Module 1

PPT-1

Programming with JAVA

Jagjit Bhatia

PG Dept of Computer Science

Module-I

- Introduction to JAVA, History of JAVA, Features of JAVA, Versions and IDE of JAVA,Platform Independence & Cross Platform Computing,Java literals , variables , Datatypes,Operators, Control statements, Arrays in Java

- What is java to JAVA
Characteristics of Java
History of java, versions of java

Introduction to Java

- What Is Java?
- Getting Started With Java Programming
  - Create, Compile and Running a Java Application

History

- James Gosling and Sun Microsystems
- Oak
- Java, May 20, 1995, Sun World
- HotJava
  - The first Java-enabled Web browser
- JDK Evolutions
- J2SE, J2ME, and J2EE (not mentioned in the book, but could discuss here optionally)

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

JDK Versions

- JDK 1.02 (1995)
- JDK 1.1 (1996)
- Java 2 SDK v 1.2 (a.k.a JDK 1.2, 1998)
- Java 2 SDK v 1.3 (a.k.a JDK 1.3, 2000)
- Java 2 SDK v 1.4 (a.k.a JDK 1.4, 2002)

JDK Editions

- Java Standard Edition (J2SE)
  - J2SE can be used to develop client-side standalone applications or applets.
- Java Enterprise Edition (J2EE)
  - J2EE can be used to develop server-side applications such as Java servlets and Java ServerPages.
- Java Micro Edition (J2ME).
  - J2ME can be used to develop applications for mobile devices such as cell phones.

This book uses J2SE to introduce Java programming.

Java IDE Tools

- Forte by Sun MicroSystems
- Borland JBuilder
- Microsoft Visual J++
- WebGain Café
- IBM Visual Age for Java
- Netbeans
- Eclipse

