Advanced Programming

Java Basic

ThS. Trần Thị Thanh Nga
Khoa CNTT, Trường ĐH Nông Lâm TPHCM
Email: ngattt@hcmuaf.edu.vn

Assessment

- Attendance + exercise: 20%
- Midterm exam: 30%, closed-book, lab test
- Final exam: 50%, opened-book, lab test

Java programming

- Java was developed by a team led by James Gosling at Sun Microsystems.
- Orignially called Oak, it was designed in 1991 for use in embedded chips in consumer electronic appliances.
- In 1995, renamed Java, it was redesigned for developing Internet applications

Java programming

- Java is:
  - simple, object oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high performance, multithreaded, and dynamic.
- It is employed for
  - Web programming,
  - Standalone applications across platforms on servers, desktops, and mobile devices.

The Java Language Specification, API, JDK, IDE

- Computer languages have strict rules of usage. You need to follow the rules when writing a program, then the computer can understand it.
- **The Java language specification** and **Java API** define the Java standard.
  - **The Java language specification** is a technical definition of the language that includes the syntax and semantics of the Java programming language.
  - The **application program interface** (API) contains predefined classes and interfaces for developing Java programs.
- **The Java language specification** is stable, but the **API** is still expanding.

The Java Language Specification, API, JDK, IDE

- Java is a full-fledged and powerful language that can be used in many ways.
  - **Java Standard Edition** (Java SE): to develop client-side standalone applications or applets.
  - **Java Enterprise Edition** (Java EE): to develop server-side applications, such as Java servlets and Java Server Pages.
  - **Java Micro Edition** (Java ME): to develop applications for mobile devices, such as cell phones.

The Java Language Specification, API, JDK, IDE

- Use Java SE to introduce Java programming in this subject.
- There are many versions of Java SE. Sun releases each version with a Java Development Toolkit (JDK).
- For Java SE 6, the Java Development Toolkit is called JDK 1.6 (also known as Java 6 or JDK 6).

The Java Language Specification, API, JDK, IDE

- Use a Java development tool (e.g., NetBeans, Eclipse) - software that provides an **integrated development environment** (**IDE**) for rapidly developing Java programs.
  - Editing, compiling, building, debugging, and online help are integrated in one graphical user interface.

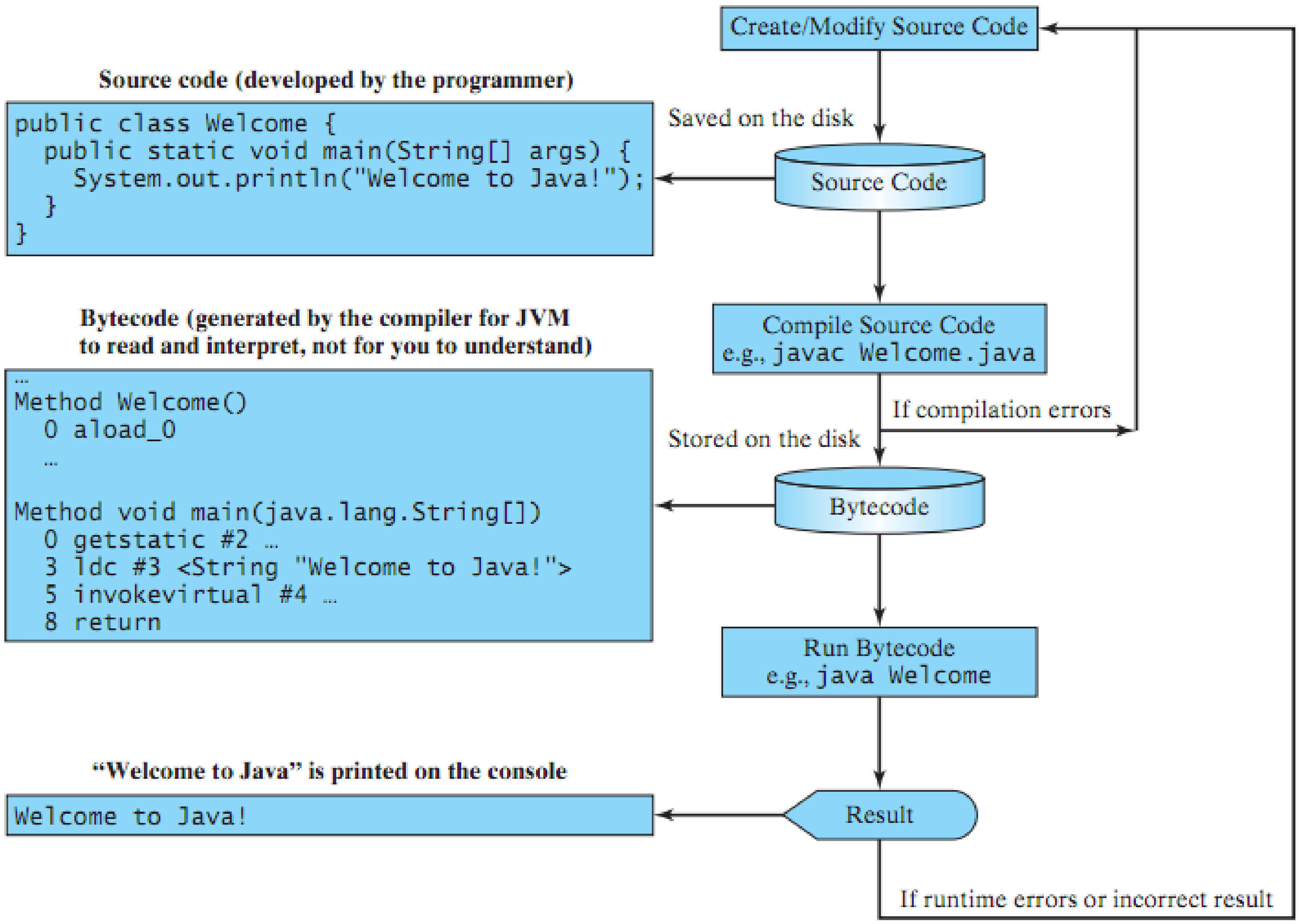
A simple Java program

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

A simple Java program

- Line 1 defines a **class**.
  - Every Java program must have **at least one class**, and class has a **name**.
- Line 2 defines the **main method**.
  - To run a class, the class must contain a method named **main**. The program is executed from the main method.
- A method is a construct that contains statements.
  - **System.out.println**: prints a message "*Welcome to Java!*" to the console (line 4).
- Every statement in Java ends with a semicolon (;).

Creating, Compiling, and Executing



Source code (developed by the programmer)

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

Bytecode (generated by the compiler for JVM to read and interpret, not for you to understand)

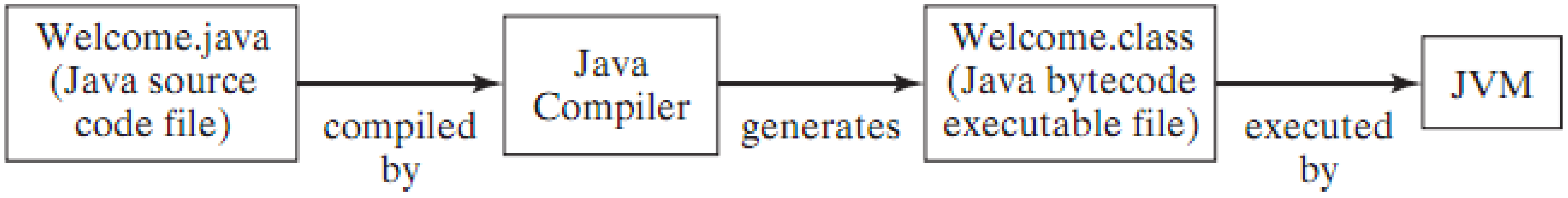
```
…
Method Welcome()
  0 aload_0
  …

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

"Welcome to Java" is printed on the console

```
Welcome to Java!
```

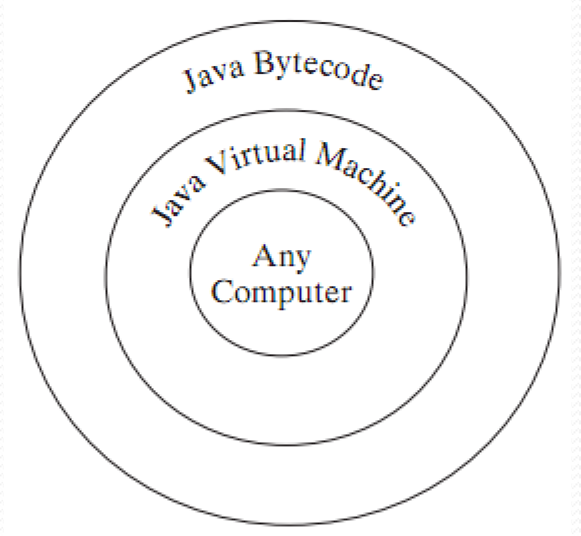
Creating, Compiling, and Executing

- If there are no syntax errors, the compiler generates a bytecode file with a *.class* extension.
- The Java language is a high-level language while Java bytecode is a low-level languag.


Welcome.java (Java source code file)
compiled by
Java Compiler
generates
Welcome.class (Java bytecode executable file)
executed by
JVM

Creating, Compiling, and Executing

- The bytecode is similar to machine instructions and can run on any platform that has a **Java Virtual Machine** (**JVM**).
- The virtual machine is a program that interprets Java bytecode.
- Java bytecode can run on a variety of hardware platforms and operating systems.



Writing Simple Programs

- Writing a program involves **designing algorithms** and **translating algorithms into code**.
  - An algorithm describes *how a problem is solved* in terms of the actions to be executed and the order of their execution.
  - Algorithms can help the programmer *plan a program* before writing it in a programming language.
- Algorithms can be described in **natural languages** or in **pseudocode** (i.e., natural language mixed with programming code).

Writing Simple Programs

1. Read in the radius.
2. Compute the area using the following formula:

$$area = radius * radius * \pi$$

3. Display the area.

Writing Simple Programs

- When you *code*, you translate an algorithm into a program.

