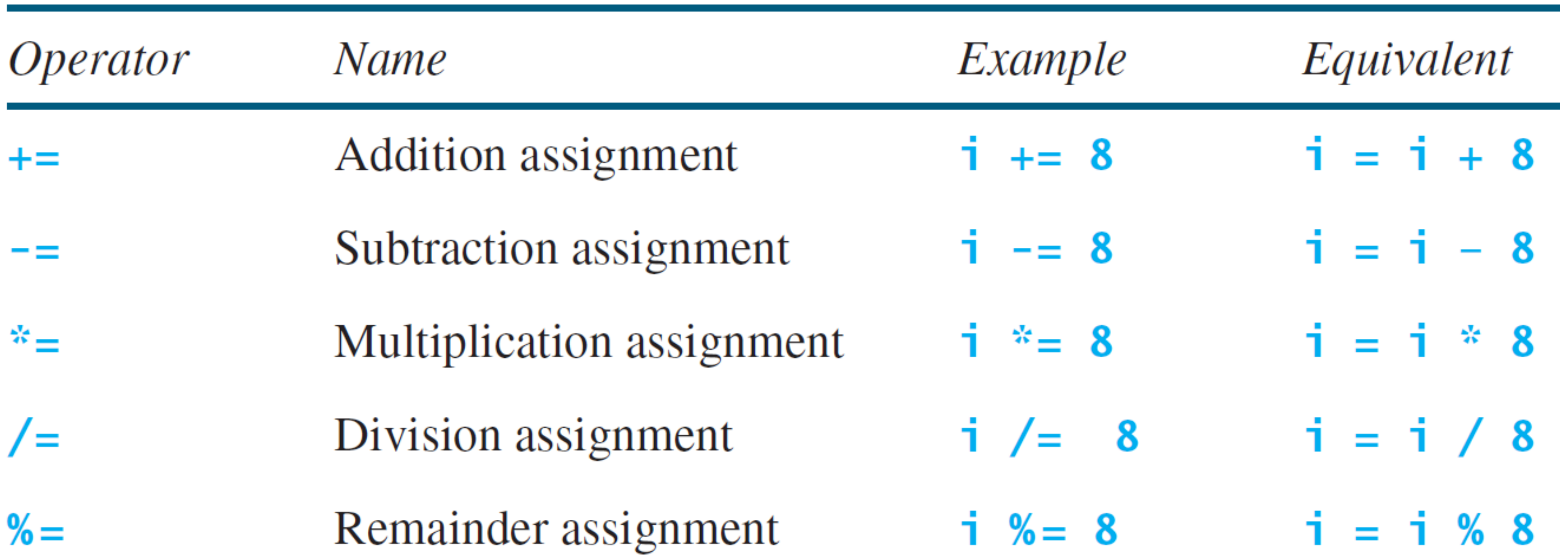
Augmented Assignment Operators

- We can rewrite this with augmented assignment operators:

```
area = radius * radius * 3.14;
```

Solution:

```
area = radius;
area *= radius;
area *= 3.14;
```



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`

Increment and Decrement Operators

What is the output?

```
int i = 2;
System.out.println(7 - ++i * 1.5 / 3 + 6 % 4 - ++i);
System.out.println(7 - i++ * 1.5 / 3 + 6 % 4 - ++i);
```

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`

Increment and Decrement Operators

What is the output?

```
int i = 2;
System.out.println(7 - ++i * 1.5 / 3 + 6 % 4 - ++i);
System.out.println(7 - i++ * 1.5 / 3 + 6 % 4 - ++i);
```

3.5
4.0

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`

Numeric Type Conversion

- Consider the following statements:

```
byte i = 100;
long k = i * 3 + 4;
double d = i * 3 + k / 2;
```

- Try this... And what happens?

```
int x = d + 1;
```

Type Casting

- Implicit casting

```
double d = 3;
```

type widening

- Explicit casting

```
int i = (int)3.0;
int i = (int)3.9;
```

type narrowing

fraction part is truncated

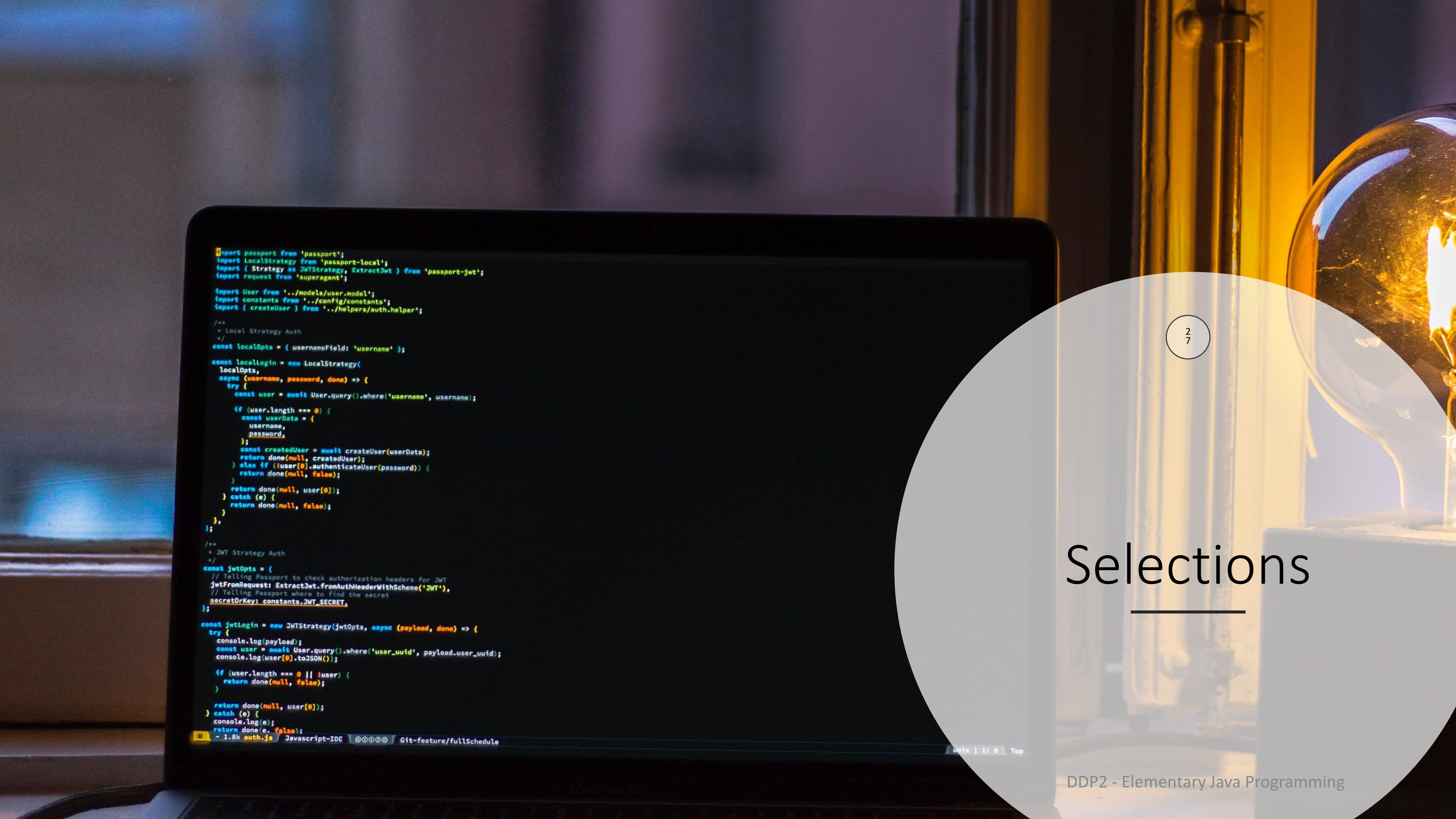
- What is wrong?

```
int x = 5 / 2.0;
```

range increases

$\longrightarrow$

byte, short, int, long, float, double



```
import passport from 'passport';
import LocalStrategy from 'passport-local';
import { Strategy as JWTStrategy, ExtractJwt } from 'passport-jwt';
import request from 'superagent';

import User from '../models/user.model';
import constants from '../config/constants';
import { createUser } from '../helpers/auth.helper';

/**
 * Local Strategy Auth
 */
const localOpts = { usernameField: 'username' };

const localLogin = new LocalStrategy(
  localOpts,
  async (username, password, done) => {
    try {
      const user = await User.query().where('username', username);

      if (user.length === 0) {
        const userData = {
          username,
          password,
        };
        const createdUser = await createUser(userData);
        return done(null, createdUser);
      } else if (!user[0].authenticateUser(password)) {
        return done(null, false);
      }
      return done(null, user[0]);
    } catch (e) {
      return done(null, false);
    }
  },
);

/**
 * JWT Strategy Auth
 */
const jwtOpts = {
  // Telling Passport to check authorization headers for JWT
  jwtFromRequest: ExtractJwt.fromAuthHeaderWithScheme('JWT'),
  // Telling Passport where to find the secret
  secretOrKey: constants.JWT_SECRET,
};

const jwtLogin = new JWTStrategy(jwtOpts, async (payload, done) => {
  try {
    console.log(payload);
    const user = await User.query().where('user_uuid', payload.user_uuid);
    console.log(user[0].toJSON());

    if (user.length === 0 || !user) {
      return done(null, false);
    }

    return done(null, user[0]);
  } catch (e) {
    console.log(e);
    return done(e, false);
```

Selections

coding

Code: Compute Area of A Circle with Console Input

https://liveexample.pearsoncmg.com/html/ComputeAreaWithConsoleInput.html

Try this…

1. Give input -8
2. What is the result?

coding

Code: Compute Area of A Circle with Console Input

https://liveexample.pearsoncmg.com/html/ComputeAreaWithConsoleInput.html

Try this…

1. Give input -8
2. What is the result?

- For a negative value for `radius`, ComputeAreaWithConsoleInput.java would print an invalid result.
- If the radius is negative, you don't want the program to compute the area.
- How can you deal with this situation?

Selections

- `boolean` variables and expressions
- **if** statements
- **if-else** statements
- nested **if** and multi-way **if** statements
- Common errors and pitfalls in **if** statements
- Logical operators
- To implement selection control using switch statements
- The conditional expression
- Operator precedence and associativity

The **boolean** Type and Operators

- Often in a program you need to compare two values, such as whether i is greater than j. Java provides six comparison operators (also known as relational operators) that can be used to compare two values. The result of the comparison is a Boolean value: true or false.