Getting Started with Java Programming

- A Simple Java Application
- Compiling Programs
- Executing Applications

A Simple Application

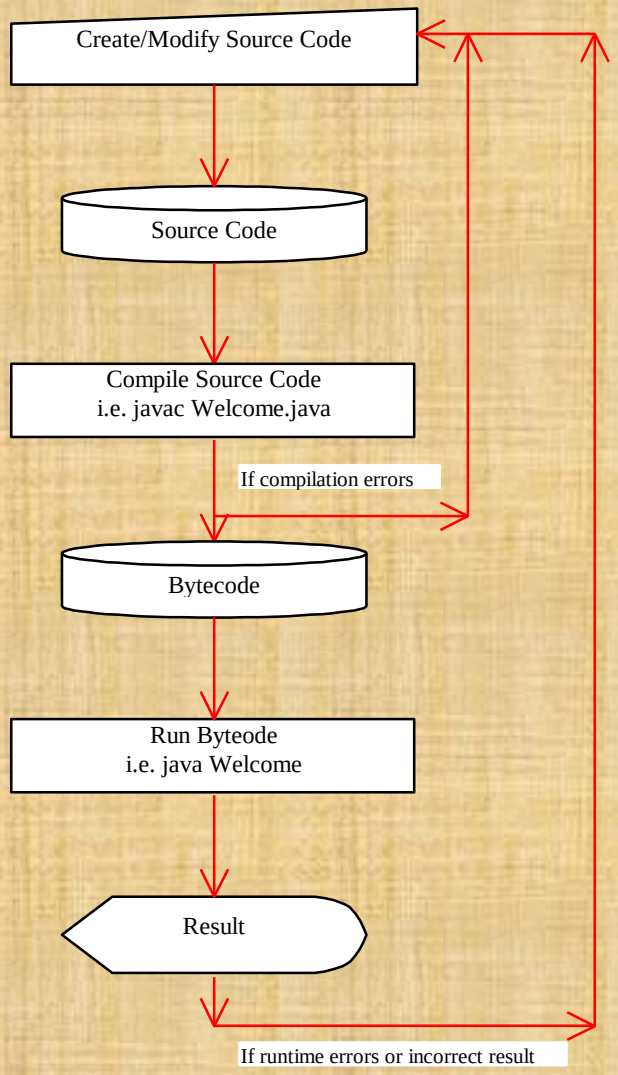
Example 1.1

```
//This application program prints Welcome
//to Java!
package chapter1;

