Java Fundamentals

Slides adapted from Daniel Liang's Introduction to Java Programming and Data Structures – Comprehensive Version book.

Chapter 1

Introduction to Java

Java and Web

- Java can be used to develop standalone applications
- Java can be used to develop applications running from a browser
- Java can also be used to develop applications for hand-held devices
- Java can be used to develop applications for Web servers

History

- James Gosling and Sun Microsystems
- Oak
- 1995

Characteristics of Java

- Java Is Simple
- Java Is Object-Oriented
- Java Is Distributed
- Java Is Interpreted
- Java Is Robust
- Java Is Secure
- Java Is Architecture-Neutral
- Java Is Portable
- Java's Performance
- Java Is Multithreaded
- Java Is Dynamic

Characteristics of Java

- Java Is Simple
- Java Is Object-Oriented
- Java Is Distributed
- Java Is Interpreted
- Java Is Robust
- Java Is Secure
- Java Is Architecture-Neutral
- Java Is Portable
- Java's Performance
- Java Is Multithreaded
- Java Is Dynamic

Java is partially modeled on C++, but greatly simplified and improved. Some people refer to Java as "C++--" because it is like C++ but with more functionality and fewer negative aspects.

Characteristics of Java

- Java Is Simple
- Java Is Object-Oriented
- Java Is Distributed
- Java Is Interpreted
- Java Is Robust
- Java Is Secure
- Java Is Architecture-Neutral
- Java Is Portable
- Java's Performance
- Java Is Multithreaded
- Java Is Dynamic

Java is inherently object-oriented. Although many object-oriented languages began strictly as procedural languages, Java was designed from the start to be object-oriented. Object-oriented programming (O O P) is a popular programming approach that is replacing traditional procedural programming techniques.

One of the central issues in software development is how to reuse code. Object-oriented programming provides great flexibility, modularity, clarity, and reusability through encapsulation, inheritance, and polymorphism.

Characteristics of Java

- Java Is Simple
- Java Is Object-Oriented
- Java Is Distributed
- Java Is Interpreted
- Java Is Robust
- Java Is Secure
- Java Is Architecture-Neutral
- Java Is Portable
- Java's Performance
- Java Is Multithreaded
- Java Is Dynamic

Distributed computing involves several computers working together on a network. Java is designed to make distributed computing easy. Since networking capability is inherently integrated into Java, writing network programs is like sending and receiving data to and from a file.

Characteristics of Java

- Java Is Simple
- Java Is Object-Oriented
- Java Is Distributed
- Java Is Interpreted
- Java Is Robust
- Java Is Secure
- Java Is Architecture-Neutral
- Java Is Portable
- Java's Performance
- Java Is Multithreaded
- Java Is Dynamic

You need an interpreter to run Java programs. The programs are compiled into the Java Virtual Machine code called bytecode. The bytecode is machine-independent and can run on any machine that has a Java interpreter, which is part of the Java Virtual Machine (J V M).

Characteristics of Java

- Java Is Simple
- Java Is Object-Oriented
- Java Is Distributed
- Java Is Interpreted
- Java Is Robust
- Java Is Secure
- Java Is Architecture-Neutral
- Java Is Portable
- Java's Performance
- Java Is Multithreaded
- Java Is Dynamic

Write once, run anywhere
With a Java Virtual Machine (J V M), you can write one program that will run on any platform.

Because Java is architecture neutral, Java programs are portable. They can be run on any platform without being recompiled.

Java Development Kit Editions

- Java Standard Edition (JSE)
  - J2SE can be used to develop client-side standalone applications or applets
  - We use JSE to introduce programming
- Java Enterprise Edition (JEE)
  - J2EE can be used to develop server-side applications such as Java servlets, Java ServerPages, and Java ServerFaces.
- Java Micro Edition (JME).
  - J2ME can be used to develop applications for mobile devices such as cell phones.

Popular Java IDEs

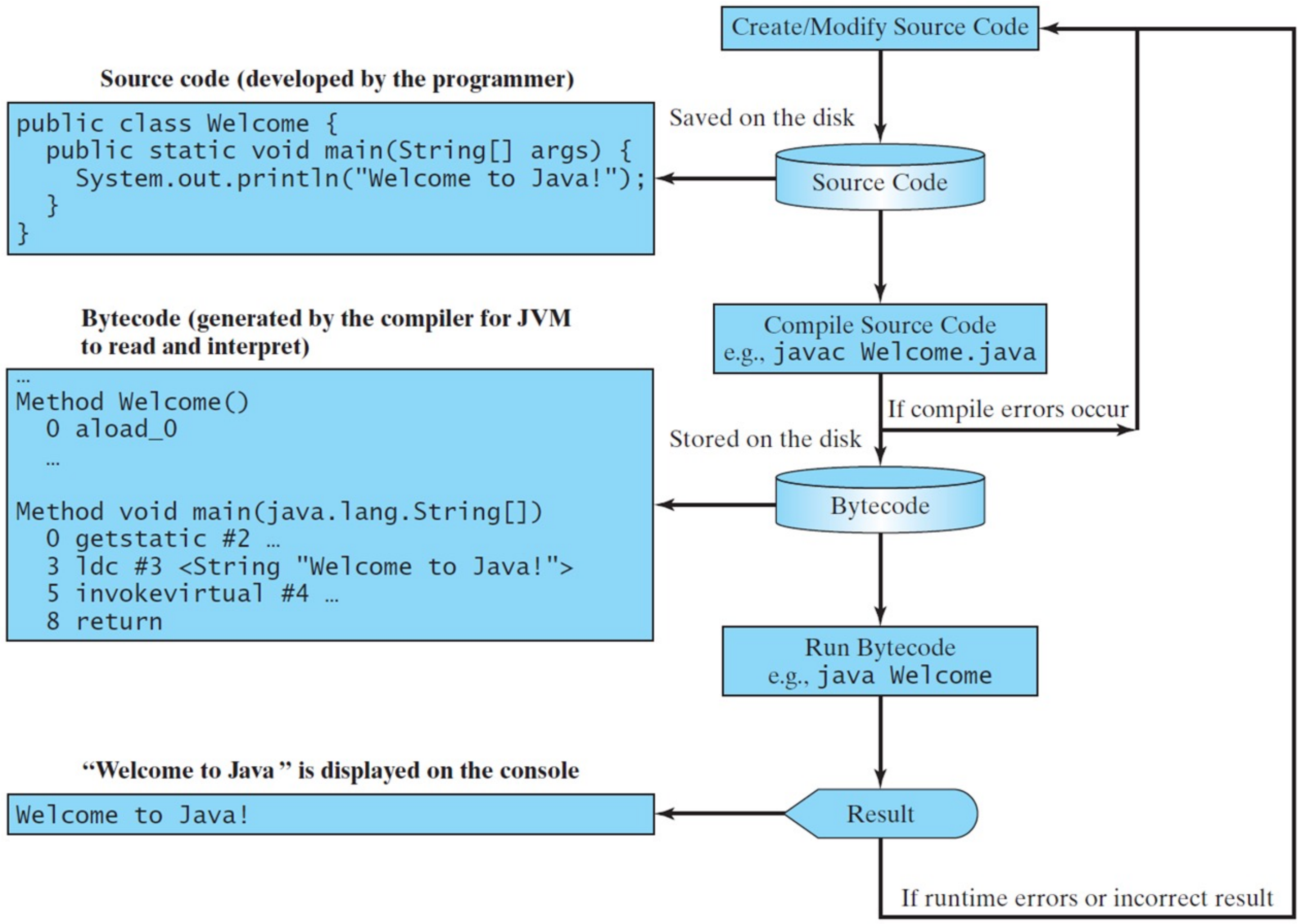
- Eclipse
- IntelliJ
- Netbeans

A Simple Java Program

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

Welcome   Clicking the link displays the source code. You can also run the code in a browser

Creating, Compiling, and Running


Create/Modify Source Code
Saved on the disk
Source Code
Compile Source Code
e.g., javac Welcome.java
If compile errors occur
Stored on the disk
Bytecode
Run Bytecode
e.g., java Welcome
Result
If runtime errors or incorrect result
Source code (developed by the programmer)
Bytecode (generated by the compiler for JVM to read and interpret)
"Welcome to Java" is displayed on the console

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

```
…
Method Welcome()
  0 aload_0
  …

Method void main(java.lang.String[])
  0 getstatic #2 …
  3 ldc #3 <String "Welcome to Java!">
  5 invokevirtual #4 …
  8 return
```

```
Welcome to Java!
```

With Java, you write the program once, and compile the source program into a special type of object code, known as **bytecode**.