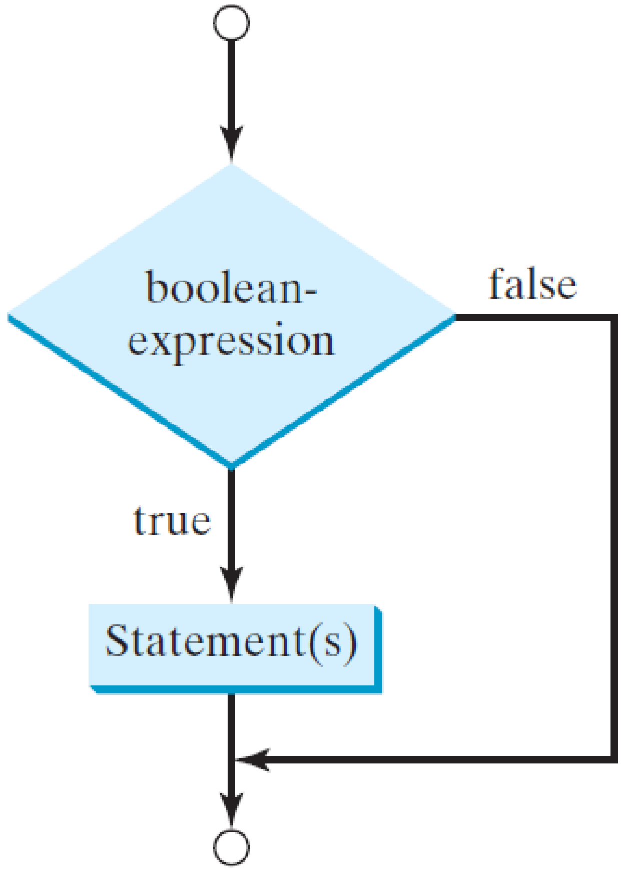
```
boolean b = (1 > 2);
```

Relational Operators

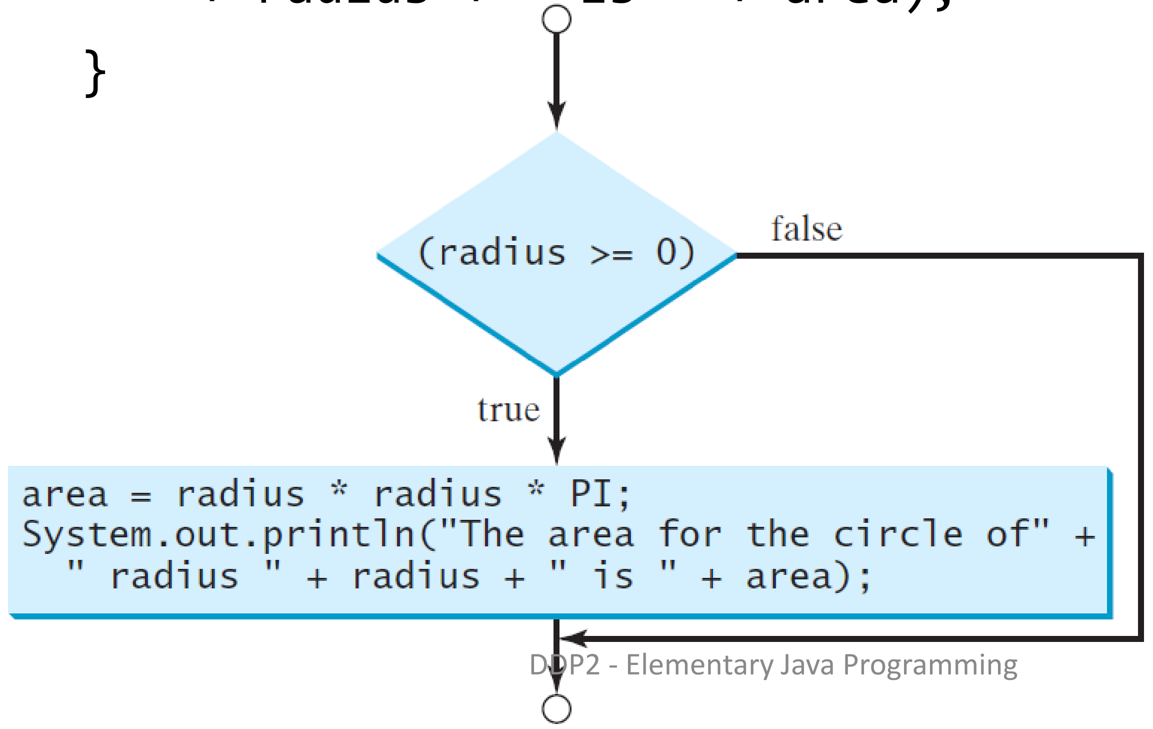
Java Operator	Mathematics Symbol	Name	Example (radius is 5)	Result
`<`	$<$	less than	`radius < 0`	`false`
`<=`	$\leq$	less than or equal to	`radius <= 0`	`false`
`>`	$>$	greater than	`radius > 0`	`true`
`>=`	$\geq$	greater than or equal to	`radius >= 0`	`true`
`==`	$=$	equal to	`radius == 0`	`false`
`!=`	$\neq$	not equal to	`radius != 0`	`true`

One-way `if` Statements

```
if (boolean-expression) {
  statement(s);
}
```



```
if (radius >= 0) {
   area = radius * radius * PI;
   System.out.println("The area"
      + " for the circle of radius "
      + radius + " is " + area);
}
```



```
area = radius * radius * PI;
System.out.println("The area for the circle of" +
  " radius " + radius + " is " + area);
```

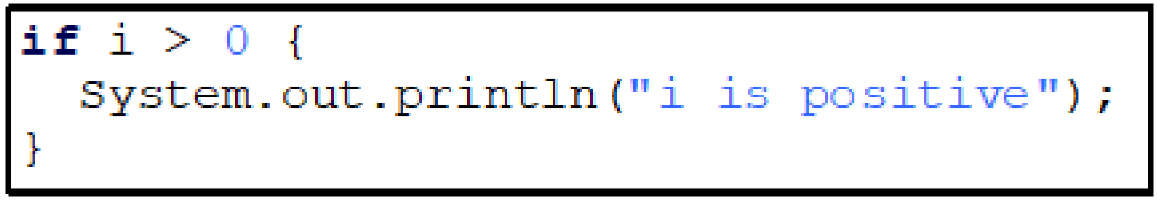
coding

Code: Compute Area of A Circle with If

- Handle the negative values!

Note

- What’s wrong?



```
if i > 0 {
  System.out.println("i is positive");
}
```

Note

- What’s wrong?

```
if i > 0 {
  System.out.println("i is positive");
}
```

```
if (i > 0) {
  System.out.println("i is positive");
}
```

```
if (i > 0) {
  System.out.println("i is positive");
}
```

(a)

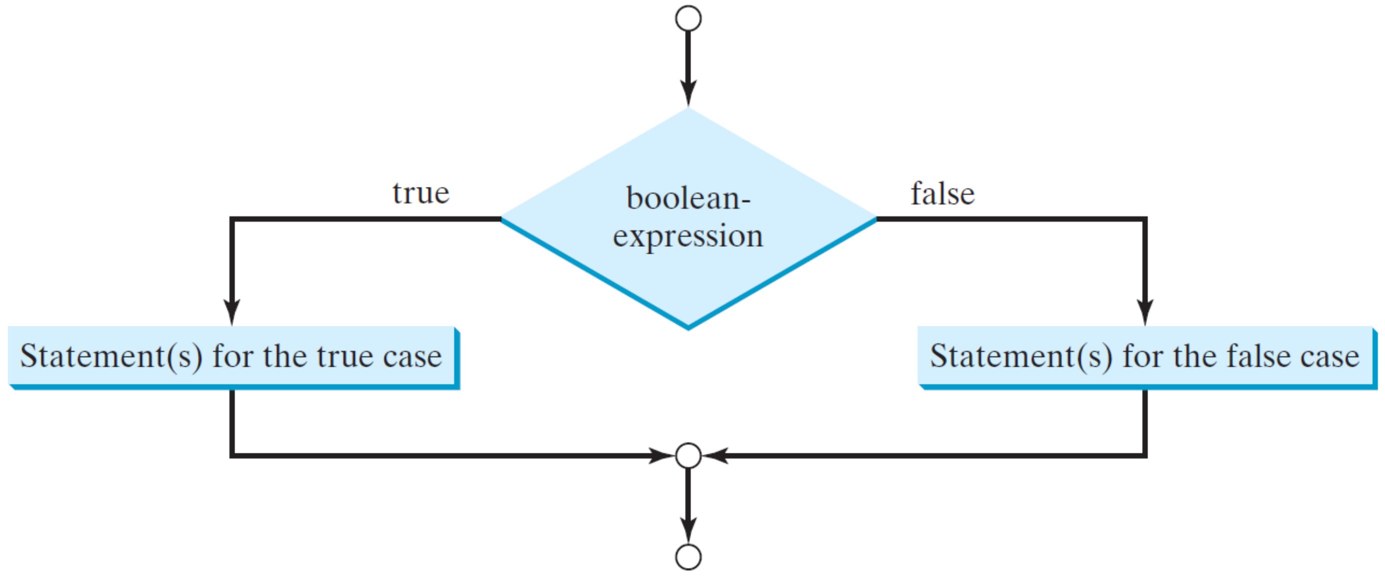
Equivalent

```
if (i > 0)
  System.out.println("i is positive");
```

(b)

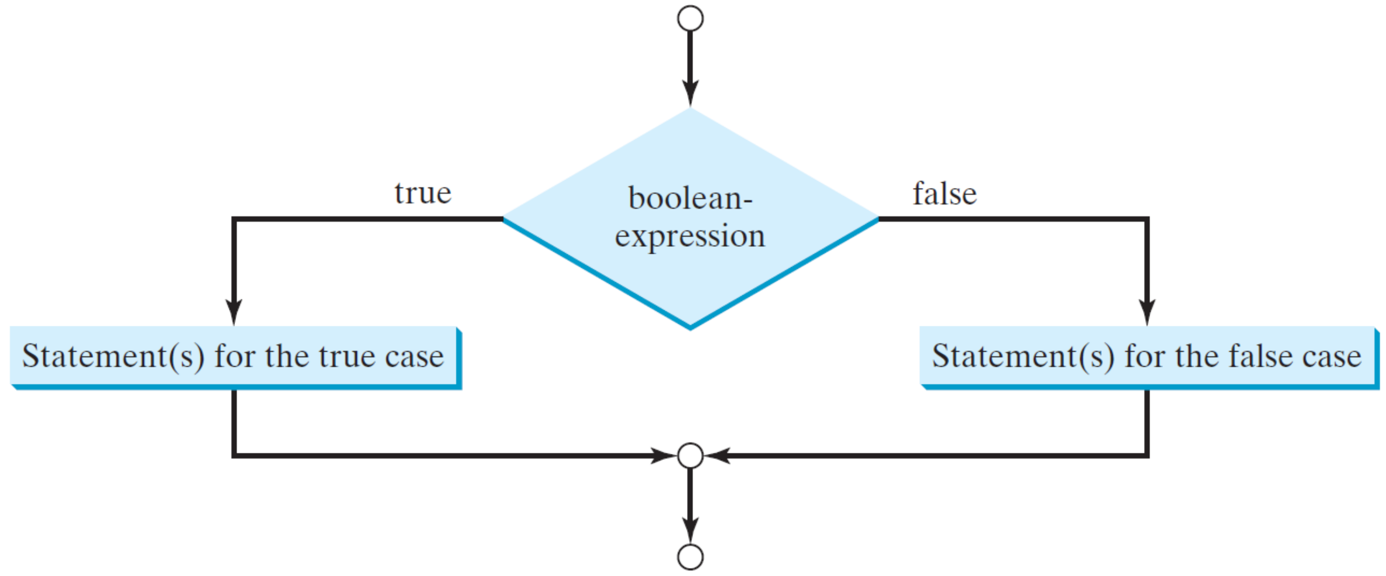
Only for the first statement after if!

The Two-way `if` Statement (If-Else)


true
boolean-expression
false
Statement(s) for the true case
Statement(s) for the false case

```
if (boolean-expression) {
  statement(s)-for-the-true-case;
}
else {
  statement(s)-for-the-false-case;
}
```

`if-else` Example


true
boolean-expression
false
Statement(s) for the true case
Statement(s) for the false case

```
if (radius >= 0) {
  area = radius * radius * 3.14159;

  System.out.println("The area for the “
    + “circle of radius " + radius + " is " + area);
}
else {
  System.out.println("Negative input");
}
```

Multi-Way if-else Statements

```
if (score >= 90.0)
  System.out.print("A");
else
  if (score >= 80.0)
    System.out.print("B");
  else
    if (score >= 70.0)
      System.out.print("C");
    else
      if (score >= 60.0)
        System.out.print("D");
      else
        System.out.print("F");
```

(a)