```
public class ComputeArea {
      public static void main(String[] args) {
      // Step 1: Read in radius
      // Step 2: Compute area
      // Step 3: Display the area
      }
}
```

Writing Simple Programs

- The program needs to read the *radius* entered from the keyboard.
  - **Reading** the radius.
  - **Storing** the radius.
- To store the **radius**, the program needs to ***declare*** a symbol called a *variable*.
  - A *variable* designates a location in memory for storing data and computational results in the program.
  - A *variable* has a **name** that can be used to **access** the memory location.

Writing Simple Programs

- Using **x** and **y** as variable names?
  - Choose descriptive names: **radius** for *radius*, and **area** for *area*.
- To let the compiler know what **radius** and **area** are, specify their *data types*.
- Variables such as **radius** and **area** correspond to memory locations.
- Every variable has a **name**, a **type**, a **size**, and a **value**.

Writing Simple Programs

```
public class ComputeArea {
   public static void main(String[] args) {
      double radius; // Declare radius
      double area; // Declare area
      // Assign a radius
      radius = 20; // New value is radius
      // Compute area
      area = radius * radius * 3.14159;
      // Display results
      System.out.println("The area for the circle of 
radius " + radius + " is " + area);
   }
}
```

Reading Input from the Console

- Java uses **System.out** to refer to the standard *output device* and **System.in** to the standard *input device*.
  - The output device is the display **monitor**,
  - The input device is the **keyboard**.
- Use the **println** method to display a primitive value or a string to the console.
- Use the **Scanner** class to create an object to read input from **System.in**:

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

Reading Input from the Console

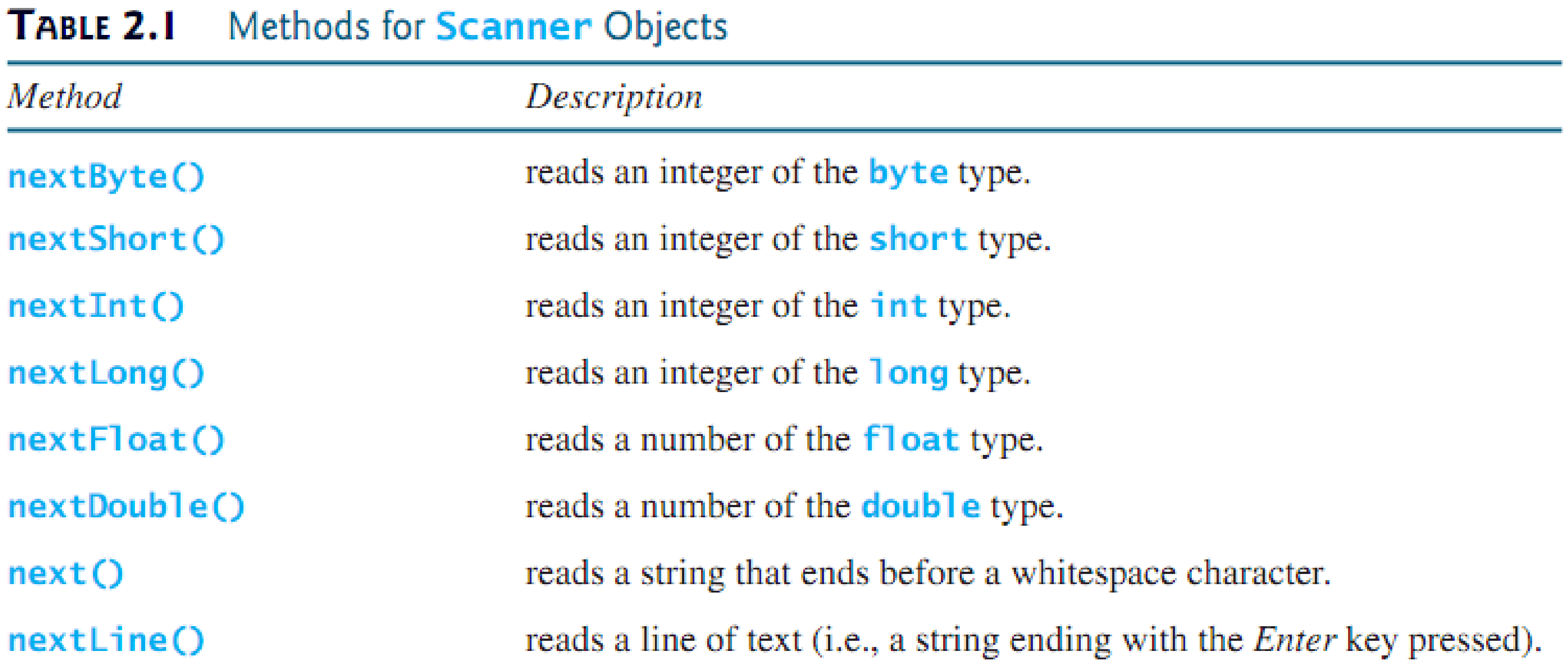


TABLE 2.1 Methods for `Scanner` Objects

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.
`next()`	reads a string that ends before a whitespace character.
`nextLine()`	reads a line of text (i.e., a string ending with the *Enter* key pressed).

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 input = new Scanner(System.in);
       // Prompt the user to enter a radius
       System.out.print("Enter a number for radius: ");
       double radius = input.nextDouble();
       // Compute area
       double area = radius * radius * 3.14159;
       // Display result
       System.out.println("The area for the circle of radius "
                                 + radius + " is " + area);
   }
}
```

Finger Exercise

- Reading 3 numbers from the keyboard, and displays their average.

Finger Exercise

```java
import java.util.Scanner; // Scanner is in the java.util package
public class ComputeAverage {
   public static void main(String[] args) {
        Scanner input = new Scanner(System.in);//Create a Scanner object
        // Prompt the user to enter three numbers
        System.out.print("Enter three numbers: ");
        double number1 = input.nextDouble();
        double number2 = input.nextDouble();
        double number3 = input.nextDouble();
        // Compute average
        double average = (number1 + number2 + number3) / 3;
        // Display result
        System.out.println("The average of " + number1 + " " + number2 + " 
                                    "+ number3 + " is " + average);
   }
}
```

Identifiers

- **ComputeAverage**, **main**, **input**, **number1**, **number2**, **number3**,… are called identifiers.
- All identifiers must obey the following rules:
  - An identifier is a sequence of characters that consists of letters, digits, underscores (_), and dollar signs ($).
  - An identifier must start with a letter, an underscore (_), or a dollar sign ($). It cannot start with a digit.
  - An identifier cannot be a reserved word.
  - An identifier cannot be **true**, **false**, or **null**.
  - An identifier can be of any length.

Identifiers

- Java is **case sensitive**, **area**, **Area**, and **AREA** are all different identifiers.
- **Identifiers** are for naming *variables*, *constants*, *methods*, *classes*, and *packages*.
  - Descriptive identifiers make programs easy to read.
  - Do not name identifiers with the $ character.
    - The $ character should be used only in mechanically generated source code.

Reserved Words

- Literals
  - `null true false`
- Keywords
  - `abstract assert boolean break byte case catch char class continue default do double else extends final finally float for if implements import instanceof int interface long native new package private protected public return short static strictfp super switch synchronized this throw throws transient try void volatile while`
- Reserved for future use
  - `byvalue cast const future generic goto inner operator outer rest var volatile`

Variables

- **Variables** are used to *store* values to be used later in a program.
  - They are called **variables** because their values *can be changed*.
  - Example: You can assign any numerical value to **radius** and **area**, and the values of **radius** and **area** can be reassigned.
- **Variables** are for representing data of a certain **type**.
- To use a **variable**, you *declare* it by telling the compiler its *name* as well as what *type* of data it can store.

Variables

- The **variable** declaration tells the compiler to allocate appropriate memory space for the variable based on its data type.
- Syntax:

```
datatype variableName;
```

- Examples :

```
int count;
double radius;
double interestRate;
```

Declaring Variables

- The data types **int**, **double**, **char**, **byte**, **short**, **long**, **float**, **char**, and **boolean**.
- If **variables** are of the *same type*, they can be declared together, they are separated by commas:

  datatype variable1, variable2, ..., variableN;

- For example:

  int i, j, k;

Naming Variables

- By convention, **variable names** are in *lowercase*.
- If a name consists of several words, concatenate all of them and *capitalize the first letter* of each word ***except the first***.
- Examples: **radius** and **interestRate**.