The bytecode can then run on any computer with a Java Virtual Machine, as shown below.

Java Virtual Machine is a software that interprets Java bytecode.

Anatomy of a Java Program

- Class name
- Main method
- Statements
- Statement terminator
- Reserved words
- Comments
- Blocks

Class Name

- Every Java program must have at least one class.
- Each class has a name.
- By convention, class names start with an uppercase letter.
- In this example, the class name is Welcome.

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

Main Method

- In order to run a class, the class must contain a method named main.
- The program is executed from the main method.

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

Statement

- A statement represents an action or a sequence of actions
- System.out.println(“Welcome to Java!”) is a statement to display a message

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

Statement Terminator (;)

- Every statement in Java ends with a semicolon (;)

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

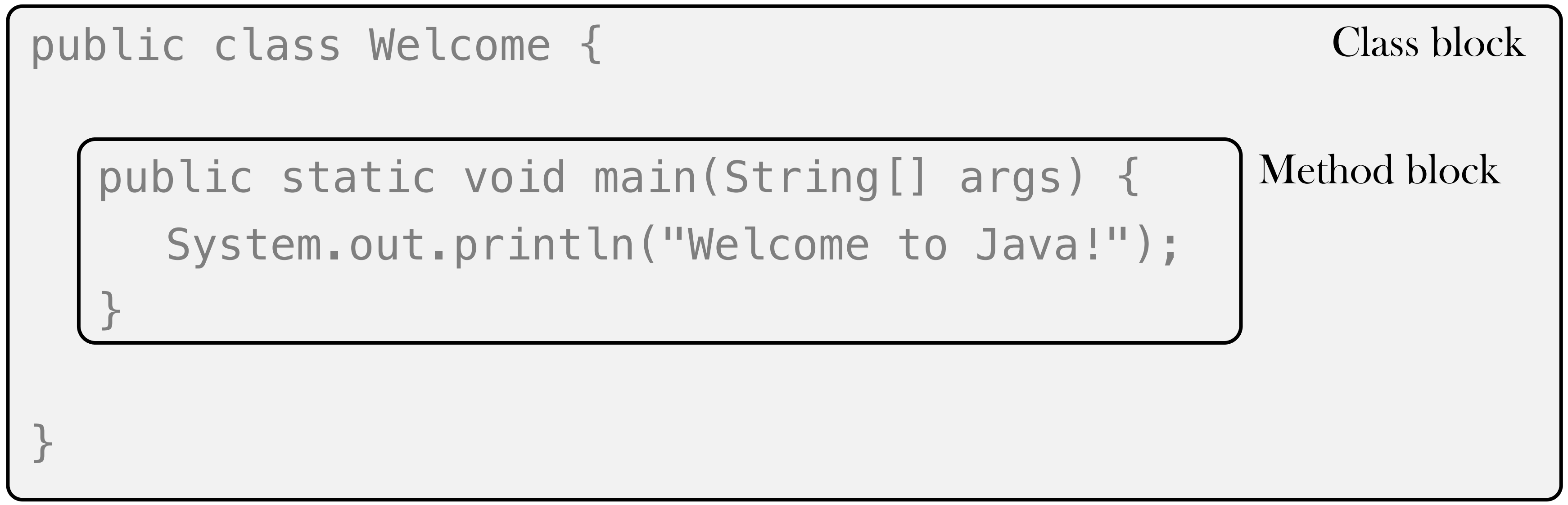
Keywords

- Keywords are words that have a specific meaning to the compiler and cannot be used for other purposes in the program
- For example, when the compiler sees the word **class**, it understands that the word after class is the name for the class

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

Blocks

- A pair of braces in a program forms a block that groups components of a program



```
public class Welcome {

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

}
```

Special Symbols

Character	Name	Description
{ }	Opening and closing braces	Denotes a block to enclose statements.
( )	Opening and closing parentheses	Used with methods.
[ ]	Opening and closing brackets	Denotes an array.
/ /	Double slashes	Precedes a comment line.
" "	Opening/closing quotation marks	Enclosing a string.
;	Semicolon	Marks the end of a statement.

Programming Style and Documentation

- Appropriate Comments
- Naming Conventions
- Proper Indentation and Spacing Lines
- Block Styles

Appropriate Comments

- Include a summary at the beginning of the program to explain what the program does
- Include your name, class section, date at the beginning of the program

- Explain your methods, variables, and classes with comments
- Explain parts of your code with comments, e.g., loops, selections

Naming Conventions

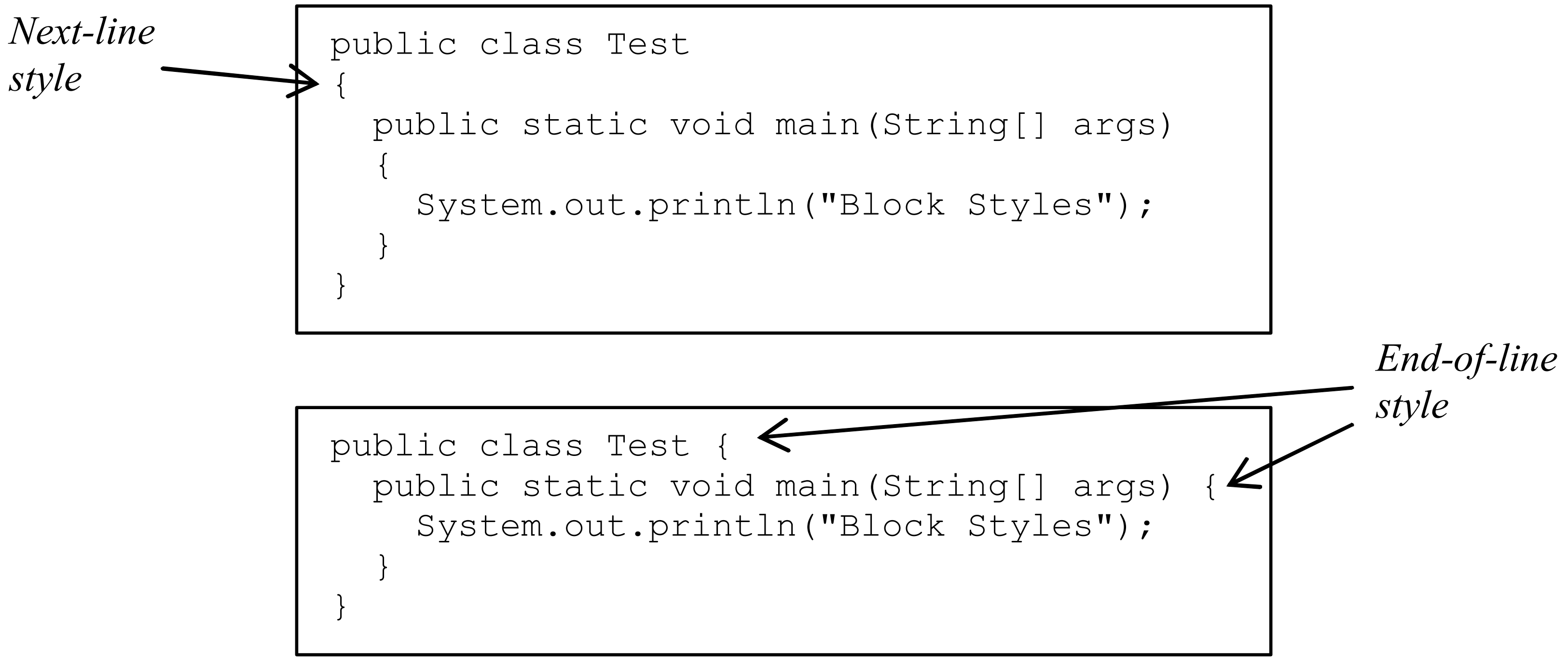
- Choose meaningful and descriptive names for your variables, methods and class names
- Class names: Capitalize the first letter of each word in the name
    - For example, the class name ComputeExpression