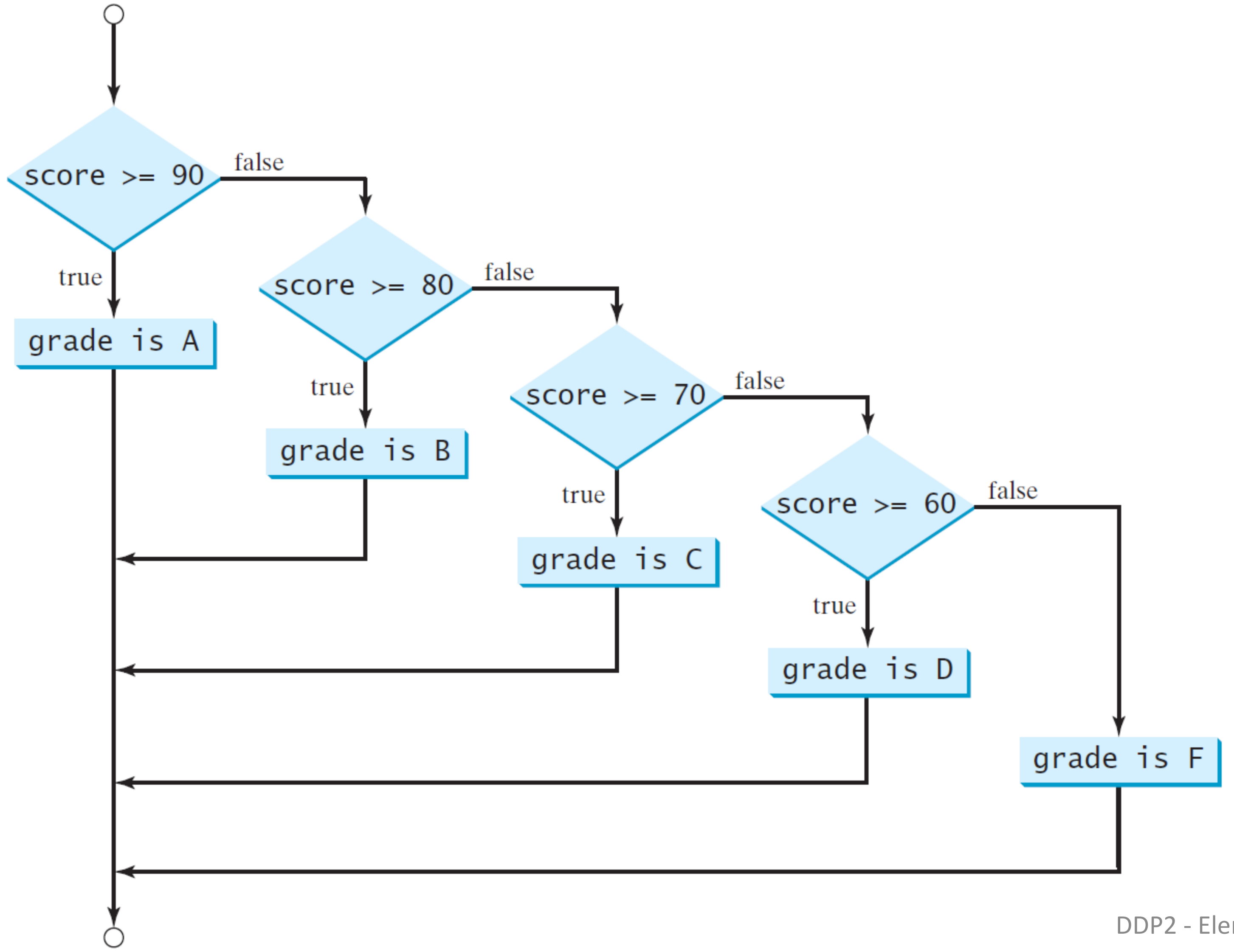
Equivalent

This is better

```
if (score >= 90.0)
  System.out.print("A");
else if (score >= 80.0)
  System.out.print("B");
else if (score >= 70.0)
  System.out.print("C");
else if (score >= 60.0)
  System.out.print("D");
else
  System.out.print("F");
```

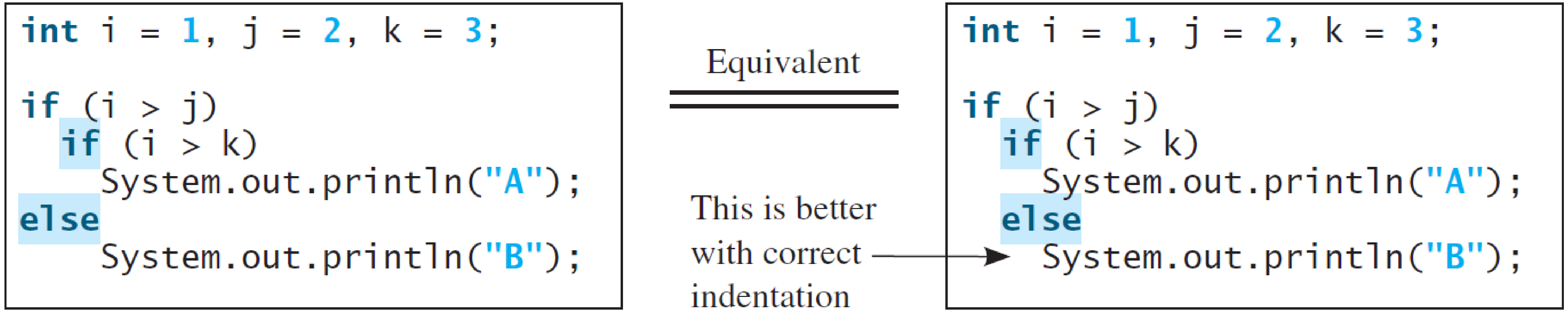
(b)

Multi-Way if-else Statements



Note!!

- The else clause matches the most recent if clause in the same block.



```
int i = 1, j = 2, k = 3;

if (i > j)
  if (i > k)
    System.out.println("A");
else
    System.out.println("B");
```

(a)

```
int i = 1, j = 2, k = 3;

if (i > j)
  if (i > k)
    System.out.println("A");
  else
    System.out.println("B");
```

(b)

Equivalent
This is better with correct indentation

- What is the output?

Note (2)

- Nothing is printed from the lefthand statement

```
int i = 1;
int j = 2;
int k = 3;
if (i > j)
  if (i > k)
    System.out.println("A");

  else
    System.out.println("B");
```

- To force the else clause to match the first if clause, you must add a pair of braces:

```
int i = 1;
int j = 2;
int k = 3;
if (i > j) {
  if (i > k)
    System.out.println("A");
}
  else
    System.out.println("B");
```

Common Errors

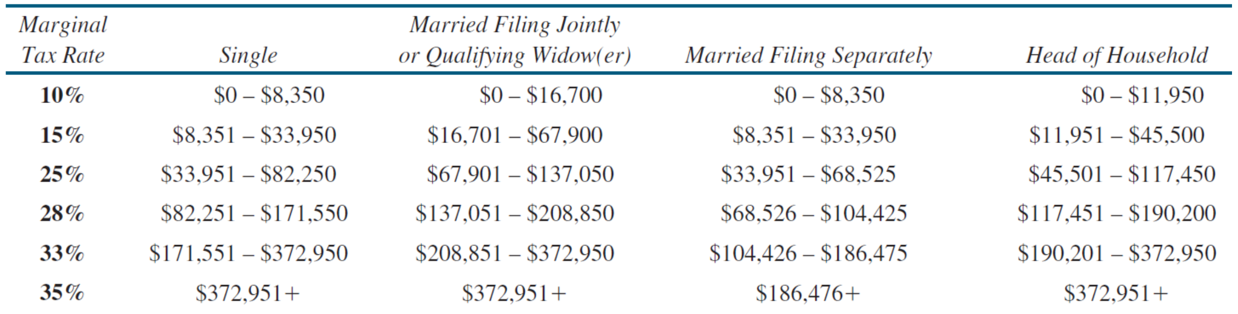
- Adding a semicolon at the end of an if clause is a common mistake.

```
if (radius >= 0);
{
  area = radius*radius*PI;
  System.out.println(
    "The area for the circle of radius " +
    radius + " is " + area);
}
```

- Not a compilation error or a runtime error, it is a **logic** error.

Recall: Multi-way Ifs

- The US federal personal income tax is calculated based on the filing status and taxable income. There are four filing statuses: single filers, married filing jointly, married filing separately, and head of household. The tax rates for 2009 are shown below.



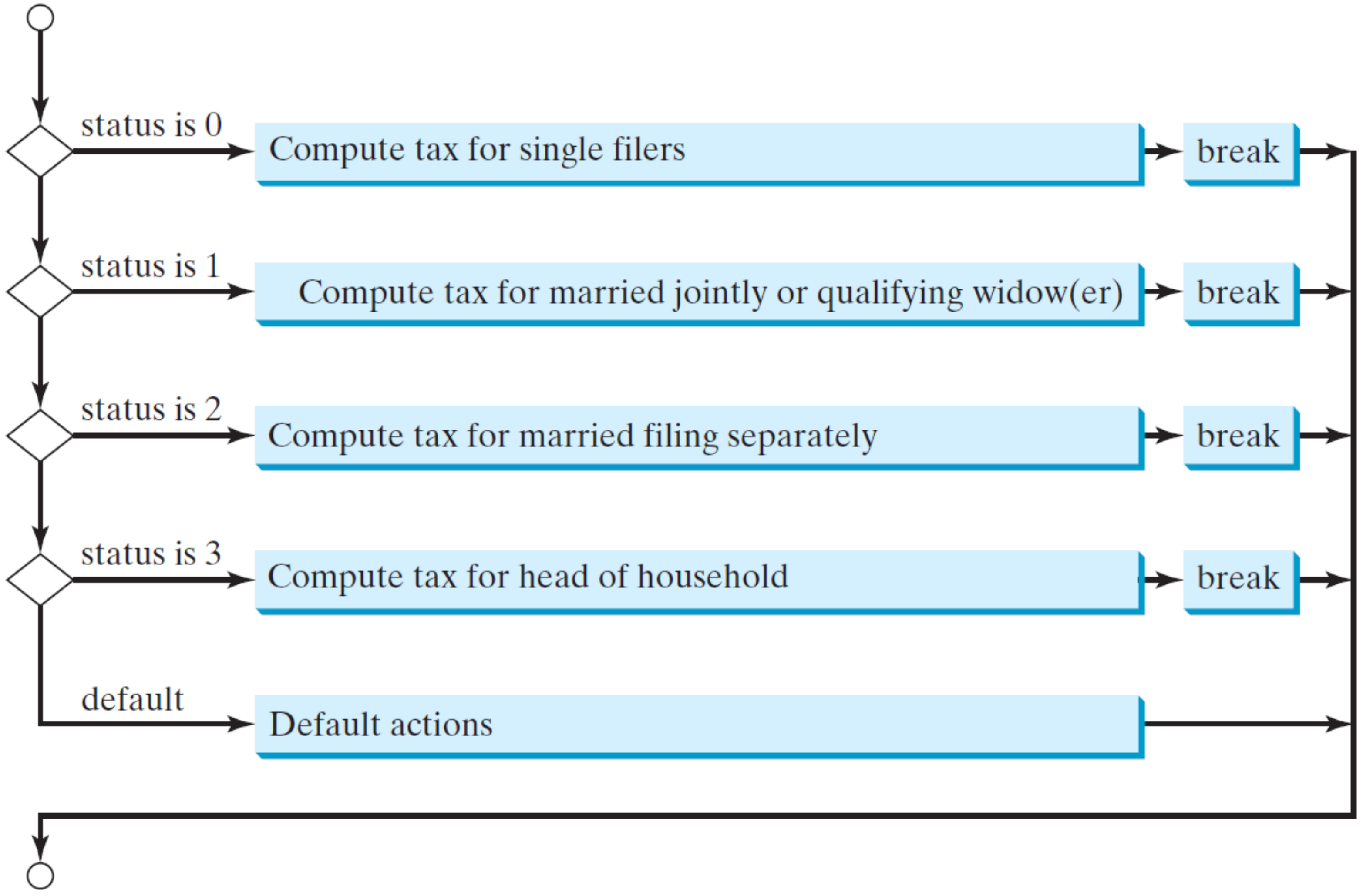
Marginal Tax Rate	*Single*	*Married Filing Jointly or Qualifying Widow(er)*	*Married Filing Separately*	*Head of Household*
10%	\$0 – \$8,350	\$0 – \$16,700	\$0 – \$8,350	\$0 – \$11,950
15%	\$8,351 – \$33,950	\$16,701 – \$67,900	\$8,351 – \$33,950	\$11,951 – \$45,500
25%	\$33,951 – \$82,250	\$67,901 – \$137,050	\$33,951 – \$68,525	\$45,501 – \$117,450
28%	\$82,251 – \$171,550	\$137,051 – \$208,850	\$68,526 – \$104,425	\$117,451 – \$190,200
33%	\$171,551 – \$372,950	\$208,851 – \$372,950	\$104,426 – \$186,475	\$190,201 – \$372,950
35%	\$372,951+	\$372,951+	\$186,476+	\$372,951+

```
if (status == 0) {
  // Compute tax for single filers
}
else if (status == 1) {
  // Compute tax for married file jointly
  // or qualifying widow(er)
}
else if (status == 2) {
  // Compute tax for married file separately
}
else if (status == 3) {
  // Compute tax for head of household
}
else {
  // Display wrong status
}
```

This is correct, but it looks complex.. Any other ways to do this?