Initializing variables

- **Variables** often have initial values.
- Declare a **variable** and initialize it in one step: **int** count = 1;
- The next two statements are same: **int** count; count = 1;
- You can also use a shorthand form to declare and initialize variables of the same type together.

  int i = 1, j = 2;
- TIP:
  - A **variable** declared in a method must be assigned a value before it can be used.
  - You should declare a **variable** and assign its initial value in one step ➔ make the program easy to read and avoid programming errors.

Assignment Statements

- You can assign a **value** to it by using an *assignment statement*.

  variable = expression;

Assignment Expressions

- An **expression** represents a computation involving *values*, *variables*, and *operators* that, taking them together, evaluates to a value.

```
int x = 1;
double radius = 1.0;
x = 5 * (3 / 2) + 3 * 2;
x = y + 1;
area = radius * radius * 3.14159;
```

- To assign a value to a **variable**, the *variable name* must be on the *left* of the *assignment operator*:

  1 = x ➔ Right or wrong?

Assignment Expressions

- An **assignment statement** is also known as *an assignment expression*.
- Example:

1. System.out.println(x = 1);

which is equivalent to: x = 1; System.out.println(x);

2. i = j = k = 1;

which is equivalent to: k = 1; j = k; i = j;

Named Constants

- The value of a **variable** may *change* during the execution of a program, but a named constant or simply constant represents permanent data that never changes.
  - In **ComputeArea** program, π is a **constant**. If you use it frequently, you don't want to keep typing 3.14159 ➔ declare a constant for π
- **Syntax**:

  final datatype CONSTANT_NAME = VALUE;
- By convention, constants are named in **uppercase**: **PI**, not **pi** or **Pi**.

Named Constants

- There are three benefits of using constants:
  - (1) you don’t have to repeatedly type the same value;
  - (2) if you have to change the constant value (e.g., from 3.14 to 3.14159 for PI), you need to change it only in a single location in the source code;
  - (3) a descriptive name for a constant makes the program easy to read.

Numeric Data Types and Operations

- Every **data type** has a *range* of values.
- The compiler allocates memory space for each variable or constant according to its **data type**.
- Java provides eight primitive data types for *numeric* values, *characters*, and *Boolean* values.

Primitives: Integers

- Signed whole numbers
- Initialized to zero

Categories:

a. integer

b. floating

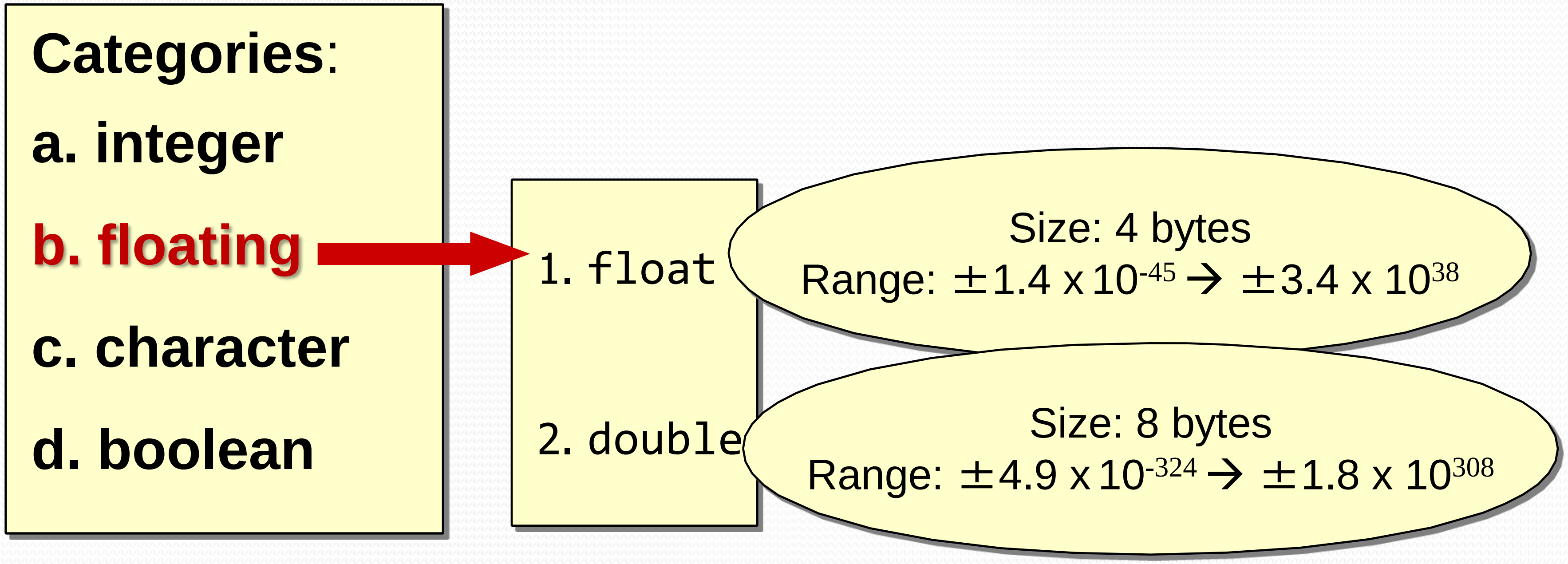
c. character

d. boolean

1. `byte`	Size: 1 byte	Range: $-2^{7} \rightarrow 2^{7} - 1$
2. `short`	Size: 2 bytes	Range: $-2^{15} \rightarrow 2^{15} - 1$
3. `int`	Size: 4 bytes	Range: $-2^{31} \rightarrow 2^{31} - 1$
4. `long`	Size: 8 bytes	Range: $-2^{63} \rightarrow 2^{63} - 1$

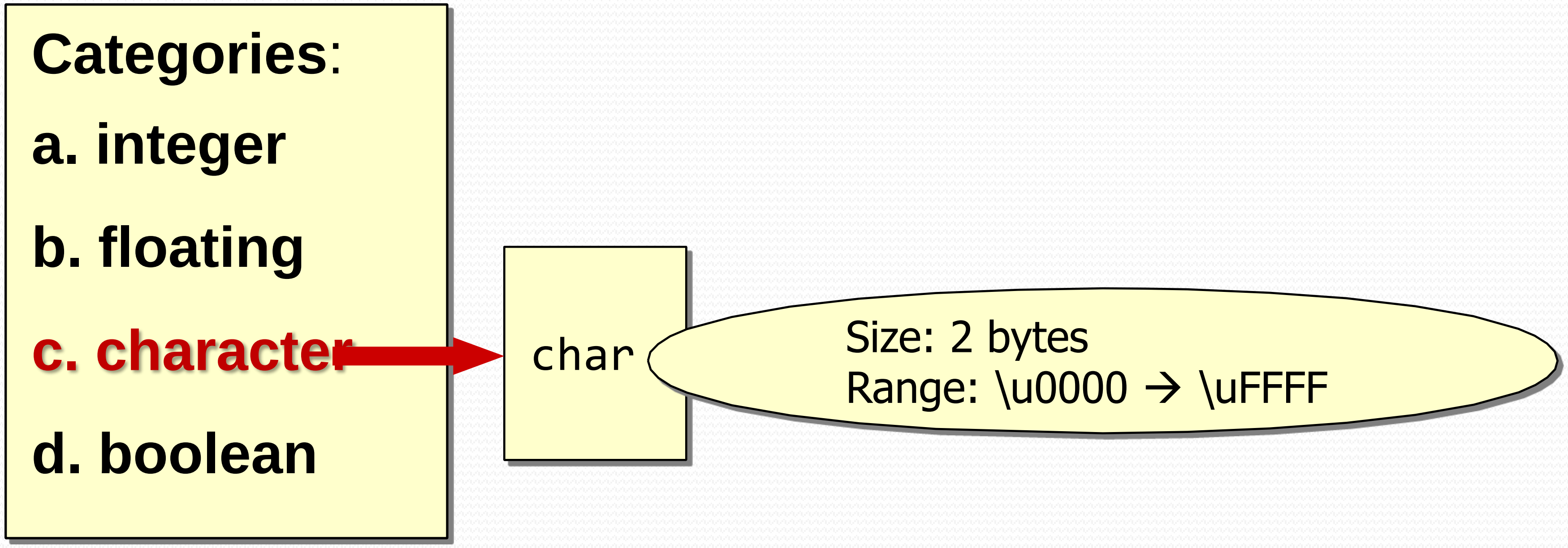
Primitives: Floating Points

- "General" numbers
  - Can have fractional parts
- Initialized to zero


Categories:
a. integer
b. floating
c. character
d. boolean
1. `float`
2. `double`
Size: 4 bytes
Range: $\pm 1.4 \times 10^{-45} \rightarrow \pm 3.4 \times 10^{38}$
Size: 8 bytes
Range: $\pm 4.9 \times 10^{-324} \rightarrow \pm 1.8 \times 10^{308}$