Proper Indentation and Spacing

- Indentation
  - Indent two to four spaces.
- Spacing
  - Use blank line to separate segments of the code

Block Styles

- Two block styles: Next-line style and end-of-line style
- Use consistent block styles
- End-of-line style is preferred



```
public class Test
{
  public static void main(String[] args)
  {
    System.out.println("Block Styles");
  }
}
```

```
public class Test {
  public static void main(String[] args) {
    System.out.println("Block Styles");
  }
}
```

Programming Errors

- Syntax Errors
  - Detected by the compiler
- Runtime Errors
  - Causes the program to abort
- Logic Errors
  - Produces incorrect result

Syntax Errors

```
public class Welcome {
   public main(String[] args) {
      Sys.out.prstatic void intln("Welcome to Java!");
   }
}
```

Runtime Errors

```
public class Welcome {
   public static void main(String[] args) {
      System.out.println(1/0);
   }
}
```

Logic Errors

```
public class Welcome {
   public static void main(String[] args) {
      System.out.println(“Celsius 35 in Fahrenheit: ”);
      System.out.println((9/5) * 35 + 32);
   }
}
```

Chapter 2

Elementary Programming

Example

- Computing the area of a circle

```
public class ComputeArea {
  public static void main(String[] args) {
    double radius; // Declare radius
    double area; // Declare area

    // Assign a radius
    radius = 20;

    // Compute area
    area = radius * radius * 3.14159;

    // Display results
    System.out.println(”Circle area of radius " + radius + " is " + area);
  }
}
```

ComputeArea

Reading Input from the Console

```
import java.util.Scanner; // Scanner is in the java.util package

public class ComputeAreaWithConsoleInput {

  public static void main(String[] args) {
    // Create a Scanner object.
    // Scanner object input is used to get input from user
    Scanner input = new Scanner(System.in);

    // Prompt the user
    System.out.print("Enter a number for radius: ");

    // Use the method nextDouble() to obtain to a double value from user
    double radius = input.nextDouble();


    double area = radius * radius * 3.14159;
    System.out.println("The area for the circle of radius " + radius + " is " + area);
  }
}
```

ComputeAreaWithConsoleInput

Implicit Import and Explicit Import

```
import java.util.Scanner; // Explicit import (Recommended)
import java.util.*;       // Implicit import (No performance difference)

public class Welcome {
  public static void main(String[] args) {
  . . .
  }
}
```

Identifiers

- Identifiers are the names that identify the elements such as classes, methods, and variables in a program

- An identifier is a sequence of characters that consist of letters, digits, underscores (_), and dollar signs ($)
- It cannot start with a digit
- An identifier cannot be a reserved word, `true`, `false`, or `null`
- An identifier can be of any length

Variables

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

    double radius; // Declare radius
    double area; // Declare area

    radius = 20;
    area = radius * radius * 3.14159;
    System.out.println(area);

  }
}
```

Declaring Variables

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

    int x;          // Declare x to be an integer variable
    double radius;
    String message;

  }
}
```

Assignment Statements

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

    int x; // Declaring a variable
    double radius;
    String message;

    x = 1; // Assignment statement. Assign the value 1 to x
    radius = 3.0;
    message = “Java”;

  }
}
```

Declaring and Initializing in One Step

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

    int x = 1;
    double radius = 3.0;
    String message = “Java”;

  }
}
```

Constants

- A named constant is an identifier that represents a permanent value
- A constant must be declared and initialized in the same statement.
- The word **final** is a Java keyword for declaring a constant
- By convention, all letters in a constant are in uppercase and use underscores to connect words

```
final double PI = 3.1415;
final int MAX_SIZE = 10;
```

Naming Conventions

- Always choose meaningful and descriptive names for your identifiers
- Variables and method names:
  - 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`
- Class names:
  - 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`

Numerical Data Types

Name	Range	Storage Size
byte	-128 to 127	8-bit
short	-32,768 to 32,767	16-bit
int	-2,147,483,648 to 2,147,483,647	32-bit
long	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	64-bit
float	negative range: -3.4028235E + 38 to -1.4E - 45, positive range: 1.4E - 45 to 3.4028235E + 38	32-bit
double	negative range: -1.7976931348623157E + 308 to -4.9E minus 324 positive range: 4.9E - 324 to 1.7976931348623157E + 308	64-bit

Reading Numbers From the Keyboard

```
import java.util.Scanner; // Scanner class is in the java.util package

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

    // Create a Scanner object named input
    // Scanner object input is used to get input from user
    Scanner input = new Scanner(System.in);

    // Prompt the user
    System.out.print("Enter a number for radius: ");

    // Use the method nextDouble() to obtain to a double value from user
    double radius = input.nextDouble();

  }
}
```

Reading Numbers From the Keyboard

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.

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

Integer Divison

- When both operands of a division are integers, the result of the division is the quotient and the fractional part is truncated
    - 5/2 yields an integer 2

- To perform a floating-point division, one of the operands must be a floating-point number
    - 5.0/2 yields a double value 2.5

Exercise: Displaying Time

```
Enter an integer for seconds: 500
500 seconds is 8 minutes and 20 seconds
```

DisplayTime

Precision of Floating-point Numbers

- Calculations involving floating-point numbers are approximated because these numbers are not stored with complete accuracy
- For example, System.out.println(1.0 - 0.9) displays 0.09999999999998, not 0.1
- Integers are stored precisely. Therefore, calculations with integers yield a precise integer result.

Exponent Operations

- The **Math.pow(a, b)** method can be used to compute a^b
- The **pow** method is defined in the **Math** class in the Java API

```
System.out.println( Math.pow(2, 3) ); // Displays 8.0
System.out.println( Math.pow(4, 0.5) ); // Displays 2.0
```

Number Literals

- A **literal** is a constant value that appears directly in the program
- For example, 34 and 5.0 are literals in the following statements:

```
int i = 34;
double d = 5.0;
```

Integer Literals

- An integer literal can be assigned to an integer variable as long as it can fit into the variable
- A compilation error would occur if the literal were too large for the variable to hold
    - For example, the statement byte b = 1000 would cause a compilation error, because 1000 cannot be stored in a variable of the byte type
- To denote an integer literal of the long type, append it with the letter L or l. L is preferred because l (lowercase L) can easily be confused with 1 (the digit one)

```
long size = 900000L;
```

Floating-Point Literals

- Floating-point literals are written with a decimal point. By default, a floating-point literal is treated as a double type value
- For example, 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

```
float height = 34.57F;
```