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

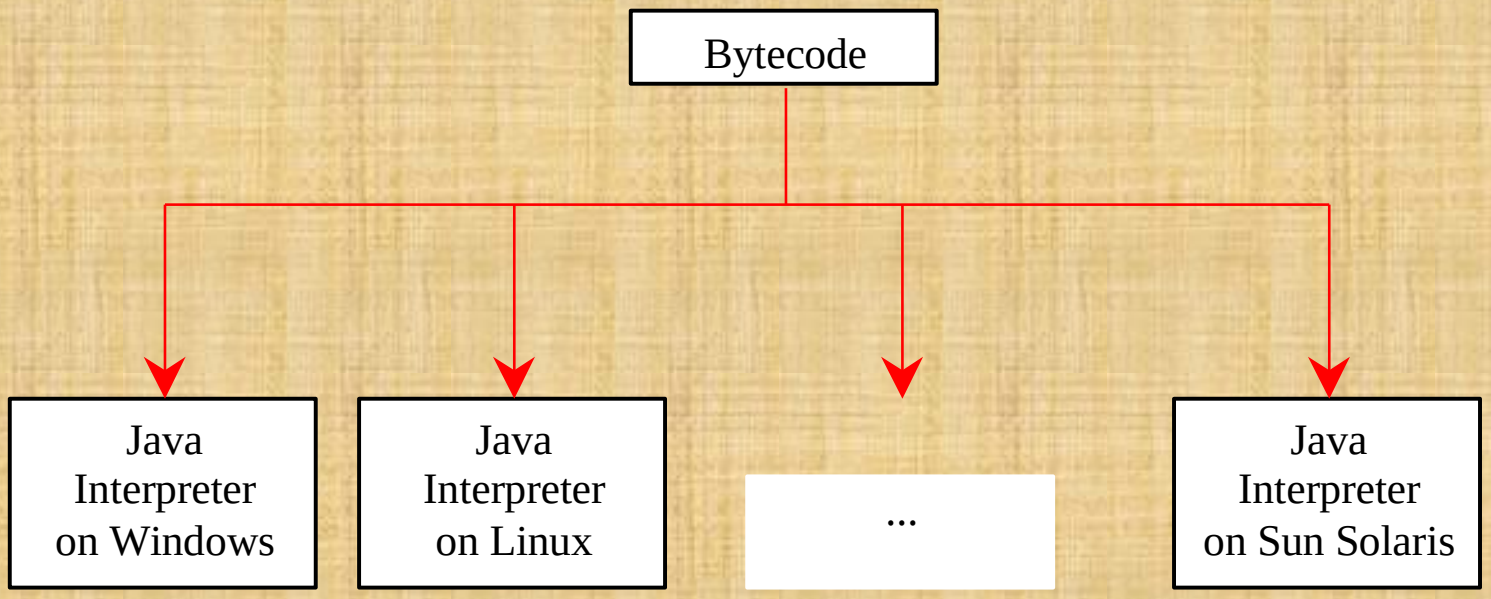
Creating and Compiling Programs

- On command line
  - `javac file.java`



Executing Applications

- On command line
  - `java classname`



Example

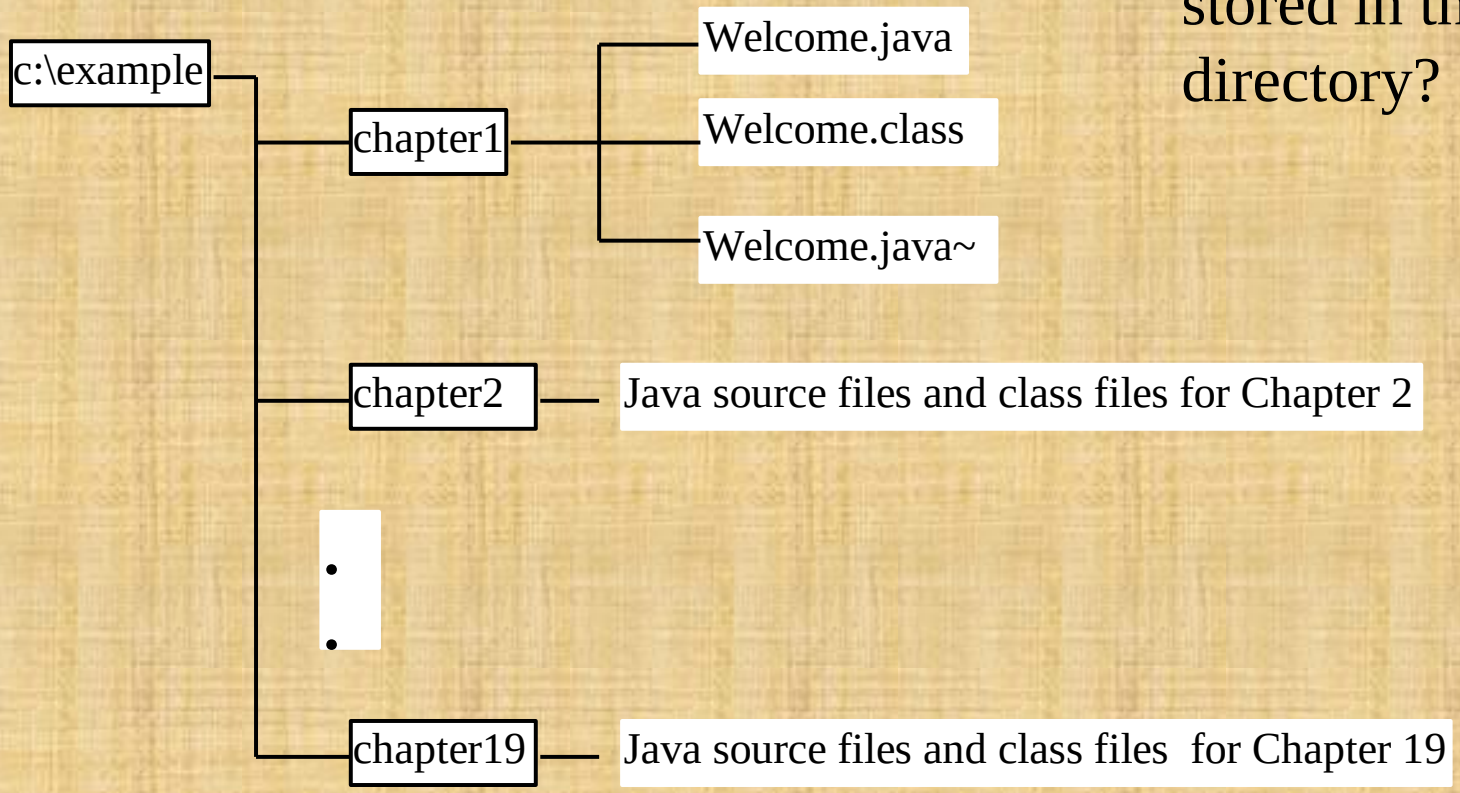
```
javac Welcome.java