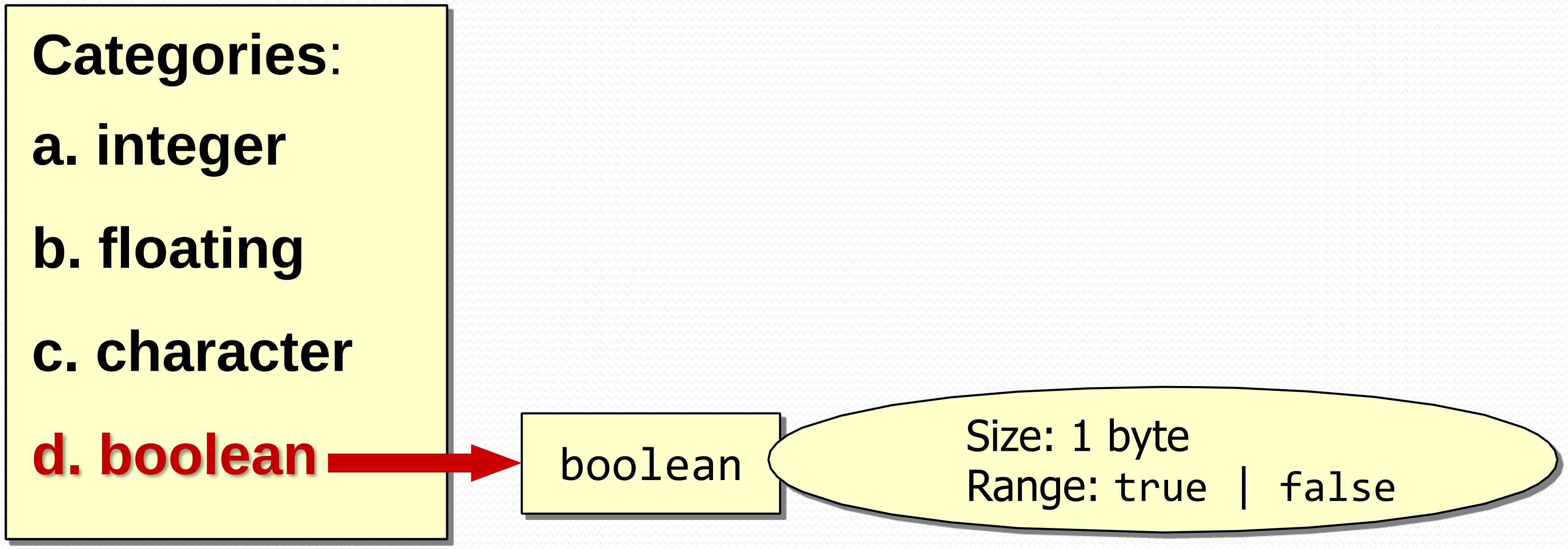
Primitives: Characters

- Char is any unsigned Unicode character
- Initialized to zero (\u0000)



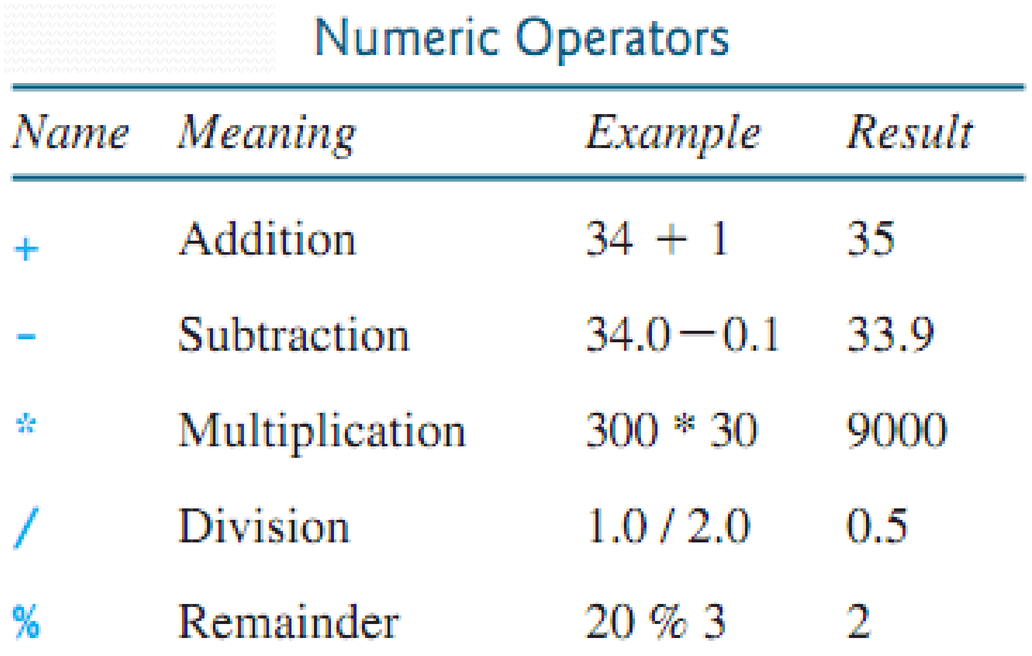
Primitives: Booleans

- `boolean` values are distinct in Java
  - Can only have a `true` or `false` value
  - An int value can NOT be used in place of a boolean
- Initialized to `false`



Numeric Operators

- The **operators** for numeric data types include the standard arithmetic operators: addition (+), subtraction (–), multiplication (*), division (/), and remainder (%).



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

Numeric Literals

- A **literal** is a constant value that appears directly in a program.

  int numberOfYears = 34;

  double weight = 0.305;

- Integer Literals:
  - An **integer literal** is assumed to be of the **int** type, whose value is between – 2147483648 và 2147483647
  - To denote an **integer literal** of the **long** type, append the letter L or l to it (e.g., 2147483648L).

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: 5.0 is considered a double value.
  - 100.2f or 100.2F
  - 100.2d or 100.2D.
- Scientific Notation
  - Floating-point literals can also be specified in scientific notation
    - **1.23456e+2**, the same as **1.23456e2**,
    - **1.23456e-2** $=1.23456 * 10^{-2} = 0.0123456$.
  - **E** (or **e**) represents an exponent

Finger Exercise

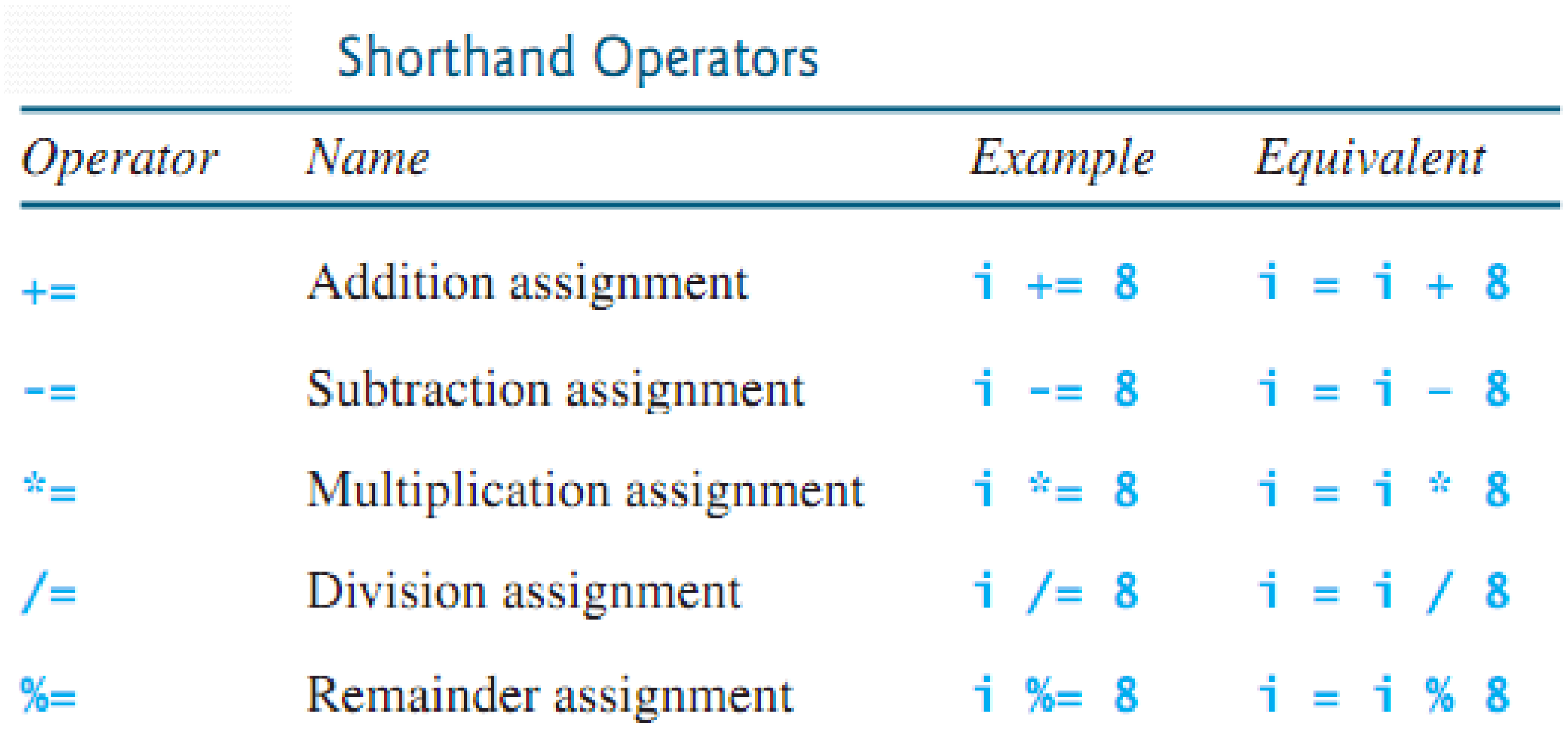
- Converts a Fahrenheit degree to Celsius using the formula

  celsius = (5/9)* (fahrenheit – 32).