Double vs Float

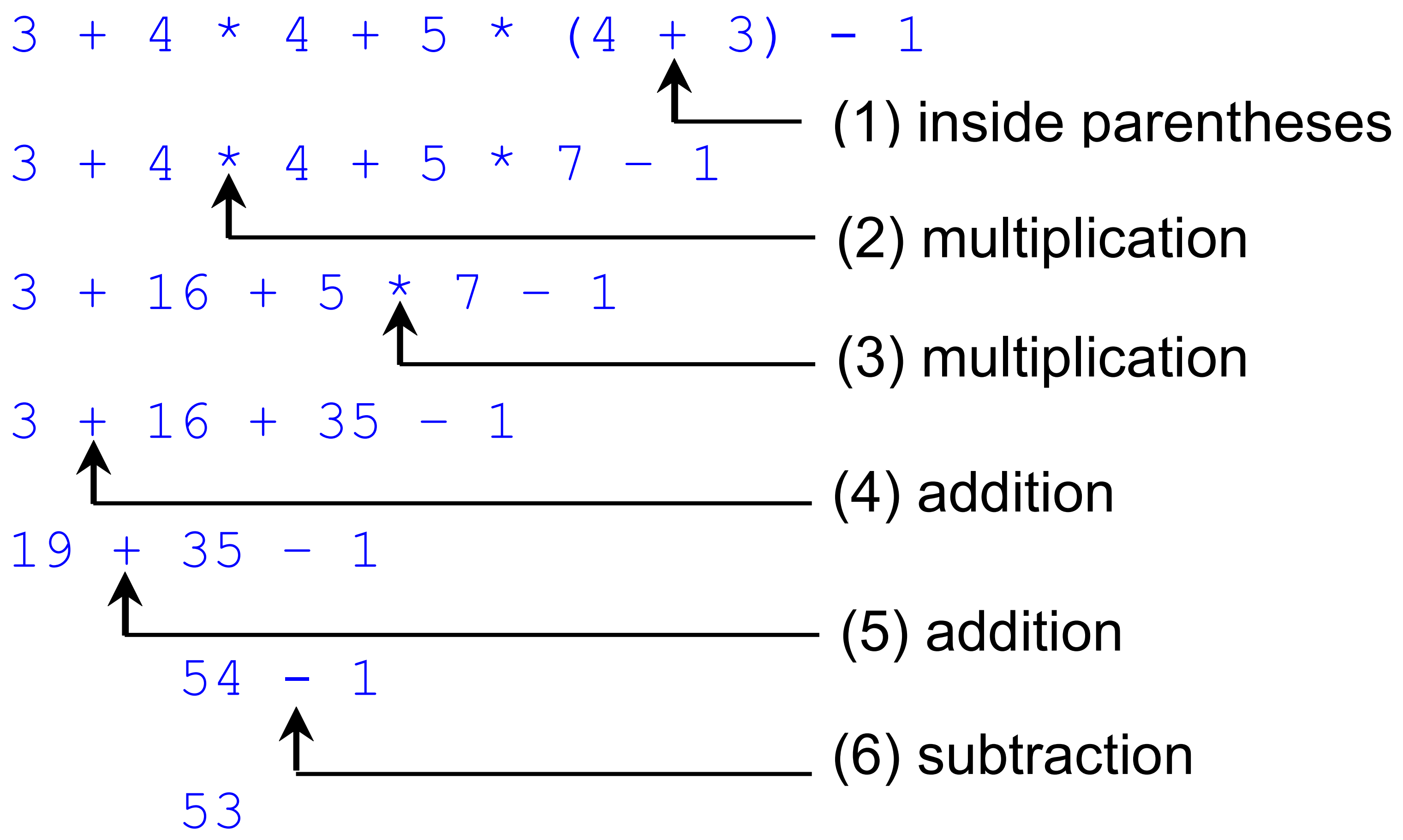
- The double type values are more accurate than the float type values

```
System.out.println(1.0/3.0);   // 0.3333333333333333 (16 digits)
System.out.println(1.0F/3.0F); // 0.33333334 (7 digits)
```

Arithmetic Expressions

$$\frac{3 + 4x}{5} \xrightarrow{\text{is written as}} (3 + 4*x)/5$$

How to Evaluate an Expression



```
3 + 4 * 4 + 5 * (4 + 3) - 1
                         (1) inside parentheses
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
```

Exercise: Converting Temperatures

- Write a program that converts a Fahrenheit degree to Celsius using the formula:

$$Celsius = \frac{5}{9} \times (Fahrenheit - 32)$$

```
double celsius = (5.0/9)*(fahrenhiet-32) // Note the floating-point division
```

FahrenheitToCelsius

Augmented Assignment Operators

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

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

```
int i = 10;
int newNum = 10 * i++;
```

Same effect as

```
int newNum = 10 * i;
i = i + 1;
```

```
int i = 10;
int newNum = 10 * (++i);
```

Same effect as

```
i = i + 1;
int newNum = 10 * i;
```

Increment and Decrement Operators

- Using increment and decrement operators makes expressions short, but it also makes them complex and difficult to read
- Avoid using these operators in expressions that modify multiple variables, or the same variable for multiple times such as this: int k = ++i + i

Type Conversion Rules

When performing a binary operation involving two operands of different types, Java automatically converts the operand based on the following rules:

1. If one of the operands is double, the other is converted into double.
2. Otherwise, if one of the operands is float, the other is converted into float.
3. Otherwise, if one of the operands is long, the other is converted into long.
4. Otherwise, both operands are converted into int.

Type Casting

Implicit casting
double d = 3; (type widening: d becomes 3.0)

Explicit casting
int i = (int)3.0; (type narrowing: i becomes 3)
int i = (int)3.9; (Fraction part is truncated: i becomes 3)

$$\xrightarrow{\texttt{range increases}}$$

`byte, short, int, long, float, double`

Type Casting

What is wrong?

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

Type casting to double

```
System.out.println( (double)1 / 2 ); // Displays 0.5 since 1 is cast to 1.0
```

Casting is necessary if you are assigning a value to a variable of a smaller type range, such as assigning a double value to an int variable.
A compile error will occur if casting is not used in situations of this kind.

Common Errors

- Common Error 1: Undeclared/Uninitialized Variables and Unused Variables
- Common Error 2: Integer Overflow
- Common Error 3: Round-off Errors
- Common Error 4: Unintended Integer Division
- Common Error 5: Redundant Input Objects
- Common Pitfall 1: Redundant Input Objects

Common Error 1

- Common Error 1: Undeclared/Uninitialized Variables and Unused Variables

```
double interestRate = 0.05;
double value = interestrate * 45;
```

Common Error 2

- Common Error 2: Integer Overflow

```
int value = 2147483647 + 1; // value will be -2147483648
```

Common Error 3

- Common Error 3: Round-off Errors

```
System.out.println(1.0-0.9) // Displays 0.09999999999999998
```

Common Error 4

- Common Error 4: Unintended Integer Divison

```
int number1 = 1;
int number2 = 2;
double average = (number1 + number2) / 2;
System.out.println(average);
```

(a)

```
int number1 = 1;
int number2 = 2;
double average = (number1 + number2) / 2.0;
System.out.println(average);
```

(b)

Chapter 3

Selections

The boolean Type

- Boolean data type can store two values: **true** or **false**

```
boolean flag = true;
boolean checkValidity = false;
boolean value = (1>2); // value becomes false
```

Relational Operators

Java Operator	**Explanation**	**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

Exercise: Math Learning Tool

This example creates a program to let a first grader practice additions.