java Welcome

output:...
```

Compiling and Running a Program

Where are the files stored in the directory?



Anatomy of a Java Program

- Comments
- Package
- Reserved words
- Modifiers
- Statements
- Blocks
- Classes
- Methods
- The main method

Comments

In Java, comments are preceded by two slashes (//) in a line, or enclosed between /* and */ in one or multiple lines. When the compiler sees //, it ignores all text after // in the same line. When it sees /*, it scans for the next */ and ignores any text between /* and */.

Package

The second line in the program (package chapter1;) specifies a package name, chapter1, for the class Welcome. Forte compiles the source code in Welcome.java, generates Welcome.class, and stores Welcome.class in the chapter1 folder.

Reserved Words

Reserved words or *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. Other reserved words are public, static, and void. Their use will be introduced later in the book.

Modifiers

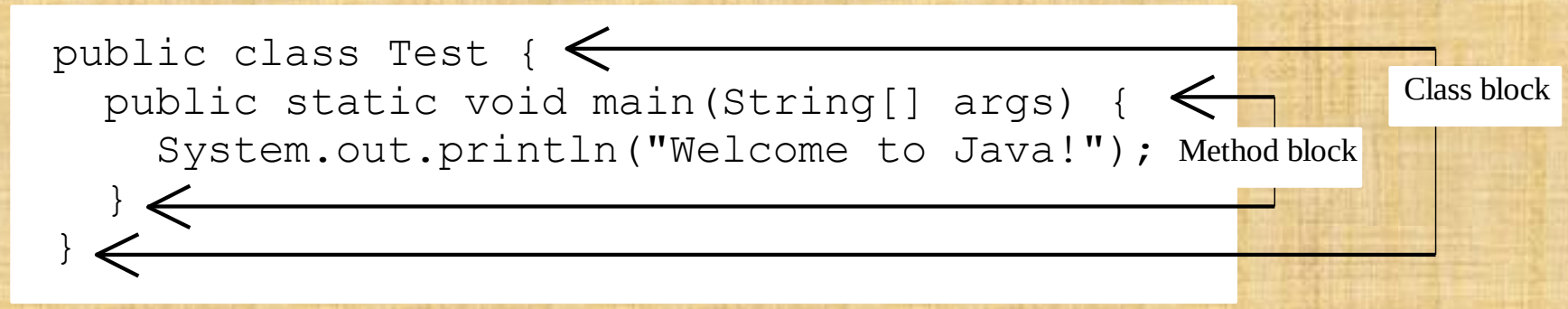
Java uses certain reserved words called *modifiers* that specify the properties of the data, methods, and classes and how they can be used. Examples of modifiers are public and static. Other modifiers are private, final, abstract, and protected. A public datum, method, or class can be accessed by other programs. A private datum or method cannot be accessed by other programs. Modifiers are discussed in Chapter 6, "Objects and Classes."

Statements

A *statement* represents an action or a sequence of actions. The statement System.out.println("Welcome to Java!") in the program in Example 1.1 is a statement to display the greeting "Welcome to Java!" Every statement in Java ends with a semicolon (;).

Blocks

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



main Method

The main method provides the control of program flow. The Java interpreter executes the application by invoking the main method.

The main method looks like this:

```
public static void main(String[] args) {
  // Statements;
}
```

Identifiers

- An identifier is a sequence of characters that consist 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. (See Appendix A, “Java Keywords,” for a list of reserved words).
- An identifier cannot be `true`, `false`, or `null`.
- An identifier can be of any length.

Variables

```
// Compute the first area
radius = 1.0;
area = radius * radius * 3.14159;
System.out.println("The area is " +
  area + " for radius "+radius);

// Compute the second area
radius = 2.0;
area = radius * radius * 3.14159;
System.out.println("The area is " +
  area + " for radius "+radius);
```

Declaring Variables

```
int x;              // Declare x to be an
                    // integer variable;
double radius;      // Declare radius to
                    // be a double variable;
char a;             // Declare a to be a
                    // character variable;
```

Assignment Statements

```
x = 1;              // Assign 1 to x;

radius = 1.0;       // Assign 1.0 to radius;

a = 'A';            // Assign 'A' to a;
```

Declaring and Initializing in One Step

- `int x = 1;`
- `double d = 1.4;`

Constants

```
final datatype CONSTANTNAME = VALUE;

final double PI = 3.14159;
final int SIZE = 3;
```

Numerical Data Types

Name	Range	Storage Size
byte	-2^7 (-128) to 2^7-1 (127)	8-bit signed
short	-2^{15} (-32768) to $2^{15}-1$ (32767)	16-bit signed
int	-2^{31} (-2147483648) to $2^{31}-1$ (2147483647)	32-bit signed
long	-2^{63} to $2^{63}-1$ (i.e., -9223372036854775808 to 9223372036854775807)	64-bit signed
float	Negative range: -3.4028235E+38 to -1.4E-45 Positive range: 1.4E-45 to 3.4028235E+38	32-bit IEEE 754
double	Negative range: -1.7976931348623157E+308 to -4.9E-324 Positive range: 4.9E-324 to 1.7976931348623157E+308	64-bit IEEE 754

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 Division

+, -, *, /, and %

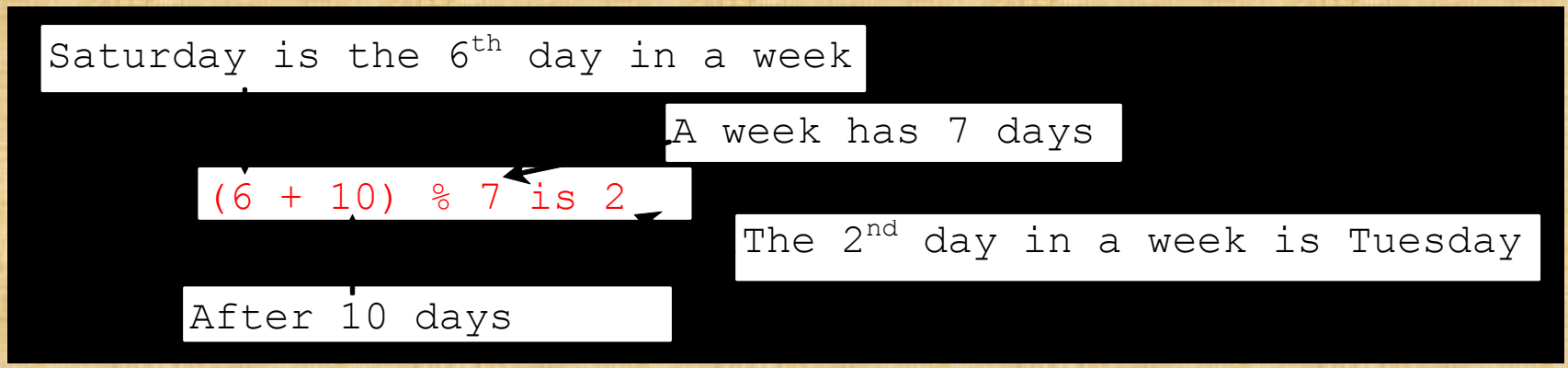
5 / 2 yields an integer 2.

5.0 / 2 yields a double value 2.5

5 % 2 yields 1 (the remainder of the division)

Remainder Operator

Remainder is very useful in programming. For example, an even number % 2 is always 0 and an odd number % 2 is always 1. So you can use this property to determine whether a number is even or odd. Suppose today is Saturday and you and your friends are going to meet in 10 days. What day is in 10 days? You can find that day is Tuesday using the following expression:



Number Literals

A *literal* is a constant value that appears directly in the program. For example, 34, 1,000,000, and 5.0 are literals in the following statements:

```
int i = 34;

long x = 1000000;

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.

An integer literal is assumed to be of the int type, whose value is between -2^{31} (-2147483648) to $2^{31}-1$ (2147483647). 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).

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, 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.

Scientific Notation

Floating-point literals can also be specified in scientific notation, for example, 1.23456e+2, same as 1.23456e2, is equivalent to 123.456, and 1.23456e-2 is equivalent to 0.0123456. E (or e) represents an exponent and it can be either in lowercase or uppercase.

Numeric Type Conversion

Consider the following statements:

```
byte i = 100;
long k = i * 3 + 4;
double d = i * 3.1 + k / 2;
```

Conversion Rules

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

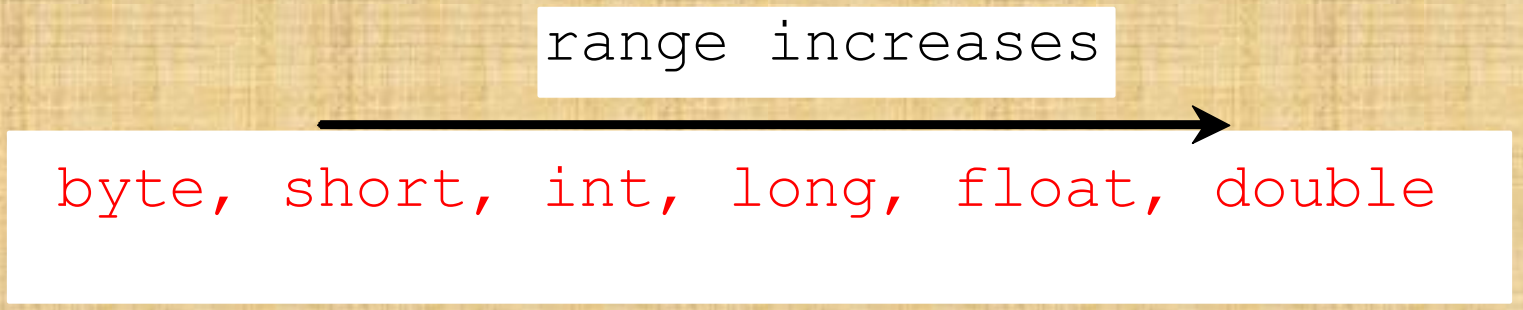
1. If one of the operands is double, the other is converted into double.
2. Otherwise, if one of the operands is float, the other is converted into float.
3. Otherwise, if one of the operands is long, the other is converted into long.
4. Otherwise, both operands are converted into int.