Fahrenheit To Celsius

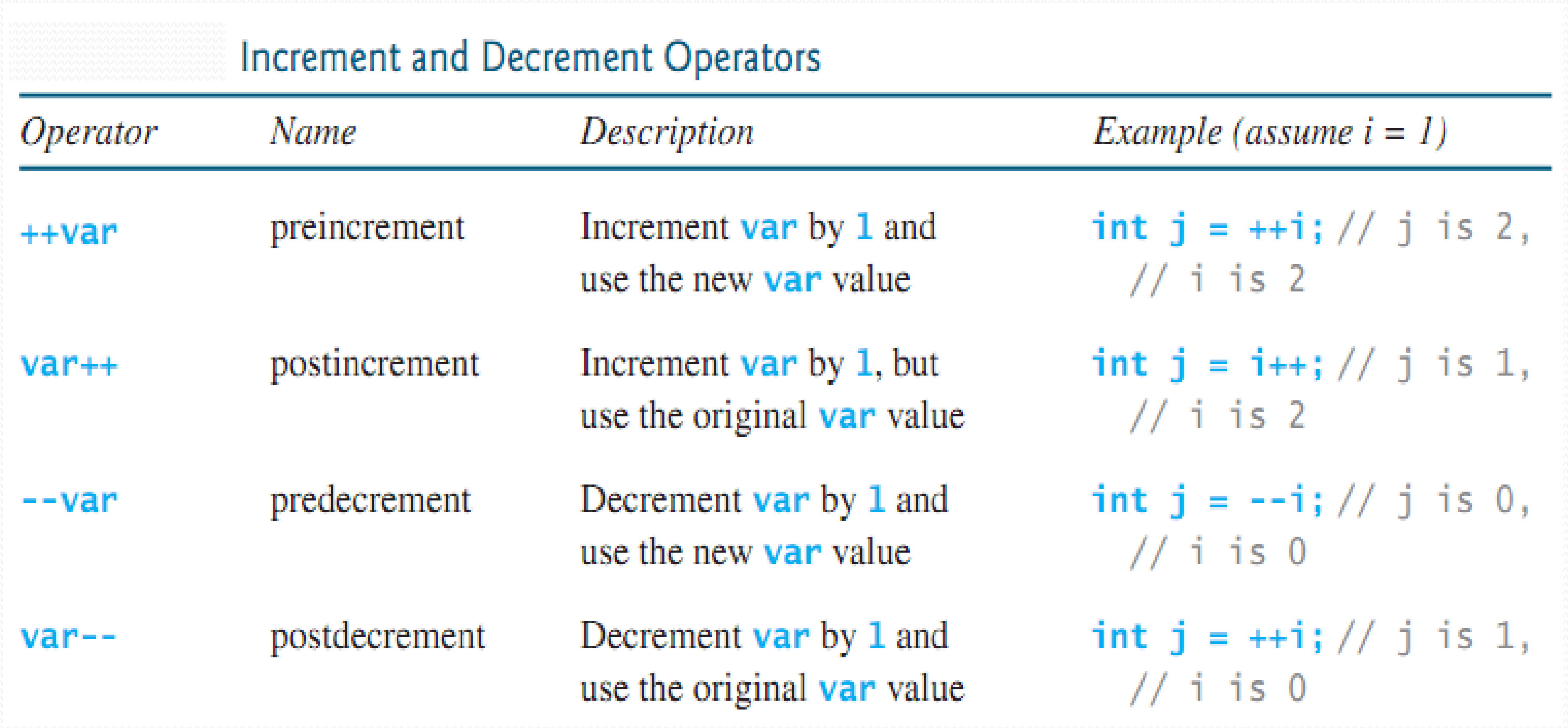
```
public class FahrenheitToCelsius {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Enter a degree in Fahrenheit: 
");
        double fahrenheit = input.nextDouble();
        // Convert Fahrenheit to Celsius
        double celsius = (5.0 / 9) * (fahrenheit - 32);
        System.out.println("Fahrenheit " + fahrenheit
        + " is " + celsius + " in Celsius");
    }
}
```

Shorthand Operators


Shorthand 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`

Shorthand Operators



Increment and Decrement Operators

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

Shorthand Operators

- Example 1:

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

- Example 2:

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

- Example 3:

```
double x = 1.0;
double y = 5.0;
double z = x— + (++y);
```

Numeric Type Conversions

- **Casting** is an operation that converts a value of one data type into a value of another data type.
  - Casting a variable of a type with a small range to a variable of a type with a larger range ➔ *widening* a type.
  - Casting a variable of a type with a large range to a variable of a type with a smaller range ➔ *narrowing* a type.
- *Widening* a type can be performed automatically without explicit casting. *Narrowing* a type must be performed explicitly.

Numeric Type Conversions

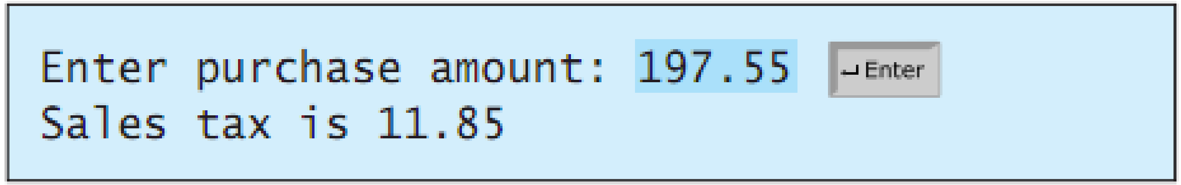
- Example:

```
System.out.println((int)1.7);
System.out.println((double)1 / 2);
System.out.println(1 / 2);
```

- Be careful when using casting. *Loss of information* might lead to *inaccurate results*.
- Casting does not change the variable being cast.

```
double d = 4.5;
int i = (int)d;
```

```java
public class SalesTax {
   public static void main(String[] args) {
      Scanner input = new Scanner(System.in);
      System.out.print("Enter purchase amount: ");
      double purchaseAmount = input.nextDouble();

      double tax = purchaseAmount * 0.06;
      System.out.println("Sales tax is " + (int) (tax *
                                    100) / 100.0);
      }
}
```



```
Enter purchase amount: 197.55 ↵Enter
Sales tax is 11.85
```

Problem: Computing Loan Payments

- The problem is to write a program that computes loan payments. The program lets the user enter the *interest rate*, *number of years*, and *loan amount*, and displays the **monthly** and **total payments**.

$$monthlyPayment = \frac{loanAmount \times monthlyInterestRate}{1 - \frac{1}{(1 + monthlyInterestRate)^{numberOfYears \times 12}}}$$

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

The steps in developing the program

1. Prompt the user to enter the *annual interest rate*, *number of years*, and *loan amount*.
2. Obtain the *monthly interest rate* from the annual interest rate.
3. Compute the **monthly payment** using the preceding formula.
4. Compute the **total payment,** which is the monthly payment multiplied by 12 and multiplied by the number of years.
5. Display the **monthly payment** and **total payment.**

```
public class ComputeLoan {
   public static void main(String[] args) {
      // Create a Scanner
      Scanner input = new Scanner(System.in);
      // Enter yearly interest rate
      System.out.print("Enter yearly interest rate, for 
                              example 8.25: ");
      double annualInterestRate = input.nextDouble();
      // Obtain monthly interest rate
      double monthlyInterestRate = annualInterestRate / 1200;

      // Enter number of years
      System.out.print("Enter number of years as an integer, 
                              for example 5: ");
      int numberOfYears = input.nextInt();
```