Python

`switch` Statements


status is 0
Compute tax for single filers
break
status is 1
Compute tax for married jointly or qualifying widow(er)
break
status is 2
Compute tax for married filing separately
break
status is 3
Compute tax for head of household
break
default
Default actions

Python

`switch` Statements

```
switch (status) {
  case 0:  // compute for single filers;
           break;
  case 1:  // compute for married file jointly;
           break;
  case 2:  // compute for married file separately;
           break;
  case 3:  // compute for head of household;
           break;
  default:
           System.out.println("invalid status");
           System.exit(1);
}
```

*) A switch works with the byte, short, char, and int primitive data types. It also works with enumerated types (discussed in Enum Types), the String class, and a few special classes that wrap certain primitive types: Character, Byte, Short, and Integer (discussed in Numbers and Strings).

`switch` Statement Rules

Must be a value of char, byte, short, or int primitive types *

```
switch (switch-expression) {
   case value1:  statement(s)1;
                 break;
   case value2: statement(s)2;
                 break;

   …
   case valueN: statement(s)N;
                 break;
   default: statement(s)-for-default;
}
```

The value1, ..., and valueN must have the same data type as the value of the switch-expression.
Note: value1, ..., and valueN are constant expressions, meaning that they cannot contain variables in the expression, such as 1 + x.

statement are executed when the value in the case statement matches the value of the switch-expression.

`switch` Statement Rules

The keyword break is *optional*, but it should be used at the end of each case to terminate the remainder of the switch statement. **Note:** If the break statement is not present, the next case statement will be executed UNTIL either a **break** statement or the end of the **switch** statement is reached..

```
switch (switch-expression) {
  case value1:  statement(s)1;
                 break;
  case value2: statement(s)2;
                 break;
  …
  case valueN: statement(s)N;
                 break;
  default: statement(s)-for-default;
}
```

The default case, which is optional, can be used to perform actions when none of the specified cases matches the switch-expression.

Example: Weekday or Weekend

```
int day = 2;
switch (day) {
  case 1: System.out.println("Weekday"); break;
  case 2: System.out.println("Weekday");
  case 3: System.out.println("Weekday"); break;
  case 4: System.out.println("Weekday"); break;
  case 5: System.out.println("Weekday"); break;
  case 0: System.out.println("Weekend"); break;
  case 6: System.out.println("Weekend");
}
```

```
Weekday
 Weekday
```

Pay Attention to the **break!**

Example: Weekday or Weekend

```
int day = 2;
switch (day) {
  case 1: System.out.println("Weekday"); break;
  case 2: System.out.println("Weekday"); break;
  case 3: System.out.println("Weekday"); break;
  case 4: System.out.println("Weekday"); break;
  case 5: System.out.println("Weekday"); break;
  case 0: System.out.println("Weekend"); break;
  case 6: System.out.println("Weekend");
}
```

//OUTPUT: ___Weekday_______________

Pay Attention to the **break!**

Example: Weekday or Weekend

```
int day = 2;
switch (day) {
  case 1:
  case 2:
  case 3:
  case 4:
  case 5: System.out.println("Weekday"); break;
  case 0:
  case 6: System.out.println("Weekend");
}
```

//OUTPUT: ____Weekday____________________

Pay Attention to the **break!**

Logical Operators

Operator	Name	Description
!	not	logical negation
&&	short circuit AND	logical conjunction
||	short circuit OR	logical disjunction
^	exclusive OR	logical exclusion

- Sometimes, whether a statement is executed is determined by a combination of several conditions.
- Example: TestBooleanOperators

  https://liveexample.pearsoncmg.com/html/TestBooleanOperators.html

The & and | Operators

- A short circuit operator (&&, ||) is an operator that doesn't necessarily check all of its operands.
- Try this…

```
int x = 1;
if((x > 1) && (x++ < 10))
    System.out.println("do something");
    // what is this output?
System.out.println("x = " + x);
// Try to change & with &&
```

Conditional Expressions

```
if (x > 0)
  y = 1
else
  y = -1;
```

=

```
y = (x > 0) ? 1 : -1;
```

```
(boolean-expression) ? expression1 : expression2
```

The symbols ? and : appearing together as a *conditional operator* or *ternary operator*.

Use conditional expressions to have equivalent result with this:

```
if (num % 2 == 0)
  System.out.println(num + “is even”);
else
  System.out.println(num + “is odd”);
```

```
import passport from 'passport';
import LocalStrategy from 'passport-local';
import { Strategy as JWTStrategy, ExtractJwt } from 'passport-jwt';
import request from 'superagent';

import User from '../models/user.model';
import constants from '../config/constants';
import { createUser } from '../helpers/auth.helper';

/**
 * Local Strategy Auth
 */
const localOpts = { usernameField: 'username' };

const localLogin = new LocalStrategy(
  localOpts,
  async (username, password, done) => {
    try {
      const user = await User.query().where('username', username);

      if (user.length === 0) {
        const userData = {
          username,
          password,
        };
        const createdUser = await createUser(userData);
        return done(null, createdUser);
      } else if (!user[0].authenticateUser(password)) {
        return done(null, false);
      }
      return done(null, user[0]);
    } catch (e) {
      return done(null, false);
    }
  },
);

/**
 * JWT Strategy Auth
 */
const jwtOpts = {
  // Telling Passport to check authorization headers for JWT
  jwtFromRequest: ExtractJwt.fromAuthHeaderWithScheme('JWT'),
  // Telling Passport where to find the secret
  secretOrKey: constants.JWT_SECRET,
};

const jwtLogin = new JWTStrategy(jwtOpts, async (payload, done) => {
  try {
    console.log(payload);
    const user = await User.query().where('user_uuid', payload.user_uuid);
    console.log(user[0].toJSON());

    if (user.length === 0 || !user) {
      return done(null, false);
    }

    return done(null, user[0]);
  } catch (e) {
    console.log(e);
    return done(e, false);
0  - 1.8k auth.js  Javascript-IDE  Git-feature/fullSchedule          unix | 1: 0  Top
```

Loops

Print a string (e.g., "Welcome to Java!") 100 times!

Option 1

```
                 System.out.println("Welcome to Java!");
                 System.out.println("Welcome to Java!");
                 System.out.println("Welcome to Java!");
                 System.out.println("Welcome to Java!");
                 System.out.println("Welcome to Java!");
                 System.out.println("Welcome to Java!");
100
times            …
                 …
                 …
                 System.out.println("Welcome to Java!");
                 System.out.println("Welcome to Java!");
                 System.out.println("Welcome to Java!");
```

Maybe this is easier

Option 2 (use loop)

```
int count = 0;
while (count < 100) {
  System.out.println("Welcome to Java");
  count++;
}
```

Loops

while statements

do-while statements

for statements

Which loop to use?

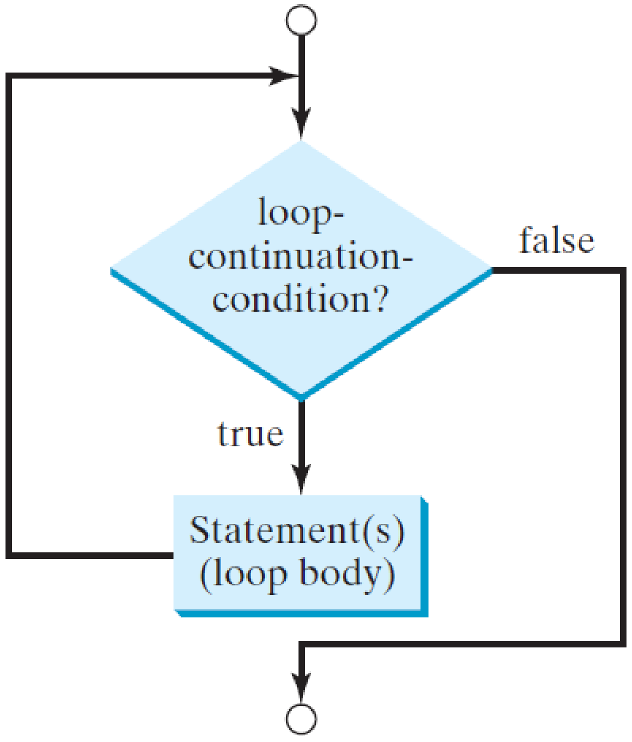
nested loops

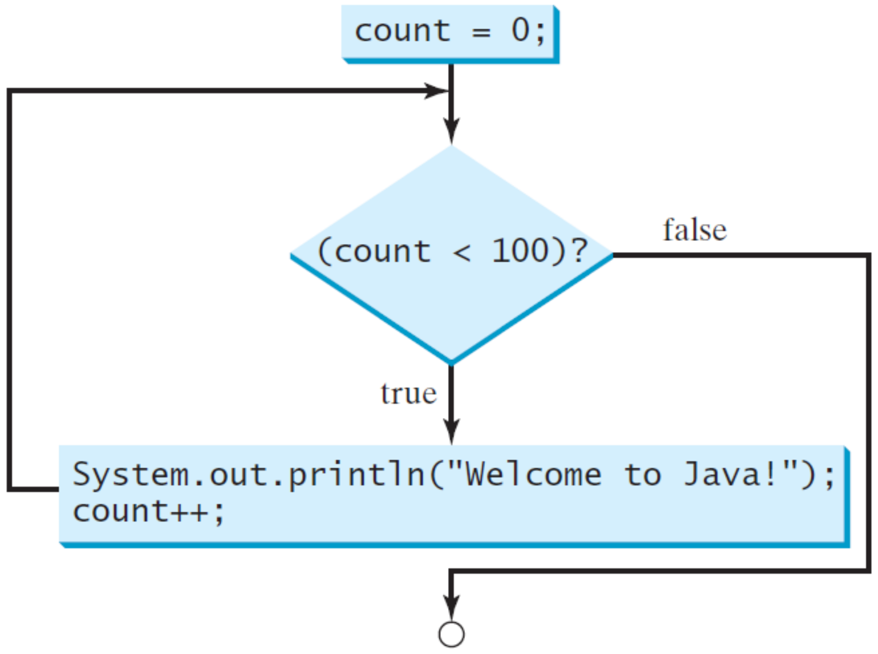
break and **continue**

`while` statements

```
while (loop-continuation-condition) {
  // loop-body;
  Statement(s);
}
```

```
int count = 0;
while (count < 100) {
  System.out.println("Welcome to Java!");
  count++;
}
```





Ending a Loop

- You may use an input value to signify the end of the loop. Such a value is known as a *sentinel value*.

Ending a Loop with a Sentinel Value

https://liveexample.pearsoncmg.com/html/SentinelValue.html

```
// Readinitial data
System.out.print( "Enter an integer (the input ends if it is 0): ");
int data = input.nextInt();
// Keep reading data until the input is 0
int sum = 0;
while (data != 0) {
    sum += data;
    // Read the next data
    System.out.print( "Enter an integer (the input ends if it is 0): ");
    data = input.nextInt();
}
System.out.println("The sum is " + sum);
```

The input value **0** is the sentinel value for this loop.

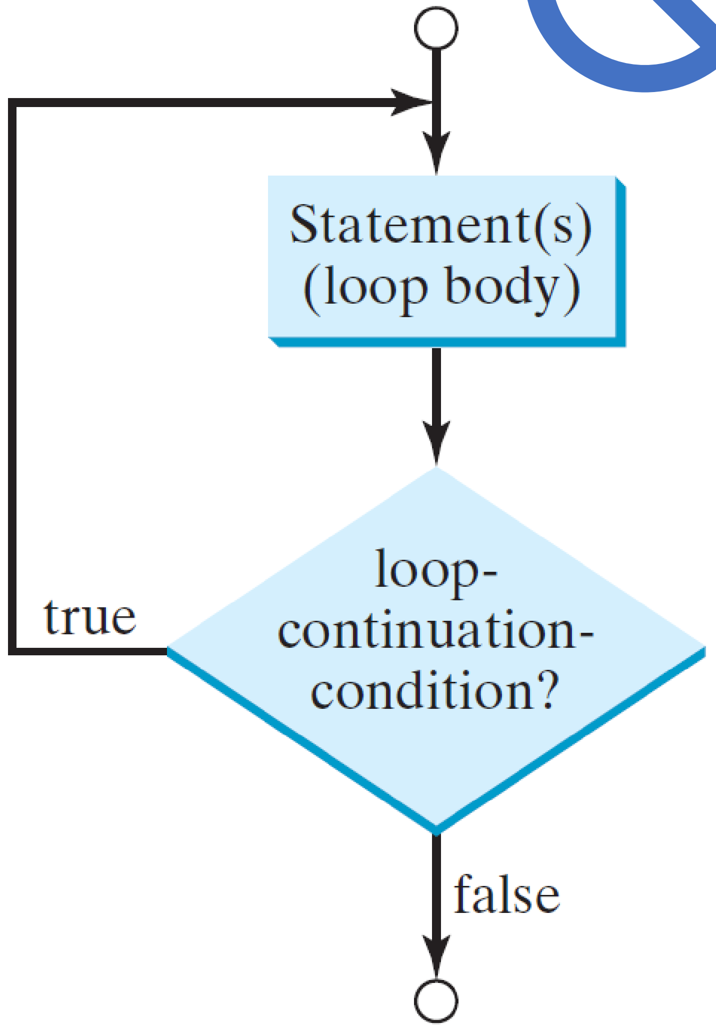
do-while statements

```
do {
  // Loop body;
  Statement(s);
} while (loop-continuation-condition);
```

!!!!