Type Casting

```
Implicit casting
  double d = 3; (type widening)

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

What is wrong? int x = 5 / 2.0;



Character Data Type

Four hexadecimal digits.

```
char letter = 'A'; (ASCII)
char numChar = '4'; (ASCII)
char letter = '\u0041'; (Unicode)
char numChar = '\u0034'; (Unicode)
```

NOTE: The increment and decrement operators can also be used on char variables to get the next or preceding Unicode character. For example, the following statements display character b.

```
char ch = 'a';
System.out.println(++ch);
```

Escape Sequences for Special Characters

Description	*Escape Sequence*	*Unicode*
Backspace	`\b`	`\u0008`
Tab	`\t`	`\u0009`
Linefeed	`\n`	`\u000A`
Carriage return	`\r`	`\u000D`
Backslash	`\\`	`\u005C`
Single Quote	`\'`	`\u0027`
Double Quote	`\"`	`\u0022`

Operators

- Group of Operators
- Arithmetic Operators
- Assignment Operator
- Order of Precedence
- Increment/Decrement Operators
- Relational Operators
- Logical Operators

Operators

- Operators are special symbols used for:
  - mathematical functions
  - assignment statements
  - logical comparisons
- Examples of operators:
  - 3 + 5 // uses + operator
  - 14 + 5 – 4 * (5 – 3) // uses +, -, * operators
- Expressions: can be combinations of variables and operators that result in a value

Groups of Operators

- There are 5 different groups of operators:
  - Arithmetic Operators
  - Assignment Operator
  - Increment / Decrement Operators
  - Relational Operators
  - Logical Operators

Java Arithmetic Operators

Addition	+
Subtraction	–
Multiplication	*
Division	/
Remainder (modulus)	%

Arithmetic Operators

- The following table summarizes the arithmetic operators available in Java.

Operation	Java Operator	Example	Value (x = 10, y = 7, z = 2.5)
Addition	+	x + y	17
Subtraction	-	x - y	3
Multiplication	*	x * y	70
Division	/	x / y	1
		x / z	4.0
Modulo division (remainder)	%	x % y	3

This is an integer division where the fractional part is truncated.

Example

Example of division issues:

10 / 3 gives 3

10.0 / 3 gives 3.33333

As we can see,

- if we divide two integers we get an integer result.
- if one or both operands is a floating-point value we get a floating-point result.

Modulus

- Generates the remainder when you divide two integer values.

5%3 gives 2 5%4 gives 1

5%5 gives 0 5%10 gives 5

- Modulus operator is most commonly used with integer operands. If we attempt to use the modulus operator on floating-point values we will garbage!

Order of Precedence

( ) evaluated first, inside-out

*, /, or % evaluated second, left-to-right

+, – evaluated last, left-to-right

Basic Assignment Operator

- We assign a value to a variable using the basic *assignment operator (=)*.
- Assignment operator stores a value in memory.
- The syntax is