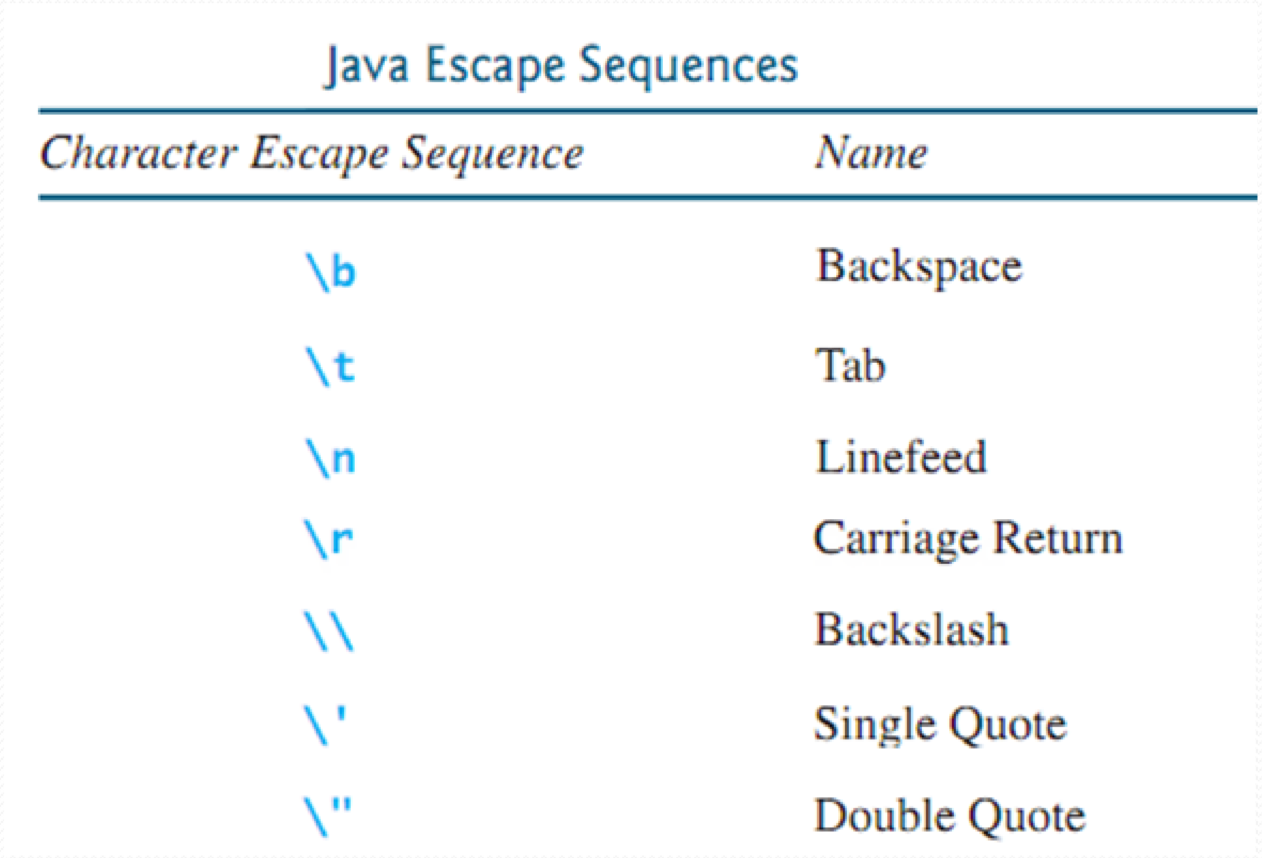
```
        // Enter loan amount
        System.out.print("Enter loan amount, for example
                     120000.95: ");
        double loanAmount = input.nextDouble();
        // Calculate payment
        double monthlyPayment = loanAmount *
              monthlyInterestRate / (1 - 1 / Math.pow(1 +
              monthlyInterestRate, numberOfYears * 12));
        double totalPayment = monthlyPayment *
                                   numberOfYears * 12;


        // Display results
        System.out.println("The monthly payment is " +
              (int) (monthlyPayment * 100) / 100.0);
        System.out.println("The total payment is " + (int)
              (totalPayment * 100)/ 100.0);
        }
}
```

Character Data Type and Operations

- The **character data type**, char, is used to represent a single character.
- A **character literal** is enclosed in single quotation marks.

```
char letter = 'A';
char numChar = '4';
```

Escape Sequences for Special Characters



Java Escape Sequences

Character Escape Sequence	*Name*
`\b`	Backspace
`\t`	Tab
`\n`	Linefeed
`\r`	Carriage Return
`\\`	Backslash
`\'`	Single Quote
`\"`	Double Quote

Casting between char and Numeric Types

- When an **integer** is cast into a **char**, only its lower 16 bits of data are used; the other part is ignored.

```
char ch = (char)0XAB0041;
// the lower 16 bits hex code 0041 is assigned to ch
System.out.println(ch); // ch is character A
```

- When a floating-point value is cast into a **char**, the floating-point value is first cast into an **int**, which is then cast into a **char**.

```
char ch = (char)65.25; // decimal 65 is assigned to ch
System.out.println(ch); // ch is character A
```

- When a **char** is cast into a numeric type, the character’s Unicode is cast into the specified numeric type.

```
int i = (int)'A';//the Unicode of character A is assigned to i
System.out.println(i); // i is 65
```

Casting between char and Numeric Types

- Implicit casting can be used if the result of a casting fits into the target variable. Otherwise, **explicit** casting **must** be used.
  - The Unicode of 'a' is 97, which is within the range of a byte, these implicit castings are fine: `byte b = 'a';`
  - But the following casting is incorrect, because the Unicode **uFFF4** cannot fit into a byte: `byte b = '\uFFF4';`
  - To force assignment, use explicit casting, as follows: `byte b = (byte)'\uFFF4';`

Problem: Counting Monetary Units

- Suppose you want to develop a program that classifies a given amount of money into smaller monetary units. The program lets the user enter an **amount** as a **double** value representing a total in dollars and cents, and outputs a report listing the monetary equivalent in dollars, quarters, dimes, nickels, and pennies.

The steps

1. Prompt the user to enter the amount as a decimal number, such as **11.56**.
2. Convert the amount (e.g., **11.56**) into cents (**1156**).
3. Divide the cents by **100** to find the number of *dollars*. Obtain the remaining cents using the cents remainder **100**.
4. Divide the remaining cents by **25** to find the number of *quarters*. Obtain the remaining cents using the remaining cents remainder **25**.
5. Divide the remaining cents by **10** to find the number of *dimes*. Obtain the remaining cents using the remaining cents remainder **1**0.
6. Divide the remaining cents by **5** to find the number of *nickels*. Obtain the remaining cents using the remaining cents remainder **5**.
7. The remaining cents are the *pennies*.
8. Display the result.

```
public class ComputeChange {
   public static void main(String[] args) {
      // Create a Scanner
      Scanner input = new Scanner(System.in);
      // Receive the amount
      System.out.print("Enter an amount in double, for
                                    example 11.56: ");
      double amount = input.nextDouble();
      int remainingAmount = (int) (amount * 100);

      // Find the number of one dollars
      int numberOfOneDollars = remainingAmount / 100;
      remainingAmount = remainingAmount % 100;
      // Find the number of quarters in the remaining amoun
      int numberOfQuarters = remainingAmount / 25;
      remainingAmount = remainingAmount % 25;
```

```
        //Find the number of dimes in the remaining amount
        remainingAmount = remainingAmount % 10;
        int numberOfDimes = remainingAmount / 10;
        //Find the number of nickels in the remaining amount
        int numberOfNickels = remainingAmount / 5;
        remainingAmount = remainingAmount % 5;
        // Find the number of pennies in the remaining amount
        int numberOfPennies = remainingAmount;
        // Display results
        System.out.println("Your amount " + amount + "
consists of \n" +
                "\t" + numberOfOneDollars + " dollars\n" +
                "\t" + numberOfQuarters + " quarters\n" +
                "\t" + numberOfDimes + " dimes\n" +
                "\t" + numberOfNickels + " nickels\n" +
                "\t" + numberOfPennies + " pennies");
    }
}
```

The String Type

- To represent a string of characters, use the data type called **String**.

  String message = "Welcome to Java";

- **String** is actually a predefined class in the Java library just like the classes **System**, and **Scanner**.
- The **String** type is not a primitive type. It is known as a *reference type*.

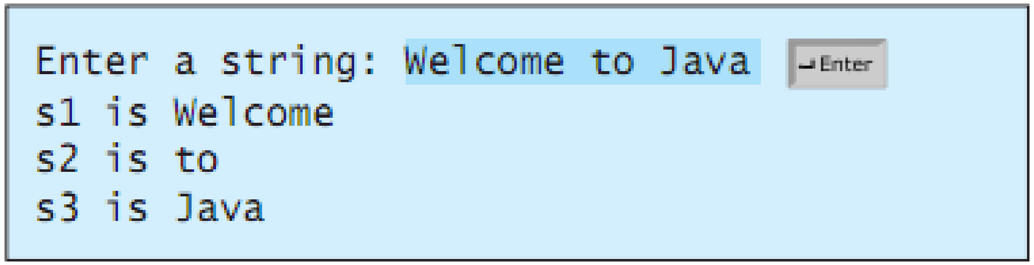
The String Type

- The plus sign (+) is the concatenation operator if **one** of the operands is a *string*.
- If one of the operands is a *nonstring* (e.g., a number), it is converted into a string and concatenated with the other string.

```
String message = "Welcome " + "to " + "Java";
String s = "Chapter" + 2;
String s1 = "Supplement" + 'B';
System.out.println("i + j is " + i + j);
System.out.println("i + j is " + (i + j));
```

Read strings with next() method

```java
public class ReadingStrings {
   public static void main(String[] args) {
      Scanner input = new Scanner(System.in);
      System.out.print("Enter three strings: ");
      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);
   }
}
```



```
Enter a string: Welcome to Java
s1 is Welcome
s2 is to
s3 is Java
```

Read strings with nextLine()

```
public class NextLineMethod {
   public static void main(String[] args) {
       Scanner input = new Scanner(System.in);
       System.out.print("Enter a string: ");
       String s = input.nextLine();
       System.out.println("The string entered is " + s);
}
}
```