Compare it with while statement:

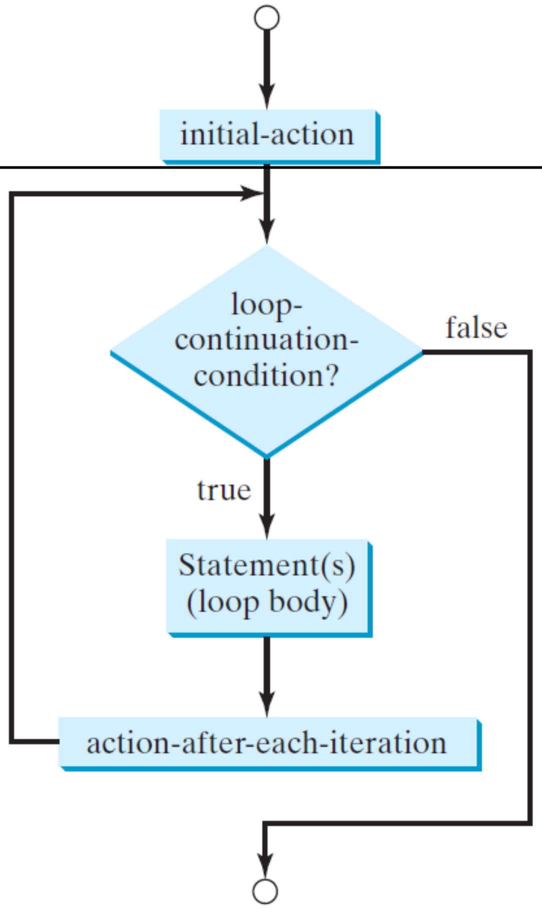
```
while (loop-continuation-condition) {
  // loop-body;
  Statement(s);
}
```



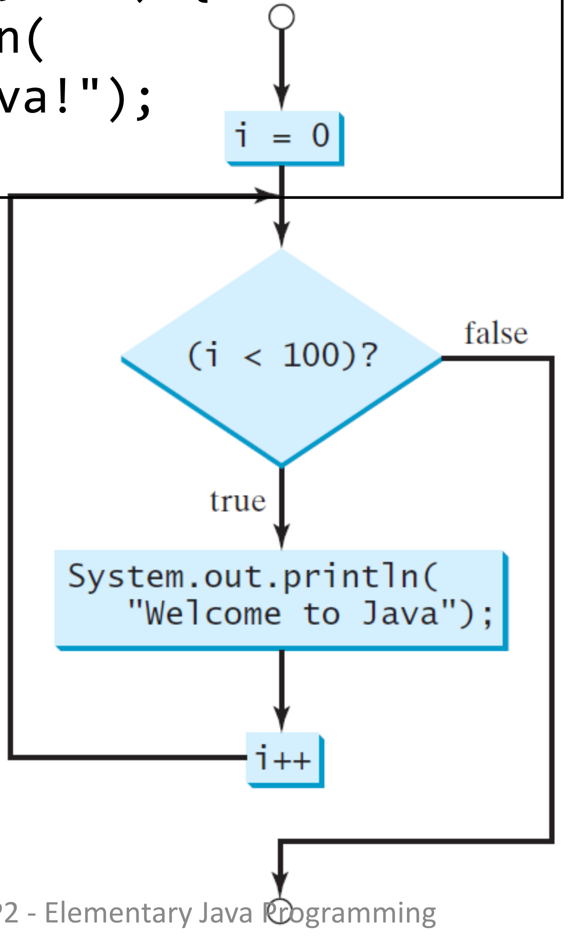
Python

for statements

```
for (initial-action;    loop-continuation-
condition; action-after-each-iteration) {
   // loop body;
   Statement(s);
}
```



```
int i;
for (i = 0; i < 100; i++) {
  System.out.println(
     "Welcome to Java!");
}
```



coding

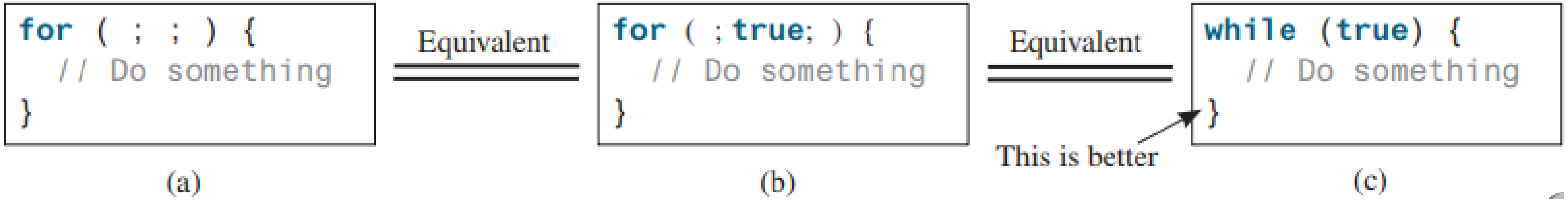
Code: Compute Area of A Circle with Loop

Try this…

1. Calculate the area X times
2. Calculate the area with sentinel value

Note

- If the loop-continuation-condition in a for loop is omitted, it is implicitly true. Thus, the statement given below in (a), which is an infinite loop, is the same as in (b). To avoid confusion, though, it is better to use the equivalent loop in (c).



```
for ( ; ; ) {
  // Do something
}
```

(a)

```
for ( ; true; ) {
  // Do something
}
```

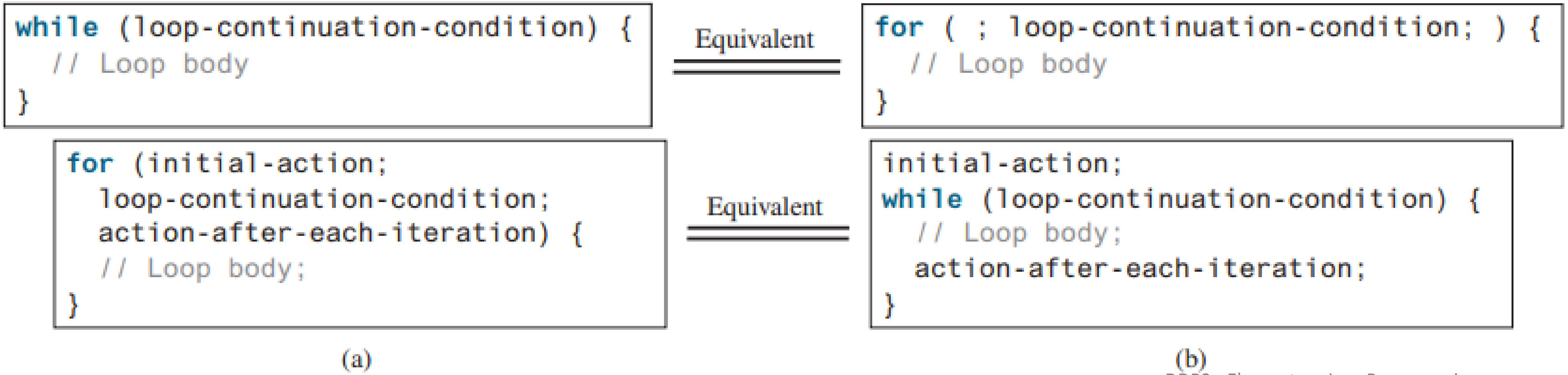
(b)

```
while (true) {
  // Do something
}
```

(c)

Which loop to use?

- a for loop may be used if ______________________________
- A while loop may be used if _____________________________________
- A do-while loop can be used to replace a while loop if ____________________________



```
while (loop-continuation-condition) {
  // Loop body
}
```

```
for ( ; loop-continuation-condition; ) {
  // Loop body
}
```

```
for (initial-action;
  loop-continuation-condition;
  action-after-each-iteration) {
  // Loop body;
}
```

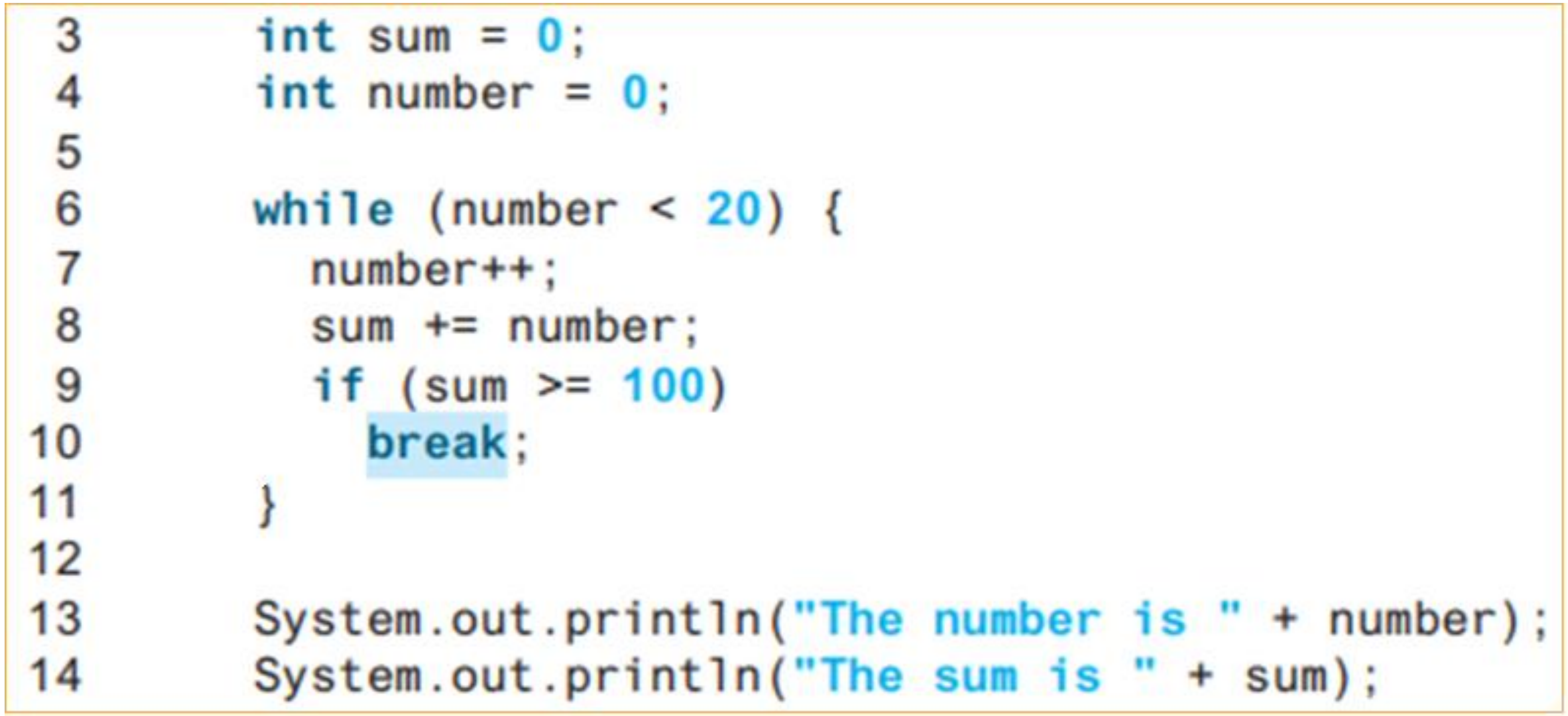
```
initial-action;
while (loop-continuation-condition) {
  // Loop body;
  action-after-each-iteration;
}
```