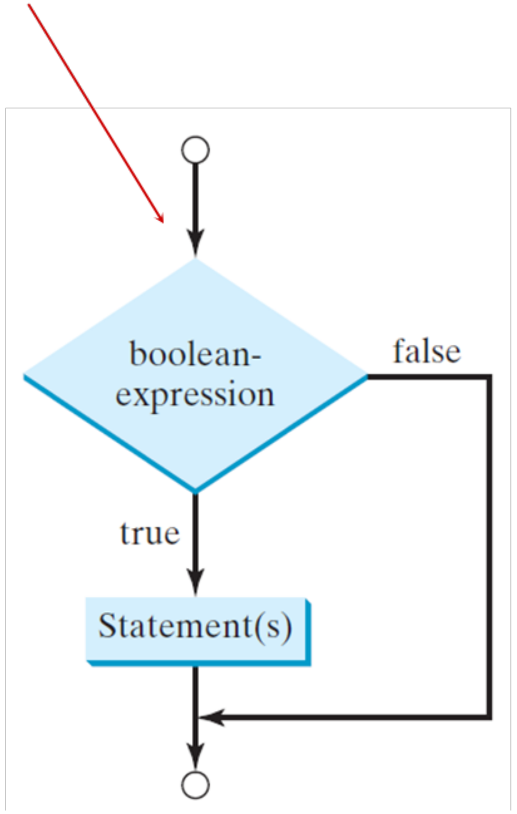
The program randomly generates two single-digit integers number1 and number2 and displays a question such as "What is 7 + 9?" to the student.

After the student types the answer, the program displays a message to indicate whether the answer is true or false.

AdditionQuiz

One-Way If Statements

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



```
if (radius >= 0) {
  area = radius * radius * Math.PI;
}
```

One-Way If Statements

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

(a) Wrong

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

(b) Correct

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

(a)

Equivalent

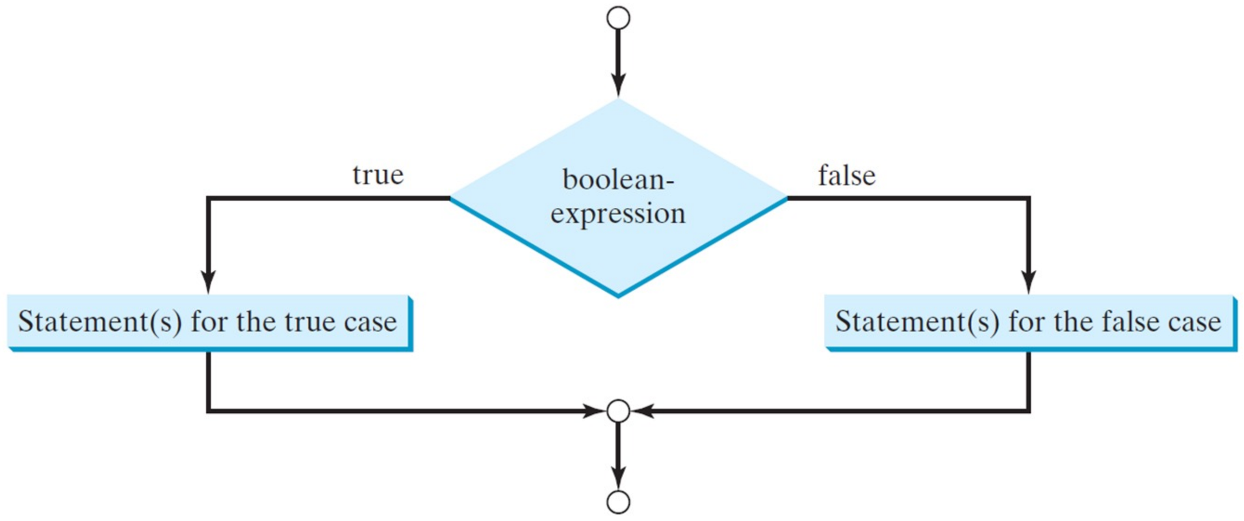
```
if (i > 0)
  System.out.println("i is positive");
```

(b)

if-else Statement

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

```
if (radius >= 0) {
  area = radius * radius * Math.PI;
}
else {
  System.out.println("Incorrect radius");
}
```


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

Multiple if Statements

```java
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

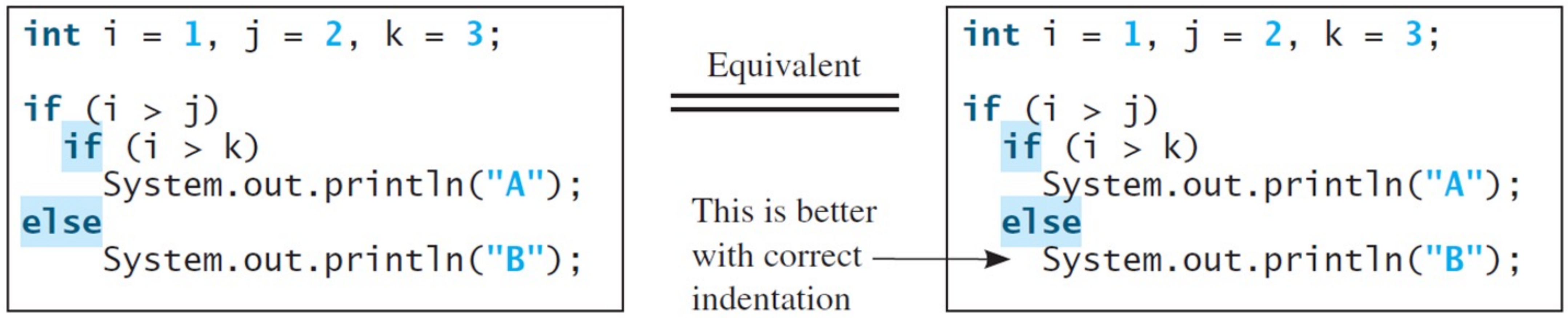
This is better

```java
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)

Note

- The <u>else</u> clause matches the most recent <u>if</u> 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)

Note

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

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

if (i > j) {
  if (i > k)
    System.out.println("A");
}
else
  System.out.println("B"); // This code prints B
```

Common Errors

- Adding a semicolon at the end of an if clause is a common mistake
- This mistake is hard to find, because it is not a compilation error or a runtime error, it is a logic error
- This error often occurs when you use the next-line block style

```
if (radius >= 0);
{
  area = radius * radius * Math.PI;
}
```

Tips

```
if (number % 2 == 0)
  even = true;
else
  even = false;
```

(a)

Equivalent

```
boolean even
  = number % 2 == 0;
```

(b)

```
if (even == true)
  System.out.println(
    "It is even.");
```

(a)

Equivalent

```
if (even)
  System.out.println(
    "It is even.");
```

(b)

Exercise: Body Mass Index

- Body Mass Index (BMI) is a measure of health on weight. It can be calculated by taking your weight in kilograms and dividing by the square of your height in meters. The interpretation of BMI for people 16 years or older is as follows:

BMI	Interpretation
BMI < 18.5	Underweight
18.5 <= BMI < 25.0	Normal
25.0 <= BMI < 30.0	Overweight
30.0 <= BMI	Obese

ComputeAndInterpretBMI

Logical Operators

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

Truth Table for Operator !

p	!p	Example (assume age = 24, weight = 140)
true	false	!(age > 18) is false, because (age > 18) is true.
false	true	!(weight == 150) is true, because (weight == 150) is false.

Truth Table for Operator &&

p_1	p_2	p_1 && p_2	**Example (assume age = 24, weight = 140)**
false	false	false	(age <= 18) && (weight < 140) is false, because both conditions are both false.
false	true	false	
true	false	false	(age > 18) && (weight > 140) is false, because (weight > 140) is false.
true	true	true	(age > 18) && (weight >= 140) is true, because both (age > 18) and (weight >= 140) are true.

Truth Table for Operator ||

p_1	**p_2**	**p_1 || p_2**	**Example (assume age = 24, weight = 140)**
false	false	false	
false	true	true	(age > 34) || (weight <= 140) is true, because (age > 34) is false, but (weight <= 140) is true.
true	false	true	(age > 14) || (weight >= 150) is false, because (age > 14) is true.
true	true	true	

Truth Table for Operator ^

p_1	p_2	p_1 ^ p_2	**Example (assume age = 24, weight = 140)**
false	false	false	(age > 34) ^ (weight > 140) is true, because (age > 34) is false and (weight > 140) is false.
false	true	true	(age > 34) ^ (weight >= 140) is true, because (age > 34) is false but (weight >= 140) is true.
true	false	true	(age > 14) ^ (weight > 140) is true, because (age > 14) is true and (weight > 140) is false.
true	true	false	

Exercise: Determining Leap Year

- This program first prompts the user to enter a year as an int value and checks if it is a leap year.
- A year is a leap year if it is divisible by 4 but not by 100, or it is divisible by 400.