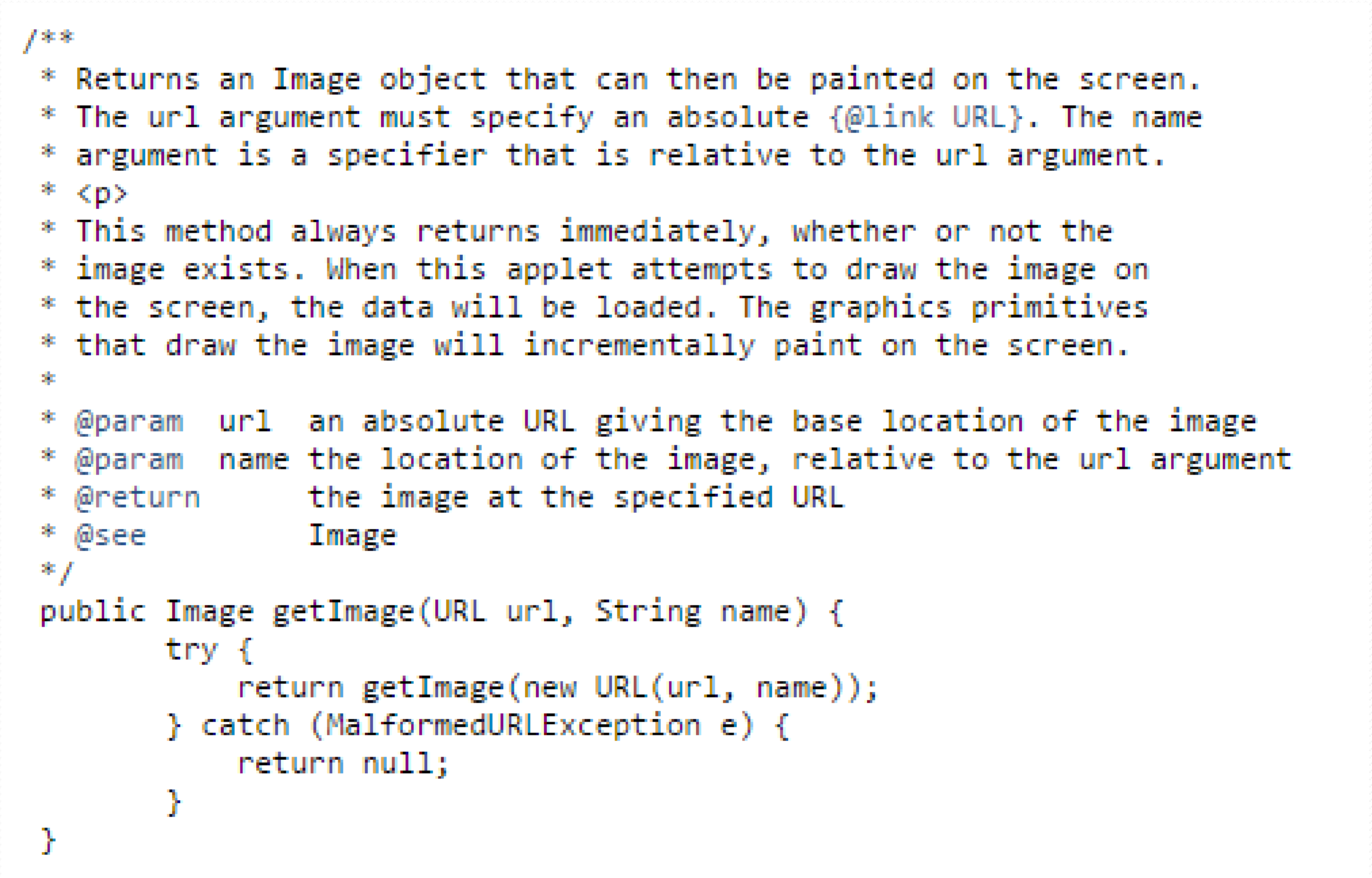
Programming Style and Documentation

- **Programming style** deals with what programs look like.
  - A program can compile and run properly even if written on only one line, but writing it all on one line would be bad programming style because it would be hard to read.
- **Documentation** is the body of explanatory remarks and comments pertaining to a program.
- **Programming style** and **documentation** are as important as coding. Good programming style and appropriate documentation reduce the chance of errors and make programs easy to read.

Appropriate Comments and Comment Styles

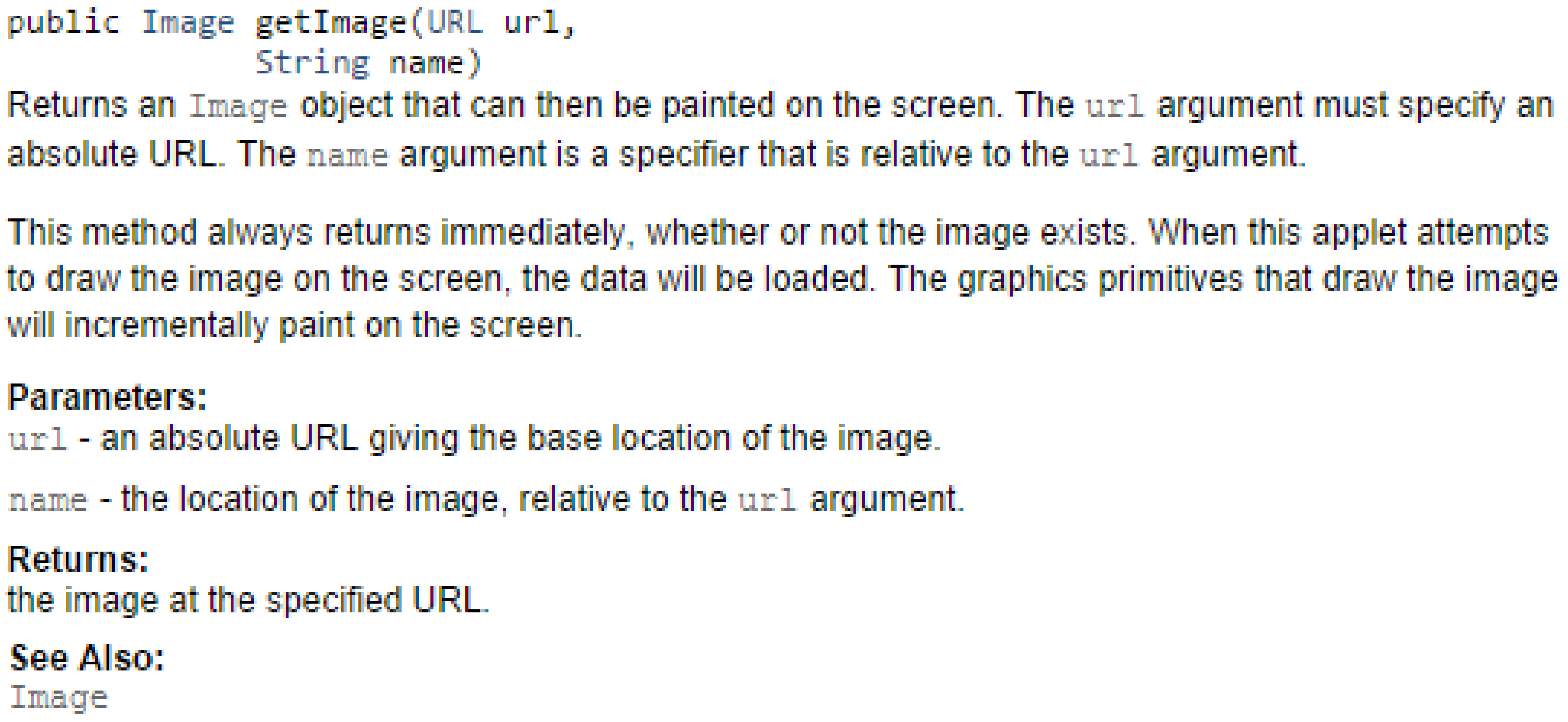
- Line comment //
- Block comment /* and */
- **Javadoc comments**, it begins with /** and end with */.
  - Use javadoc comments (/** ... */) for commenting on an *entire class* or an *entire method*.
  - Must precede the class or the method header in order to be extracted in a javadoc HTML file.

Javadoc comments



```
/**
 * Returns an Image object that can then be painted on the screen.
 * The url argument must specify an absolute {@link URL}. The name
 * argument is a specifier that is relative to the url argument.
 * <p>
 * This method always returns immediately, whether or not the
 * image exists. When this applet attempts to draw the image on
 * the screen, the data will be loaded. The graphics primitives
 * that draw the image will incrementally paint on the screen.
 *
 * @param  url  an absolute URL giving the base location of the image
 * @param  name the location of the image, relative to the url argument
 * @return      the image at the specified URL
 * @see         Image
 */
 public Image getImage(URL url, String name) {
        try {
            return getImage(new URL(url, name));
        } catch (MalformedURLException e) {
            return null;
        }
 }
```

Javadoc comments



```
public Image getImage(URL url,
                      String name)
```

Returns an `Image` object that can then be painted on the screen. The `url` argument must specify an absolute URL. The `name` argument is a specifier that is relative to the `url` argument.