Nested Loops

```
for (int i = 0; i <= 5; i++){
   for (int j = 0; j <= 4; j++){
       for (int k = 0; k <= 7; k++){
           // do something...
       }
   }
}
```

How many times will it run?

break

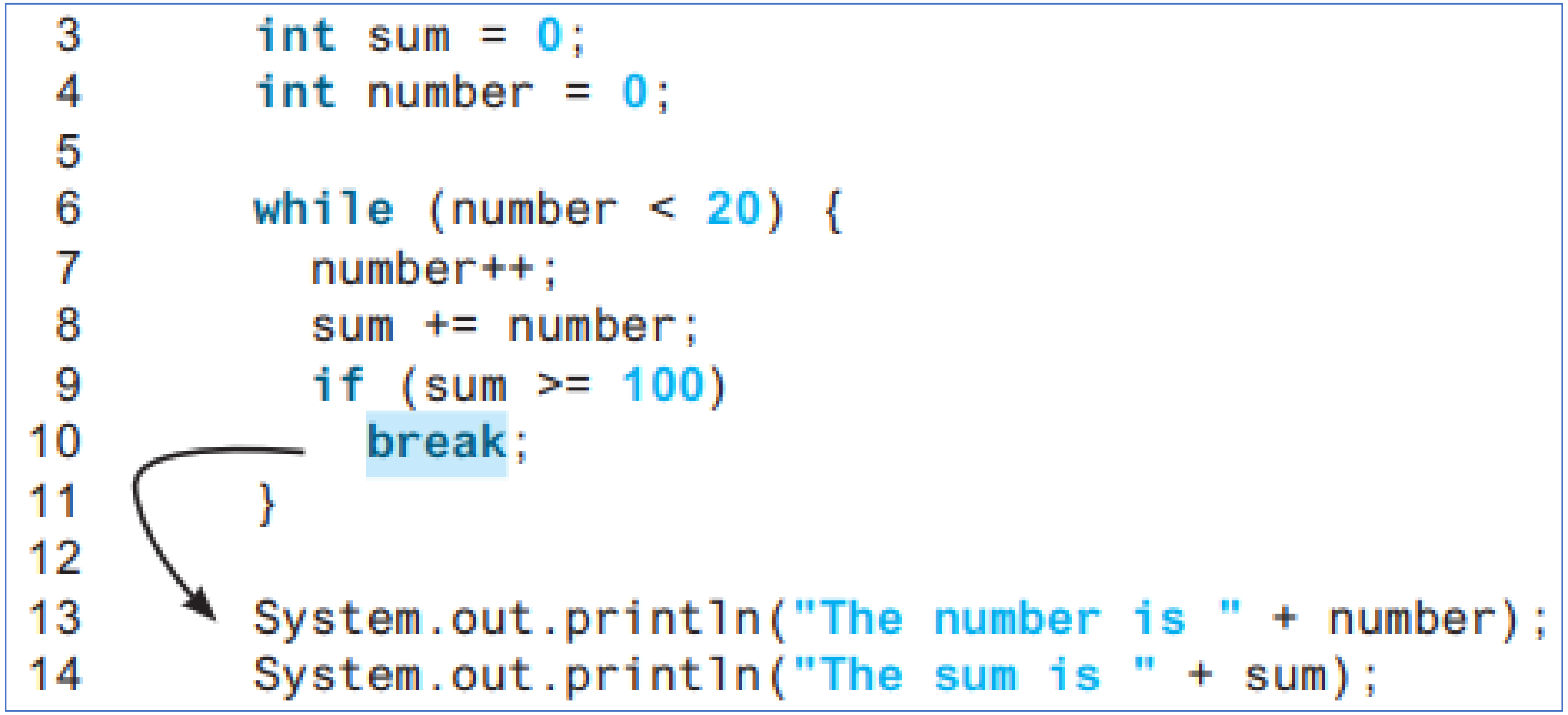


```
    int sum = 0;
    int number = 0;

    while (number < 20) {
      number++;
      sum += number;
      if (sum >= 100)
        break;
    }

    System.out.println("The number is " + number);
    System.out.println("The sum is " + sum);
```

break (2)

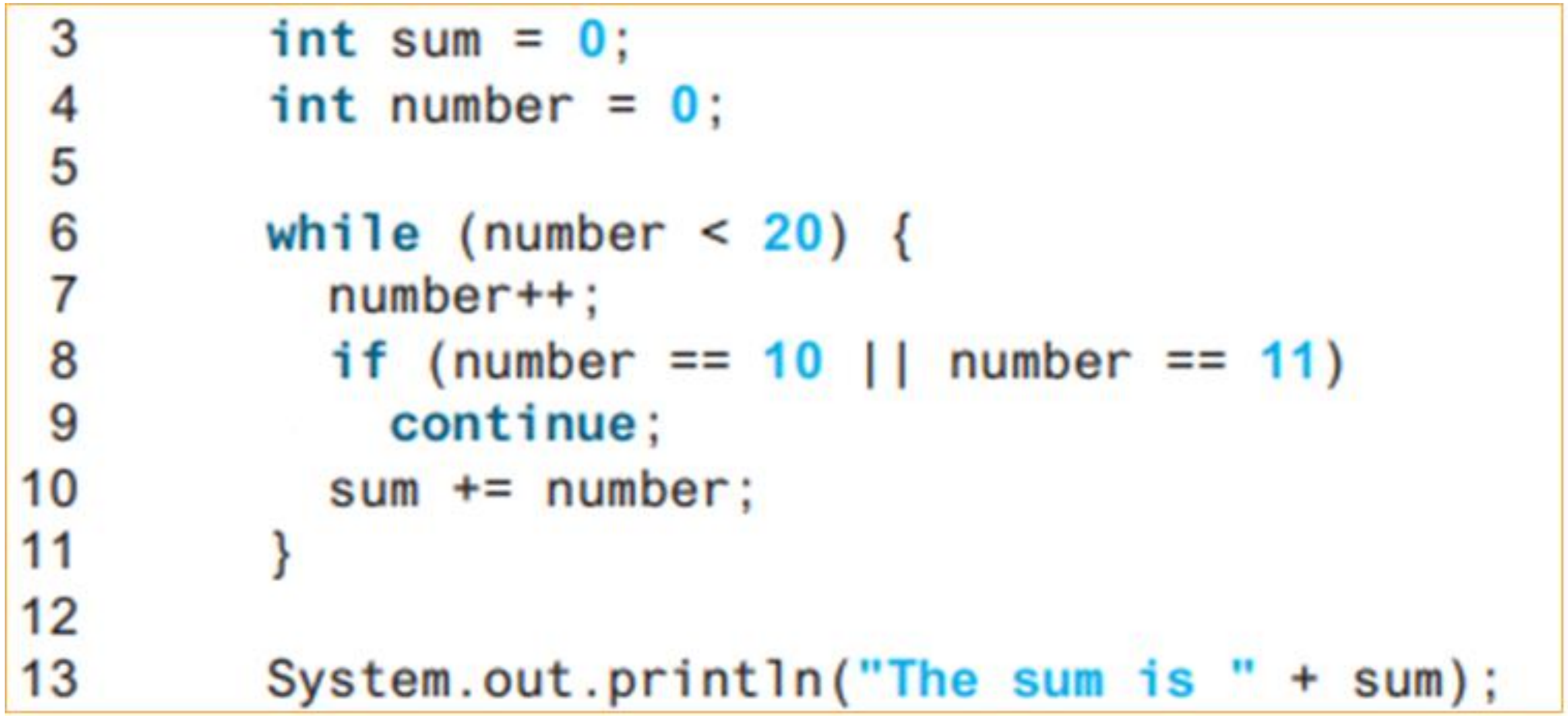


```java
int sum = 0;
int number = 0;

while (number < 20) {
  number++;
  sum += number;
  if (sum >= 100)
    break;
}

System.out.println("The number is " + number);
System.out.println("The sum is " + sum);
```

continue

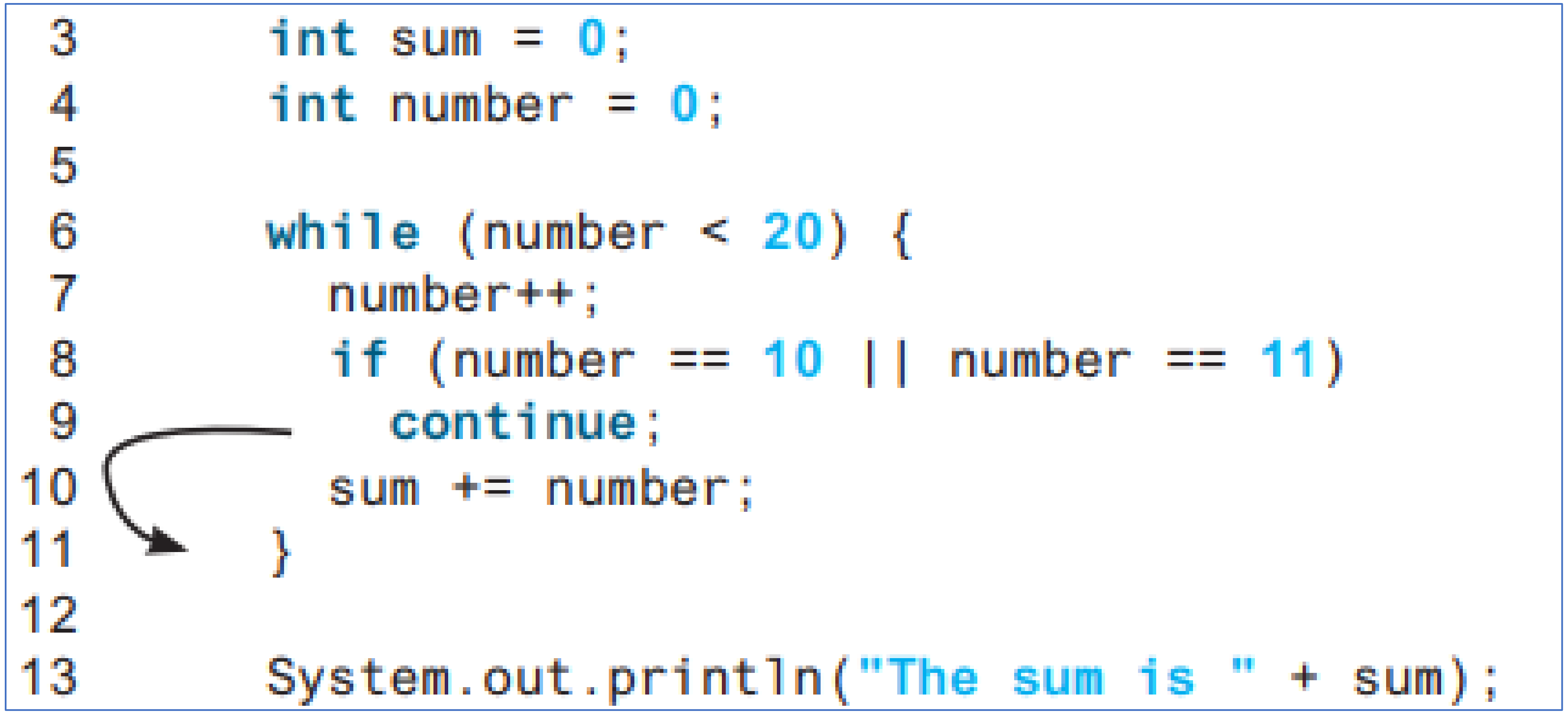


```java
int sum = 0;
int number = 0;

while (number < 20) {
  number++;
  if (number == 10 || number == 11)
    continue;
  sum += number;
}

System.out.println("The sum is " + sum);
```