```
(year % 4 == 0 && year % 100 != 0) || (year % 400 == 0)
```

LeapYear

switch Statement

- A **switch** statement executes statements based on the value of a variable or an expression

```
int month = 5;

switch (month) {

case 1:
   System.out.print("January 1, " + year + " is ");
   numberOfDaysInMonth = 31;
   break;

case 3:
   System.out.print("March 1, " + year + " is ");
   numberOfDaysInMonth = 31;
   break;

case 4: numberOfDaysInMonth = 30; break;

case 5: {
   System.out.print("May 1, " + year + " is ");
   numberOfDaysInMonth = 31;
   break;
}

default:
   System.out.println("Invalid month");
}
```

switch Statement

The switch-expression must yield a value of char, byte, short, or int type and must always be enclosed in parentheses.

```
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. The resulting statements in the case statement are executed when the value in the case statement matches the value of the switch-expression. Note that value1, ..., and valueN are constant expressions, meaning that they cannot contain variables in the expression, such as 1 + x.

switch Statement

The keyword <u>break</u> is optional, but it should be used at the end of each case in order to terminate the remainder of the <u>switch</u> statement. If the <u>break</u> statement is not present, the next <u>case</u> statement will be executed.

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

```
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;
}
```

When the value in a **case** statement matches the value of the **switch-expression**, the statements *starting from this case* are executed until either a **break** statement or the end of the **switch** statement is reached.

Conditional Operators

- A conditional operator evaluates an expression based on a condition
- Also called ternary operator

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

// is equivalent to

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

(boolean-expression) ? expression1 : expression2

Conditional Operators

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


// is equivalent to

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

Operator Precedence

- var++, var--
- +, - (Unary plus and minus), ++var,--var
- (type) Casting
- ! (Not)
- *, /, % (Multiplication, division, and remainder)
- +, - (Binary addition and subtraction)
- <, <=, >, >= (Relational operators)
- ==, !=; (Equality)
- ^ (Exclusive OR)
- && (Conditional AND) Short-circuit AND
- || (Conditional OR) Short-circuit OR
- =, +=, -=, *=, /=, %= (Assignment operator

Operator Precedence and Associativity

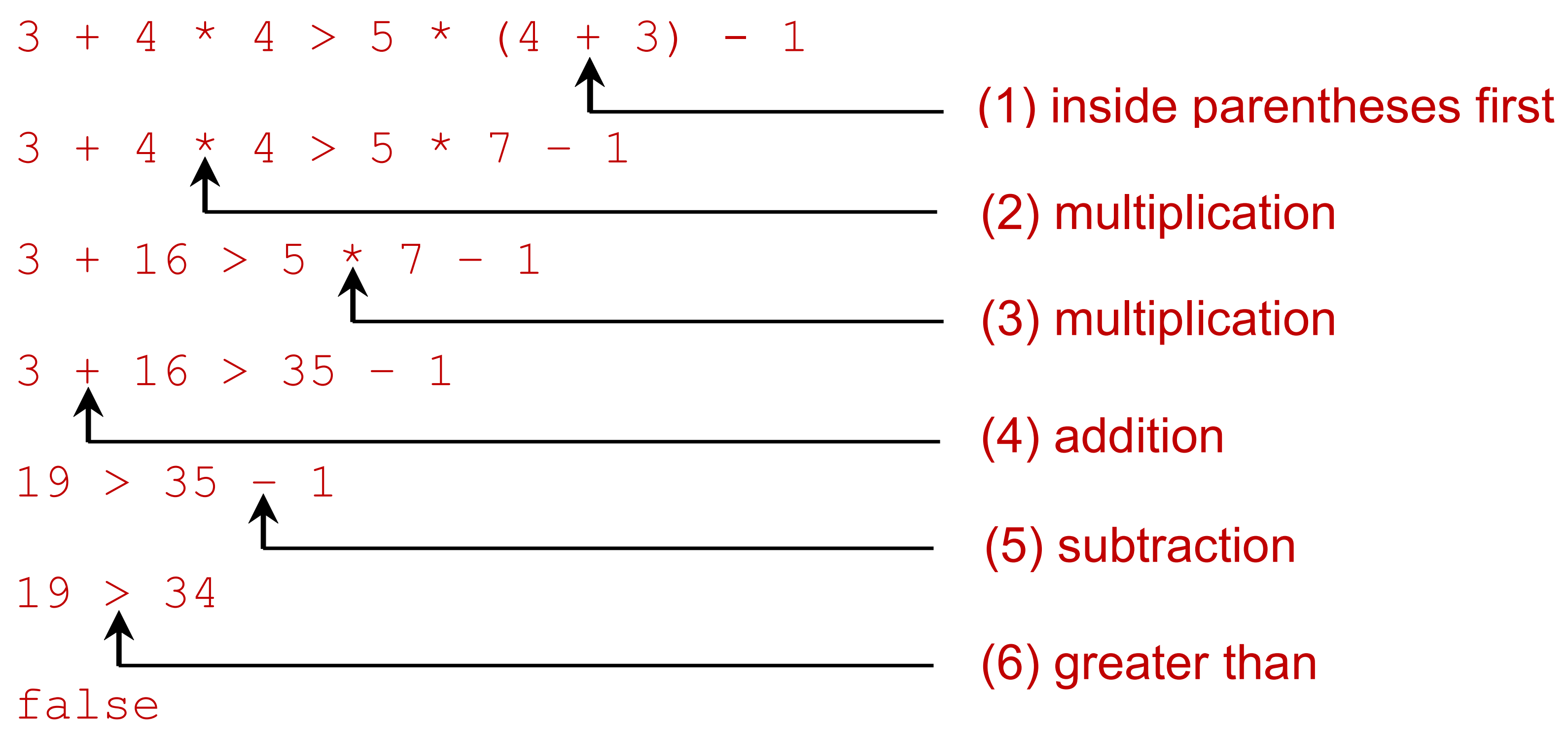
- The expression in the parentheses is evaluated first. (Parentheses can be nested, in which case the expression in the inner parentheses is executed first.)
- When evaluating an expression without parentheses, the operators are applied according to the precedence rule and the associativity rule
- If operators with the same precedence are next to each other, their associativity determines the order of evaluation
- All binary operators except assignment operators are left-associative

Operator Associativity

- When two operators with the same precedence are evaluated, the **associativity** of the operators determines the order of evaluation
- All binary operators except assignment operators are **left-associative**
    - **Example:** a – b + c – d is equivalent to ((a-b)+c)-d
- Assignment operators are **right-associative**
    - Example: a = b += c = 5 is equivalent to a = (b += (c = 5))

Example: Operator Precedence

- Applying the operator precedence and associativity rule, the expression 3 + 4 * 4 > 5 * (4 + 3) - 1 is evaluated as follows



```
3 + 4 * 4 > 5 * (4 + 3) - 1
3 + 4 * 4 > 5 * 7 - 1
3 + 16 > 5 * 7 - 1
3 + 16 > 35 - 1
19 > 35 - 1
19 > 34
false
```