This method always returns immediately, whether or not the image exists. When this applet attempts to draw the image on the screen, the data will be loaded. The graphics primitives that draw the image will incrementally paint on the screen.

Parameters:
`url` - an absolute URL giving the base location of the image.

`name` - the location of the image, relative to the `url` argument.

Returns:
the image at the specified URL.

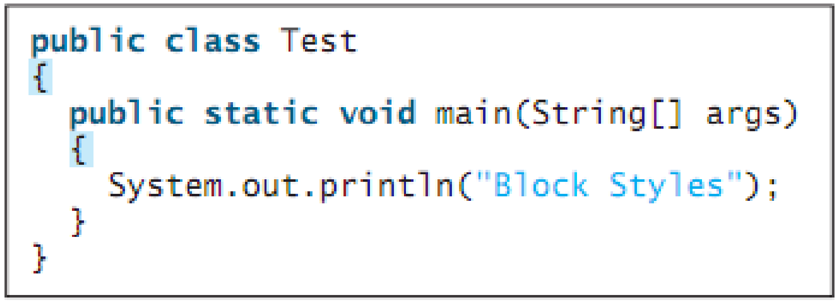
See Also:
`Image`

Naming Conventions

- Use **lowercase** for **variables** and **methods**.
- If a name consists of several words, concatenate them into one, making the first word lowercase and capitalizing the first letter of each subsequent word:
  - **Example: radius, area, showInputDialog**.
- Capitalize the first letter of each word in a class name:
  - **Example: ComputeArea**, **Math**.
- Capitalize every letter in a constant, and use underscores between words:
  - **Example: PI** and **MAX_VALUE**.

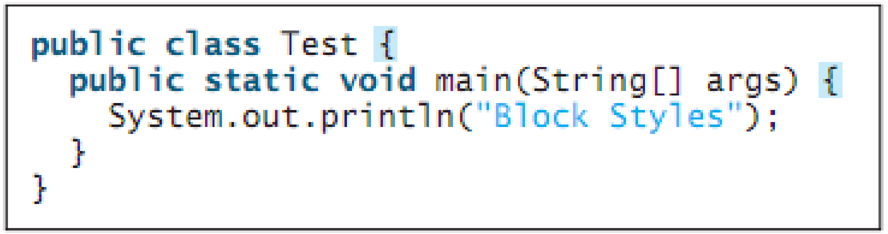
Block Styles

- A block is a group of statements surrounded by braces. There are two popular styles, **next-line** style and **end-of-line** style.



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

Next-line style



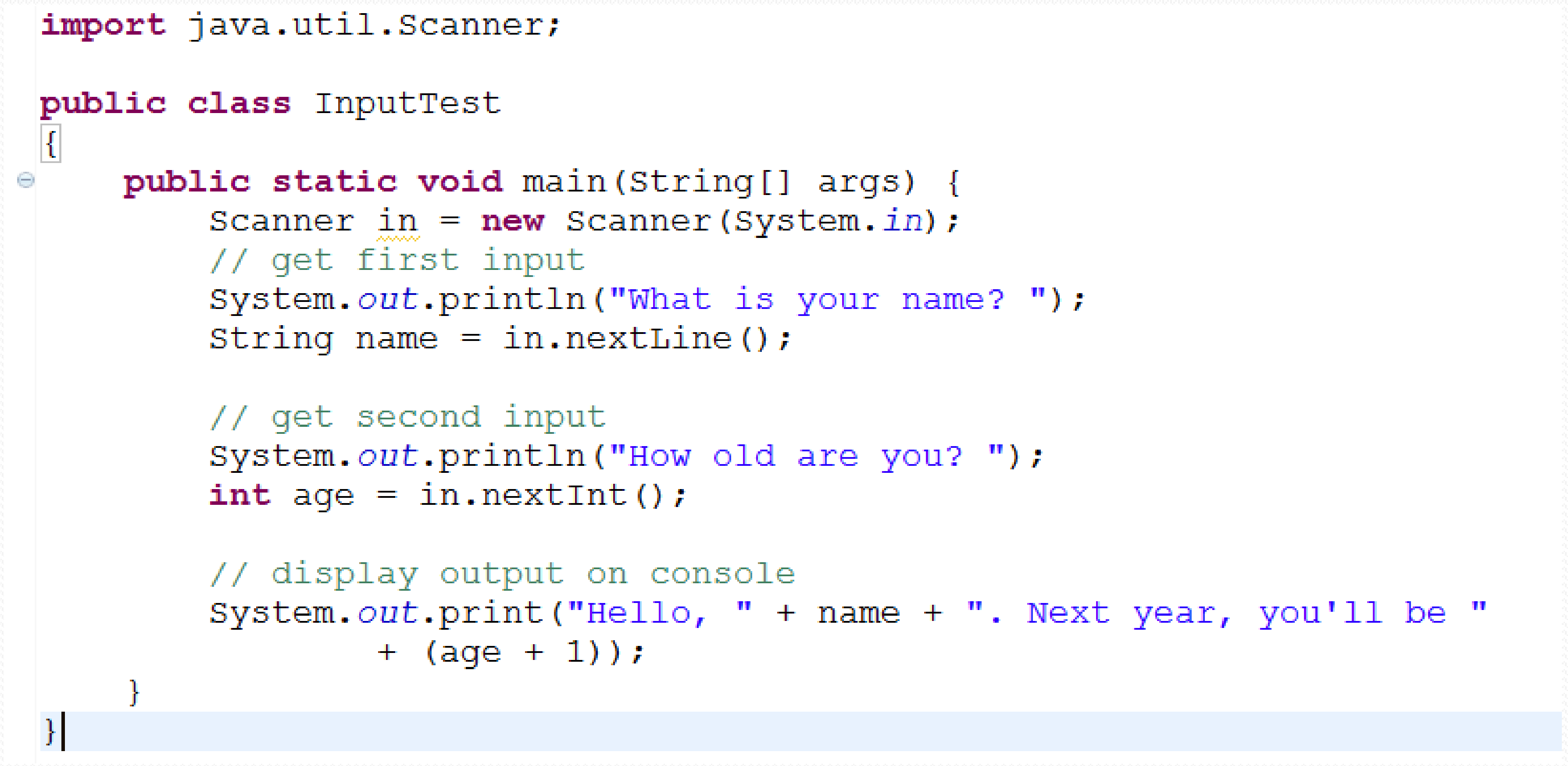
```
public class Test {
  public static void main(String[] args) {
    System.out.println("Block Styles");
  }
}
```

End-of-line style

Remember: **Input and Output**

- Reading Input: import java.util.Scanner;
- To reading console input:
  - Scanner in = new Scanner (System.in);
- The nextLine method reads a line of input:
  - System.out.print(“What is your name?”);
  - String name = in.nextLine();
- To read an integer, us the nextInt method:
  - int age = in.nextInt();
- The nextDouble method reads the next floating-point number.

Sample



```
import java.util.Scanner;

public class InputTest
{
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        // get first input
        System.out.println("What is your name? ");
        String name = in.nextLine();

        // get second input
        System.out.println("How old are you? ");
        int age = in.nextInt();

        // display output on console
        System.out.print("Hello, " + name + ". Next year, you'll be "
                + (age + 1));
    }
}
```

Exercises

1. **Ex1**: Viết chương trình hiển thị ra thông báo: “Please input one integer and one floating-point number”. Sau đó người dùng sẽ nhập vào từ bàn phím một số nguyên x và một số thập phân y. Hiển thị ra màn hình: “Input values are” + x + “and ” + y + “their product is ” + x*y
2. **Ex2**: Có phương trình như sau:

$$y = 4(x-3) + 20$$

Viết chương trình nhập x từ bàn phím và tính giá trị của y, với x và y là kiểu số nguyên.

Exercises (cont)

3. **<u>Ex3</u>**: Một mile bằng 1.609km. Viết chương trình cho người dùng nhập 1 số, sau đó chuyển đổi số đó sang km (ví dụ, người dùng nhập 5, thì thông báo ra màn hình: 5 miles = 8.045 km.
4. **<u>Ex4</u>**: Nhập vào bán kính hình tròn. Tính chu vi và diện tích hình tròn.
5. **<u>Ex5</u>**: Nhập vào tổng số giây. Hãy chuyển đổi sang giờ, phút, giây và in ra theo dạng h:m:s.
   Ví dụ: 1999 giây => 5:3:19

Exercises (cont)

6. Nhập vào độ cao h của một vật rơi tự do. Tính thời gian và vận tốc của vật lúc chạm đất theo công thức sau: Thời gian t = sqrt(2*h/g) và vận tốc v = gt.
7. Nhập vào các số thực xA, yA, xB, yB là hoành độ và tung độ của 2 điểm A, B. Tính khoảng cách d giữa 2 điểm theo công thức d = sqrt((xA-xB)^2 + (yA-yB)^2)

Reference

- **Introduction to Java Programming** 8th , Y. Daniel Liang.
- **Head First Java,** 2nd, Kathy Sierra & Bert Bates
- **Core Java Volume I Fundamentals**, 8th, Cay S.Horstmann & Gary Cornell