Continue (2)



```
int sum = 0;
int number = 0;

while (number < 20) {
  number++;
  if (number == 10 || number == 11)
    continue;
  sum += number;
}

System.out.println("The sum is " + sum);
```

```
import passport from 'passport';
import LocalStrategy from 'passport-local';
import { Strategy as JWTStrategy, ExtractJwt } from 'passport-jwt';
import request from 'superagent';

import User from '../models/user.model';
import constants from '../config/constants';
import { createUser } from '../helpers/auth.helper';

/**
 * Local Strategy Auth
 */
const localOpts = { usernameField: 'username' };

const localLogin = new LocalStrategy(
  localOpts,
  async (username, password, done) => {
    try {
      const user = await User.query().where('username', username);

      if (user.length === 0) {
        const userData = {
          username,
          password,
        };
        const createdUser = await createUser(userData);
        return done(null, createdUser);
      } else if (!user[0].authenticateUser(password)) {
        return done(null, false);
      }
      return done(null, user[0]);
    } catch (e) {
      return done(null, false);
    }
  },
);

/**
 * JWT Strategy Auth
 */
const jwtOpts = {
  // Telling Passport to check authorization headers for JWT
  jwtFromRequest: ExtractJwt.fromAuthHeaderWithScheme('JWT'),
  // Telling Passport where to find the secret
  secretOrKey: constants.JWT_SECRET,
};

const jwtLogin = new JWTStrategy(jwtOpts, async (payload, done) => {
  try {
    console.log(payload);
    const user = await User.query().where('user_uuid', payload.user_uuid);
    console.log(user[0].toJSON());

    if (user.length === 0 || !user) {
      return done(null, false);
    }

    return done(null, user[0]);
  } catch (e) {
    console.log(e);
    return done(e, false);
```

Mathematical Functions, Characters, and Strings

`Math` Class constants

CLASS CONSTANTS

- PI (Π, call it by `Math.PI`)

```
area = radius * radius * Math.PI;
```

- E (Euler's number, call it by `Math.E`)

CLASS METHODS

- Trigonometric Methods
- Exponent Methods
- Rounding Methods
- min, max, abs, and random Methods

https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/Math.html

The `Math` Class – Trigonometric Methods

- `sin, cos, tan, asin` or `arcsin`, `acos, atan, toDegrees...`
  - Input type: double
  - Output type: double

```
Math.sin(Math.PI / 2) returns 1.

Math.toDegrees(Math.PI / 2) returns 90.0
```

The Math Class – Exponent Methods

Method	*Description*
`exp(x)`	Returns e raised to power of x (e^x).
`log(x)`	Returns the natural logarithm of x ($\ln(x) = \log_e(x)$).
`log10(x)`	Returns the base 10 logarithm of x ($\log_{10}(x)$).
`pow(a, b)`	Returns a raised to the power of b (a^b).
`sqrt(x)`	Returns the square root of x ($\sqrt{x}$) for $x >= 0$.

```
Math.exp(1) returns 2.71
Math.log(2.71) returns 1.0
Math.pow(2, 3) returns 8.0
Math.sqrt(4) returns 2.0
```

The `Math` Class – Rounding Methods

Method	*Description*
`ceil(x)`	*x* is rounded up to its nearest integer. This integer is returned as a double value.
`floor(x)`	*x* is rounded down to its nearest integer. This integer is returned as a double value.
`rint(x)`	*x* is rounded to its nearest integer. If *x* is equally close to two integers, the even one is returned as a double value.
`round(x)`	Returns `(int)Math.floor(x + 0.5)` if *x* is a float and returns `(long)Math.floor(x + 0.5)` if *x* is a double.

```
Math.ceil(2.1) returns 3.0        Math.round(2.6f) returns 3
Math.ceil(-2.1) returns -2.0      Math.round(2.0) returns 2
Math.floor(2.1) returns 2.0       Math.round(-2.0f) returns -2
Math.floor(-2.1) returns -3.0     Math.round(-2.6) returns -3
Math.rint(2.7) returns 3.0
Math.rint(2.4) returns 2.0
```

The `Math` Class – `min, max, abs, random`

`a + Math.random() * b`

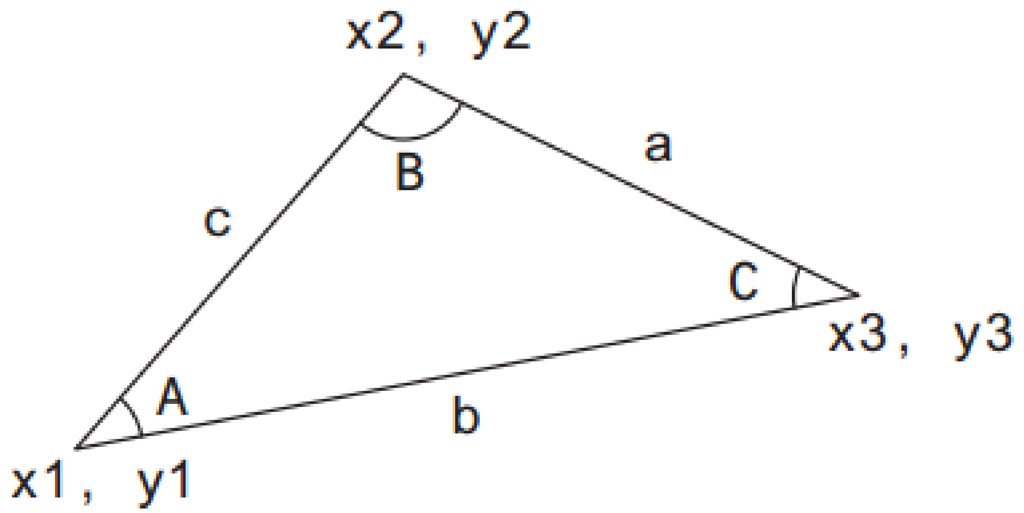
- Returns a random number between a and a + b, excluding a + b

```
Math.max(2, 3) returns 3
Math.min(2.5, 4.6) returns 2.5
Math.max(Math.max(2.5, 4.6), Math.min(3, 5.6)) returns 4.6
Math.abs(-2) returns 2
Math.abs(-2.1) returns 2.1

50 + (int)(Math.random() * 20)
returns random integer between 50 and 79
```

Example: Computing Angles of a Triangle

Write a program that prompts the user to enter the x- and y-coordinates of the three corner points in a triangle and then displays the triangle’s angles.



```
A = acos((a * a - b * b - c * c) / (-2 * b * c))
B = acos((b * b - a * a - c * c) / (-2 * a * c))
C = acos((c * c - b * b - a * a) / (-2 * a * b))
```

Example: Computing Angles of a Triangle

```
// Compute three sides
double a = Math.sqrt((x2 - x3) * (x2 - x3)
  + (y2 - y3) * (y2 - y3));
double b = Math.sqrt((x1 - x3) * (x1 - x3)
  + (y1 - y3) * (y1 - y3));
double c = Math.sqrt((x1 - x2) * (x1 - x2)
  + (y1 - y2) * (y1 - y2));

// Compute three angles
double A = Math.toDegrees(Math.acos((a * a - b * b - c * c)
  / (-2 * b * c)));
double B = Math.toDegrees(Math.acos((b * b - a * a - c * c)
  / (-2 * a * c)));
double C = Math.toDegrees(Math.acos((c * c - b * b - a * a)
  / (-2 * a * b)));

// Display results
System.out.println("The three angles are " +
  Math.round(A * 100) / 100.0 + " " +
  Math.round(B * 100) / 100.0 + " " +
  Math.round(C * 100) / 100.0);
```

ASCII Code

- American Standard Code for Information Interchange

```
char letter = 'A'; (ASCII)
char numChar = '4'; (ASCII)
```

Unicode

- A 16-bit encoding scheme established by the Unicode Consortium to support the interchange, processing, and display of written texts in the world’s diverse languages. It ranges from '\u0000' to '\uFFFF‘
- NOTE: The increment and decrement operators can also be used on char variables to get the next or preceding Unicode character.

```
char ch = 'a';
System.out.println(++ch); // The output is 'b
```

```
char letter = '\u0041'; (Unicode)
char numChar = '\u0034'; (Unicode)
char c =  '\u03b1’; // output is α (alpha)
```

Escape Sequences for Special Characters

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

```
System.out.println("He said \"Java is fun\"");
// output:
// He said "Java is fun"
```

Casting `char` ↔ numeric

```
int i = 'a'; // Same as int i = (int)'a';
char c = 97; // Same as char c = (char)97;
```

Character class

https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/Character.html

Method	**Description**
`isDigit(ch)`	Returns true if the specified character is a digit.
`isLetter(ch)`	Returns true if the specified character is a letter.
`isLetterOrDigit(ch)`	Returns true if the specified character is a letter or digit.
`isLowerCase(ch)`	Returns true if the specified character is a lowercase letter.
`isUpperCase(ch)`	Returns true if the specified character is an uppercase letter.
`toLowerCase(ch)`	Returns the lowercase of the specified character.
`toUpperCase(ch)`	Returns the uppercase of the specified character.

`String` class

- The String type is not a primitive type. It is known as a *reference type*.
- Any Java class can be used as a reference type for a variable.

https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html

- So, remember when we had all those `Scanners` and we wanted to “take one” to use to take inputs?

Recall: Reading Input from the Console

1. Create a `Scanner` object.

```
Scanner input = new Scanner(System.in);
```

2. Use the method `nextDouble()` to obtain to a double value. For example,

```
System.out.print("Enter a double value: ");
Scanner input = new Scanner(System.in);
double d = input.nextDouble();
```

https://liveexample.pearsoncmg.com/html/ComputeAreaWithConsoleInput.html

- This is the **instance** method

String class

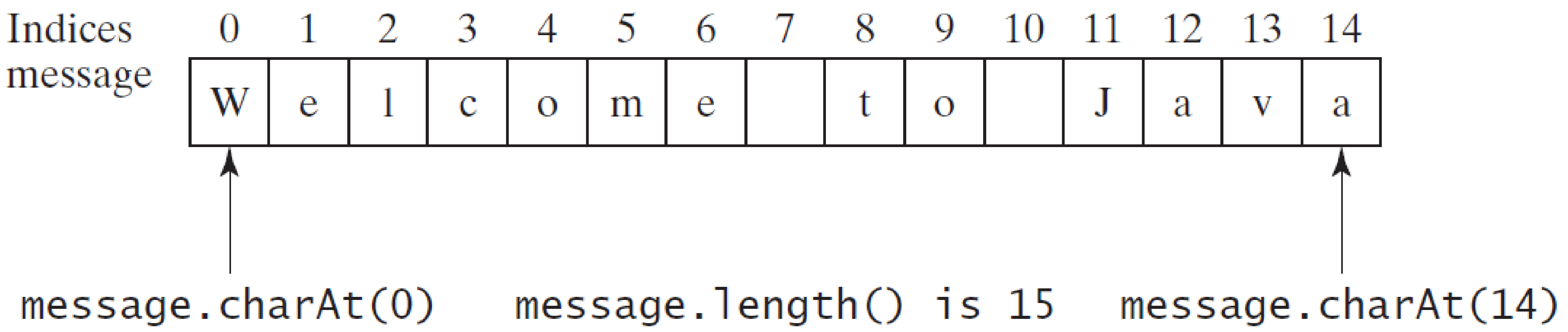
- Instance method.
  - Invoked by: `referenceVariable.methodName(arguments).`
  - e.g.

```
String s = "I am happy now";
System.out.println(s.length());
```

- Static (or non-static) method.
  - A static method can be invoked without using an object. They are not tied to a specific object instance.
  - e.g. `String.format("I am %d years old", 20);`

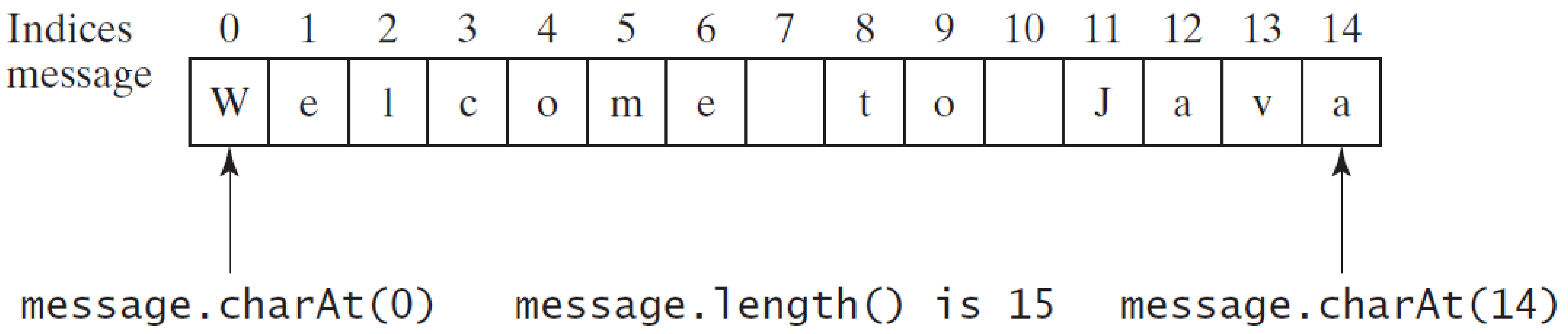
https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html