Chapter 5 Loops

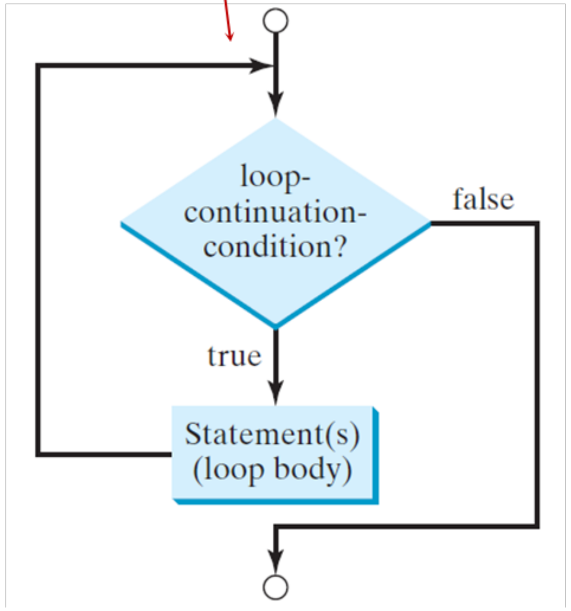
Opening Problem

100
times

```
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!");


...

...

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

while Loops

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



```
int count = 0;

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

Exercise: Guessing Numbers

- Write a program that randomly generates an integer between 0 and 100, inclusive.
- The program prompts the user to enter a number continuously until the number matches the randomly generated number.
- For each user input, the program tells the user whether the input is too low or too high, so the user can choose the next input intelligently

GuessNumber

Exercise: Loop with a Sentinel

- Write a program that reads and calculates the sum of an unspecified number of integers. The input 0 signifies the end of the input
- Often the number of times a loop is executed is not predetermined. You may use an input value to signify the end of the loop. Such a value is known as a **sentinel value**

SentinelValue

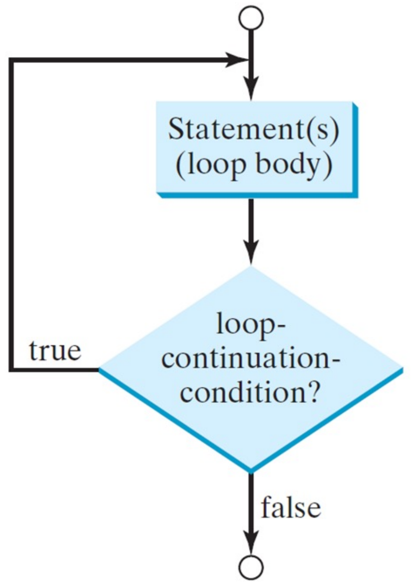
Caution: Floating-point Precision

- Don’t use floating-point values for equality checking in a loop control
- Since floating-point values are approximations for some values, using them could result in imprecise counter values and inaccurate results
- Consider the following code for computing 1 + 0.9 + 0.8 + ... + 0.1:

```java
double item = 1;
double sum = 0;
while (item != 0) { // No guarantee item will be 0
  sum += item;
  item -= 0.1;
}
System.out.println(sum);
```

do-while Loop

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



```
int counter = 1;
do {

  System.out.println(counter + ". Java");
  counter++;

} while (counter <= 10);
```

for Loop

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

```
for (int i=0; i < 100; i++) {
  System.out.println(”Java”);
}
```

for Loop

- The <u>initial-action</u> in a <u>for</u> loop can be a list of zero or more comma-separated expressions.
- The <u>action-after-each-iteration</u> in a <u>for</u> loop can be a list of zero or more comma-separated statements

```
// rarely used examples
for (int i = 1; i < 100; System.out.println(i++));


for (int i = 0, j = 0; (i + j < 10); i++, j++) {
 // Do something
}
```

for Loop

- 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 correct. Nevertheless, it is better to use the equivalent loop in (b) to avoid confusion:

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

(a)

Equivalent

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

(b)

Which loop to use?

- while, do-while, and for are expressively equivalent
- In general, a for loop may be used if the number of repetitions is known
- A while loop may be used if the number of repetitions is not known, as in the case of reading the numbers until the input is 0

Exercise: Nested Loops

- Write a program that uses nested for loops to print a multiplication table

MultiplicationTable

Exercise: Greatest Common Divisor

- Write a program that prompts the user to enter two positive integers and finds their greatest common divisor

GreatestCommonDivisor

break Statement

- You can also use **break** in a loop to immediately terminate the loop
- Loop statements after the **break** are not executed

```java
// print numbers until 5
for (int i = 1; i <= 10; i++) {
  if (i == 6)
    break;
  System.out.println(i);
}
```

continue Statement

- When the **continue** statement is encountered, it ends the current iteration and program control goes to the end of the loop body and the program continues to execute the loop

```
// print odd numbers
for (int i = 1; i <= 10; i++) {
  if (i % 2 == 0)
    continue;
  System.out.println(i);
}
```

Note

- Using **break** and **continue** can simplify programming in some cases. Overusing or improperly using them, however, can make programs difficult to read and debug.

Chapter 4

Mathematical Functions, Characters and Strings

The Math Class

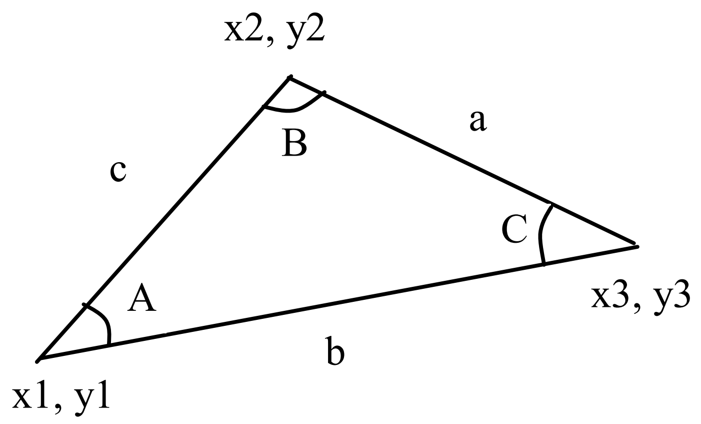
- Java provides many useful methods in the **Math** class for performing common mathematical functions
- Class constant: Math.PI
- Trigonometric Methods
  - Math.sin(degreeRadian), Math.cos(), Math.tan()
- Exponent Methods
  - Math.pow(2,3), Math.sqrt(4), Math.log(4)
- Rounding Methods
  - Math.ceil(), Math.floor(), Math.rint(), Math.round()
- Math.min(a,b), Math.max(a,b), Math.abs(a)

The Math Class

- Math.random() returns a random double value in the range [0.0, 1.0)
  - a + Math.random() * b returns a random number between [a, a+b) (excluding a+b)
  - a + (int) (Math.random() * b) returns an integer

Exercise: 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))
```

ComputeAngles

Character Data Type

```
char letter = ‘A’;
```

The increment and decrement operators can also be used on <u>char</u> variables to get the next or preceding character

```
char ch = ‘a’;
ch = ch + 1;
System.out.println(ch);
```

Casting between char and int

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

Comparing Characters

```
if (ch >= 'A' && ch <= 'Z')
  System.out.println(ch + " is an uppercase letter");
else if (ch >= 'a' && ch <= 'z')
  System.out.println(ch + " is a lowercase letter");
else if (ch >= '0' && ch <= '9')
  System.out.println(ch + " is a numeric character");
```

Methods in the Character Class

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

ASCII Values and Escape Sequences

Escape Sequence	Name
\b	Backspace
\t	Tab
\n	Linefeed
\f	Formfeed
\r	Carriage Return
\\	Backslash
\”	Double Quote

Characters	ASCII Code Value in Decimal
'0' to '9'	48 to 57
'A' to 'Z'	65 to 90
'a' to 'z'	97 to 122

The String Type

The char type only represents one character. To represent a string of characters, use the data type called String. For example,

String message = "Welcome to Java";

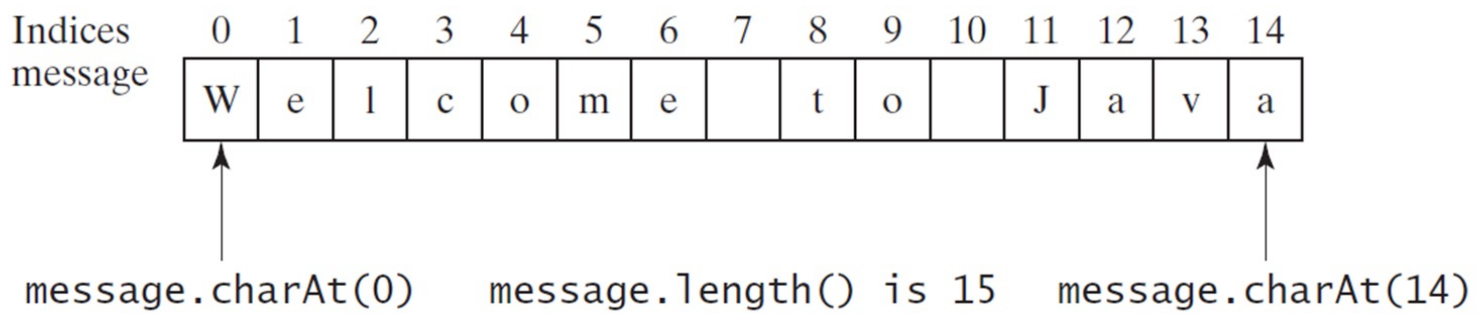
String is actually a predefined class in the Java library just like the System class and Scanner class.

The String type is not a primitive type. It is known as a **reference type**.

Methods for Strings

String message = "Welcome to Java";

Method	**Description**
message.length()	Returns the number of characters in this string.
message.charAt(index)	Returns the character at the specified index from this string.
message.concat(s1)	Returns a new string that concatenates this string with string s1.
message.toUpperCase()	Returns a new string with all letters in uppercase.
message.toLowerCase()	Returns a new string with all letters in lowercase.
message.trim()	Returns a new string with whitespace characters trimmed on both sides.



String Instance Methods

Strings are objects in Java. The methods in the preceding table can only be invoked from a specific string instance.

For this reason, these methods are called **instance methods**. The syntax to invoke an instance method is

referenceVariable.methodName(arguments)

e.g., message.concat(anotherString)

String Concatenation

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

// 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
```

Reading a String from the Console

```
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);
```

Reading a Character from the Console

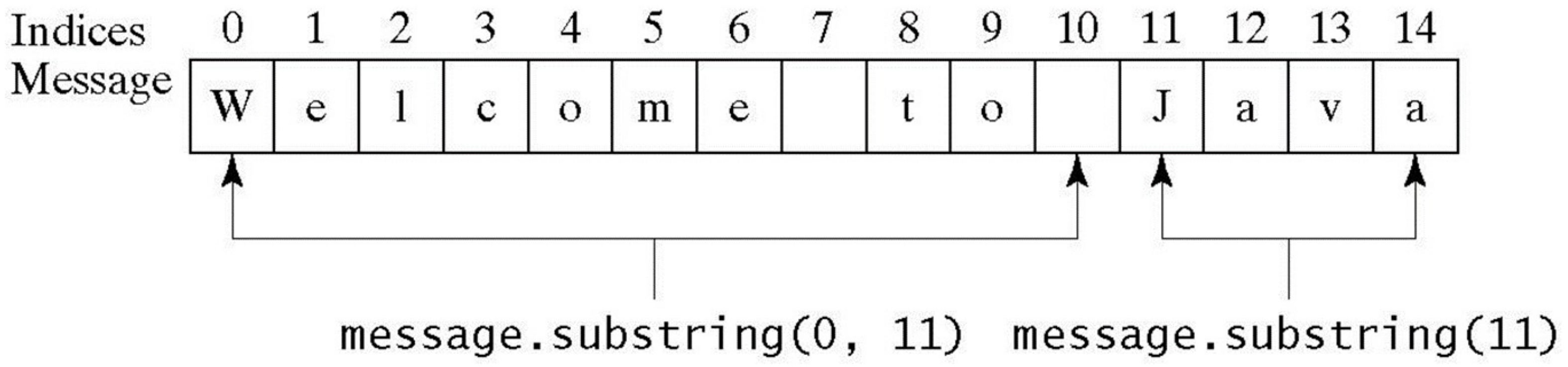
```
Scanner input = new Scanner(System.in);
System.out.print("Enter a character: ");
String s = input.nextLine();
char ch = s.charAt(0);
System.out.println("The character entered is " + ch);
```

Comparing Strings

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.
startsWith(prefix)	Returns true if this string starts with the specified prefix.
endsWith(suffix)	Returns true if this string ends with the specified suffix.

Substrings

Method	Description
substring(beginIndex)	Returns this string's substring that begins with the character at the specified beginIndex and extends to the end of the string, as shown in Figure 4.2.
substring(beginIndex, endIndex)	Returns this string's substring that begins at the specified beginIndex and extends to the character at index endIndex – 1, as shown in Figure 9.6. Note that the character at endIndex is not part of the substring.



Finding a Substring in a String

Method	Description
indexOf(ch)	Returns the index of the first occurrence of ch in the string. Returns -1 if not matched.
indexOf(ch, fromIndex)	Returns the index of the first occurrence of ch after fromIndex in the string. Returns -1 if not matched.
indexOf(s)	Returns the index of the first occurrence of string s in this string. Returns -1 if not matched.
indexOf(s, fromIndex)	Returns the index of the first occurrence of string s in this string after fromIndex. Returns -1 if not matched.
lastIndexOf(ch)	Returns the index of the last occurrence of ch in the string. Returns -1 if not matched.
lastIndexOf(ch, fromIndex)	Returns the index of the last occurrence of ch before fromIndex in this string. Returns -1 if not matched.
lastIndexOf(s)	Returns the index of the last occurrence of string s. Returns -1 if not matched.
lastIndexOf(s, fromIndex)	Returns the index of the last occurrence of string s before fromIndex. Returns -1 if not matched.

Conversion Between Strings and Numbers

Convert string ”12” to integer 12

int intValue = Integer.parseInt(myString);

Convert string ”9.68” to double 9.68

double doubleValue = Double.parseDouble(myString);

String s = number + "";

Formatting Output

Use the printf statement.

```
    System.out.printf(format, items);
```

Where format is a string that may consist of substrings and format specifiers.

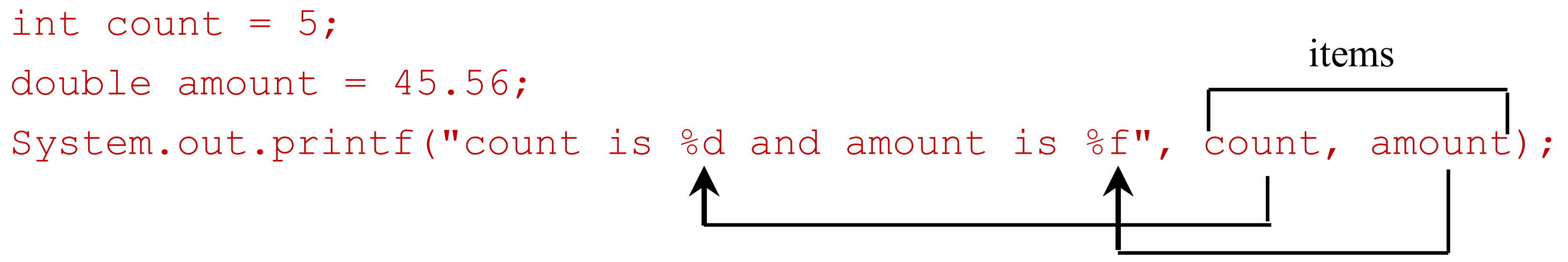
A format specifier specifies how an item should be displayed.

An item may be a numeric value, character, boolean value, or a string.

Each specifier begins with a percent sign

Format Specifiers

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"



```
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
```

FormatDemo