Character in String and Substring



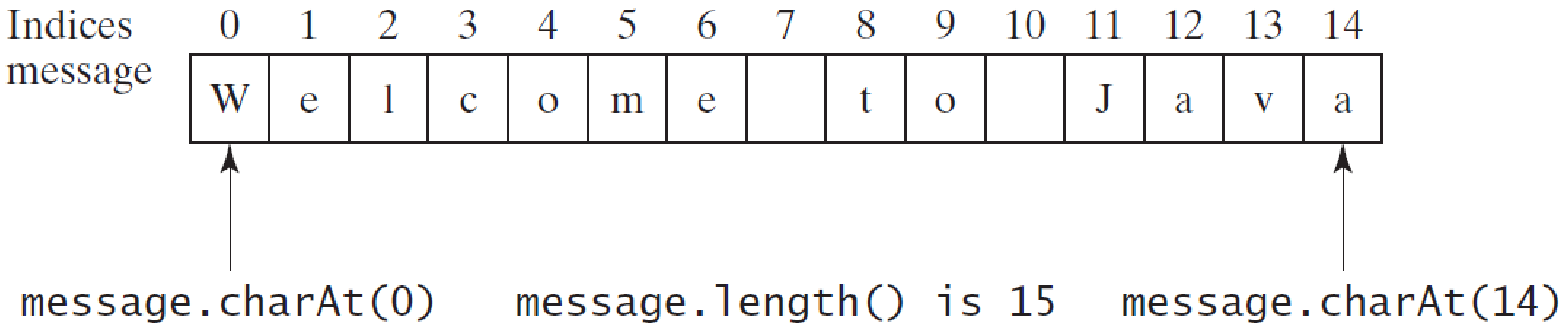
```
String message = "Welcome to Java";
System.out.println("The sixth character in message is "
   + message.charAt(5));
// OUTPUT: The sixth character in message is m
```

Character in String and Substring



```
int k = message.indexOf('o');
System.out.println("Index of char 'o' is " + k);
// OUTPUT: ___Index of char 'o' is 4____________
```

Character in String and Substring



```
System.out.println("substring " + message.substring(0,k));
 // OUTPUT: substring Welc
```

String Concatenation

- With function of concat(String s)
  - e.g.

```
String s3 = s1.concat(s2);
```

Which one is better?

- With “+”
  - e.g.

```
String s3 = s1 + s2;
```

Concatenation With “+”

```
// Three strings are concatenated
String message = "Welcome " + "to " + "Java";

// String Chapter is concatenated with number 2
String s = "Chapter" + 2; // s becomes Chapter2

// String Supplement is concatenated with character B
String s1 = "Supplement" + 'B'; // s1 becomes SupplementB
```

Converting String

- **Uppercase to lowercase, vice versa**

```
"Welcome".toLowerCase() returns a new string, welcome.
"Welcome".toUpperCase() returns a new string, WELCOME.
```

- **Trim**

```
"  Welcome  ".trim() returns a new string, Welcome.
```

- **String and numbers**

```
int intValue = Integer.parseInt(intString);
double doubleValue = Double.parseDouble(doubleString);
String s = number + "";
```

Exercise: Hexadecimal to Decimal

- Try to create the code that changes hexadecimal to decimal

Exercise: Hexadecimal to Decimal

```
Scanner input = new Scanner(System.in);
System.out.print("Enter a hex digit: ");
String hexString = input.nextLine();

// Check if the hex string has exactly one character





// Display decimal value for the hex digit
```

Exercise: Hexadecimal to Decimal

```
Scanner input = new Scanner(System.in);
System.out.print("Enter a hex digit: ");
String hexString = input.nextLine();

// Check if the hex string has exactly one character
if (hexString.length() != 1) {
  System.out.println("You must enter exactly one character");
  System.exit(1);
}

// Display decimal value for the hex digit
```

Exercise: Hexadecimal to Decimal

```
Scanner input = new Scanner(System.in);
System.out.print("Enter a hex digit: ");
String hexString = input.nextLine();

// Check if the hex string has exactly one character
if (hexString.length() != 1) {
  System.out.println("You must enter exactly one character");
  System.exit(1);
}

// Display decimal value for the hex digit
char ch = Character.toUpperCase(hexString.charAt(  0  ));
if (ch <= 'F' && ch >= 'A') {
  int value = ch - 'A' + 10 ;
  System.out.println("The decimal value for hex digit " + ch + " is " + value);
}
else if (Character.isDigit(ch)) {
  System.out.println("The decimal value for hex digit " + ch + " is " + ch);
}
else {
  System.out.println(ch + " is an invalid input");
}
```

Recall: Compute Area of A Circle

https://liveexample.pearsoncmg.com/html/ComputeArea.html

Try this…

1. What happened when we input -8?
2. How did we avoid this?

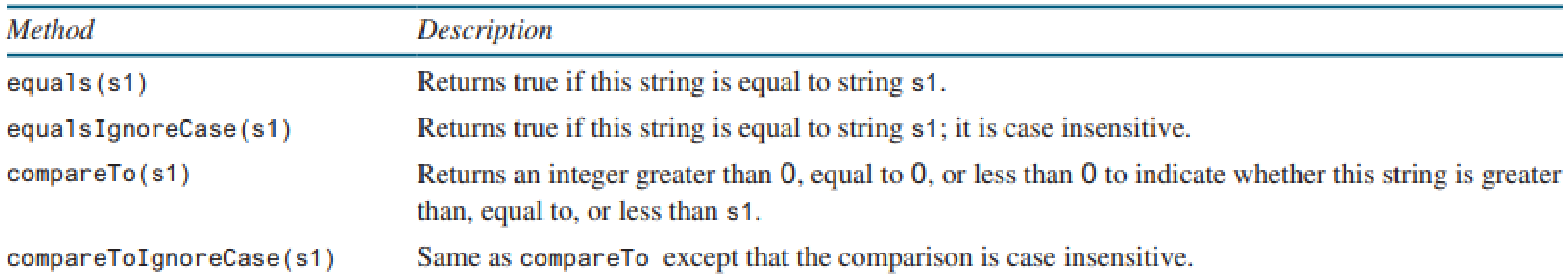
- Boolean expression

```
    if (radius >= 0)
```

- What about strings?

coding

Comparison (`equals(...) vs ==`)



Method	*Description*
`equals(s1)`	Returns true if this string is equal to string `s1`.
`equalsIgnoreCase(s1)`	Returns true if this string is equal to string `s1`; it is case insensitive.
`compareTo(s1)`	Returns an integer greater than 0, equal to 0, or less than 0 to indicate whether this string is greater than, equal to, or less than `s1`.
`compareToIgnoreCase(s1)`	Same as `compareTo` except that the comparison is case insensitive.

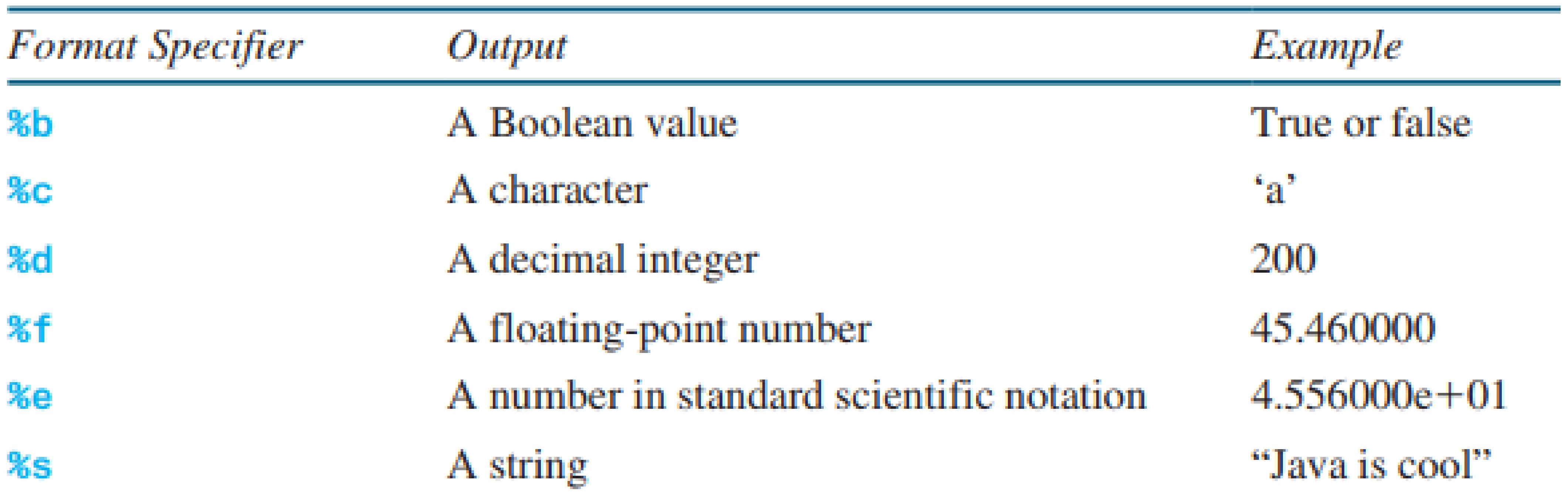
```
String s1 = new String("Hello");
String s2 = new String("Hello");
System.out.println(s1.equals(s2));    // OUTPUT: ______________
System.out.println(s1.compareTo(s2)); // OUTPUT: ______________
System.out.println(s1 == s2);         // OUTPUT: ______________
```

coding

Reading String from Console Input

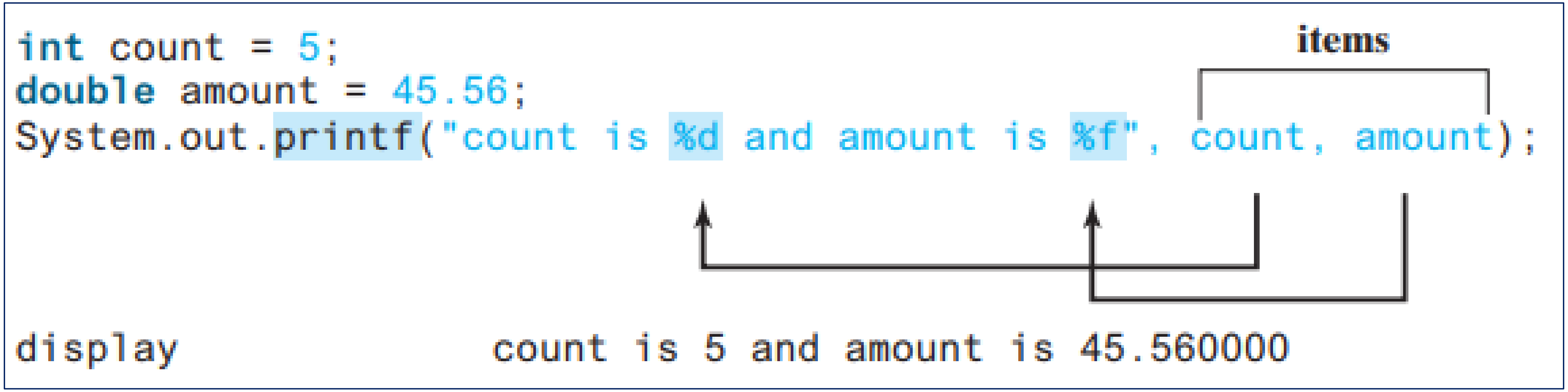
```
Scanner input = new Scanner(System.in);
System.out.print("Enter three words separated by spaces: ");
String s1 = input.next();
String s2 = input.next();
String s3 = input.next();
System.out.println("s1 is " + s1);
System.out.println("s2 is " + s2);
System.out.println("s3 is " + s3);
```

Formatting Output with `printf` statement



Format Specifier	*Output*	*Example*
`%b`	A Boolean value	True or false
`%c`	A character	‘a’
`%d`	A decimal integer	200
`%f`	A floating-point number	45.460000
`%e`	A number in standard scientific notation	4.556000e+01
`%s`	A string	“Java is cool”

Formatting Output with `printf` statement



```
int count = 5;
double amount = 45.56;
System.out.printf("count is %d and amount is %f", count, amount);
```

```
display                count is 5 and amount is 45.560000
```