```
leftSide = rightSide ;
```

Allways it is a *variable identifier.*

It is either a *literal* | a *variable identifier* | an *expression.*

Examples:

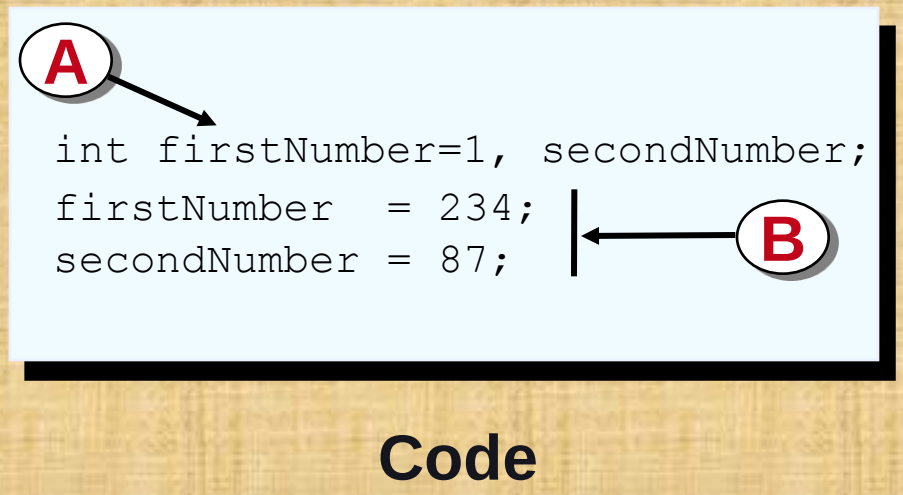
```
i = 1;
start = i;
sum = firstNumber + secondNumber;
avg = (one + two + three) / 3;
```

The Right Side of the Assignment Operator

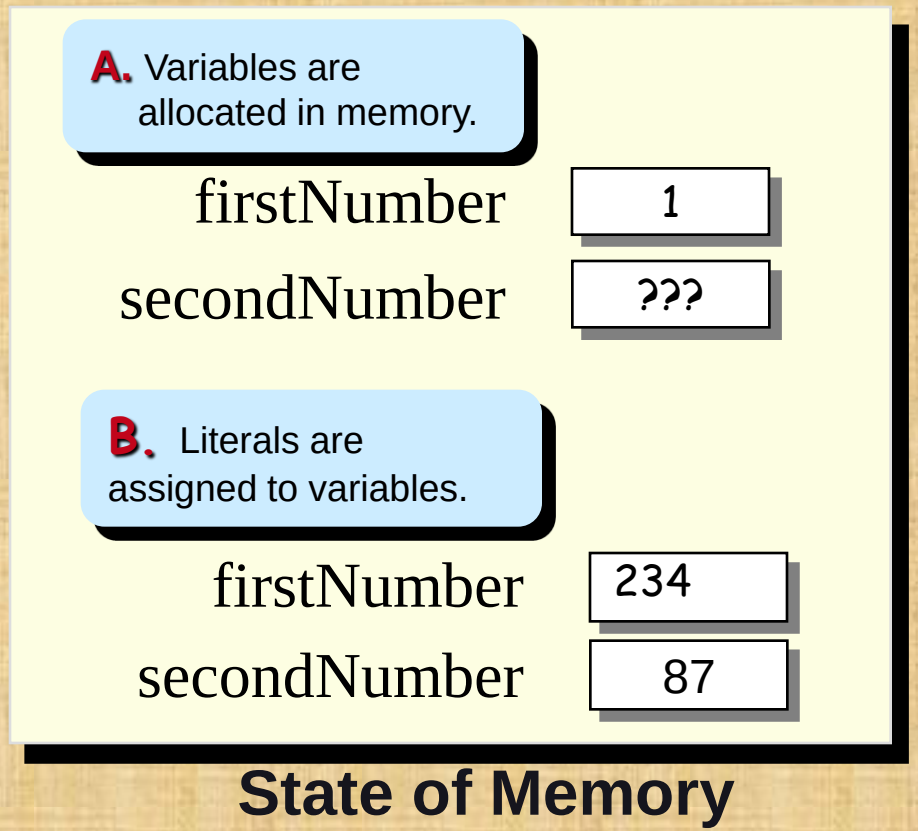
- The Java assignment operator assigns the value on the right side of the operator to the variable appearing on the left side of the operator.
- The right side may be either:
  - Literal: ex. `i = 1;`
  - Variable identifier: ex. `start = i;`
  - Expression: ex. `sum = first + second;`

Assigning Literals

- In this case, the literal is stored in the space memory allocated for the variable at the left side.



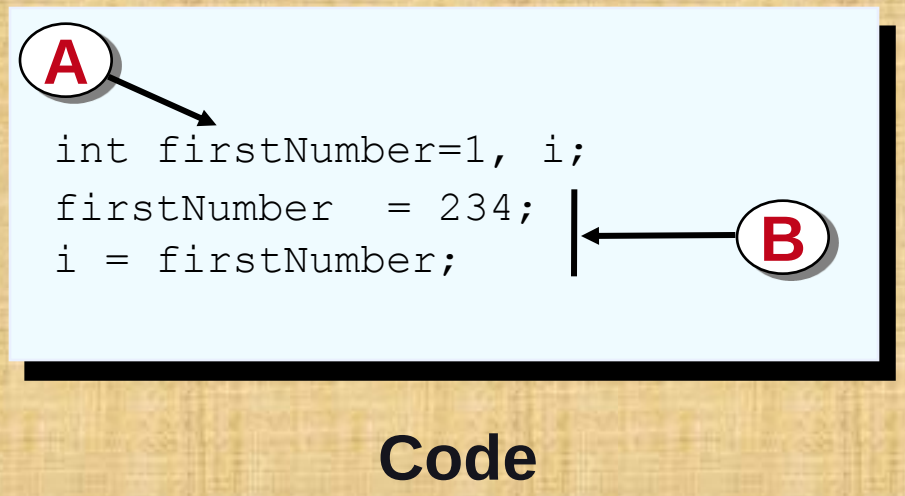
Code



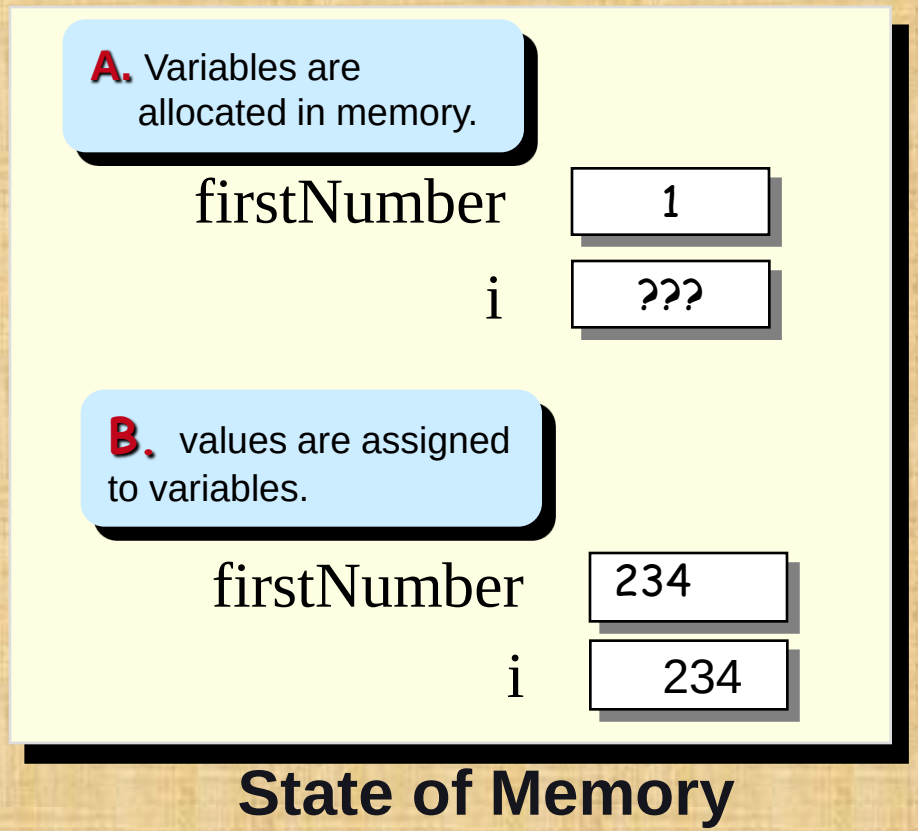
State of Memory

Assigning Variables

- In this case, the value of the variable at the right side is stored in the space memory allocated for the variable at the left side.



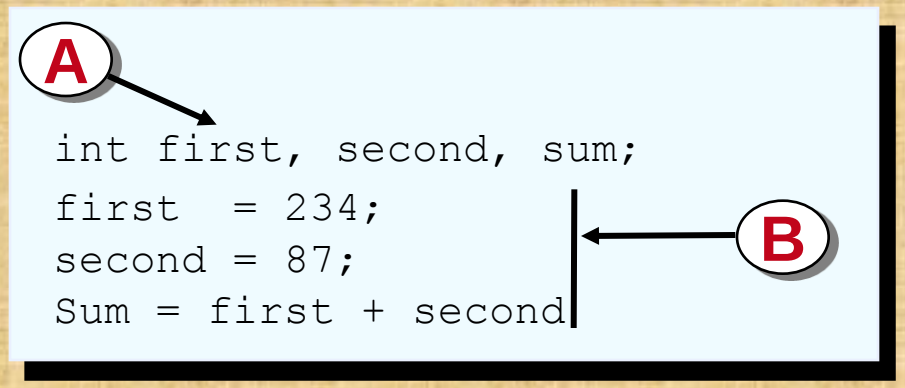
Code



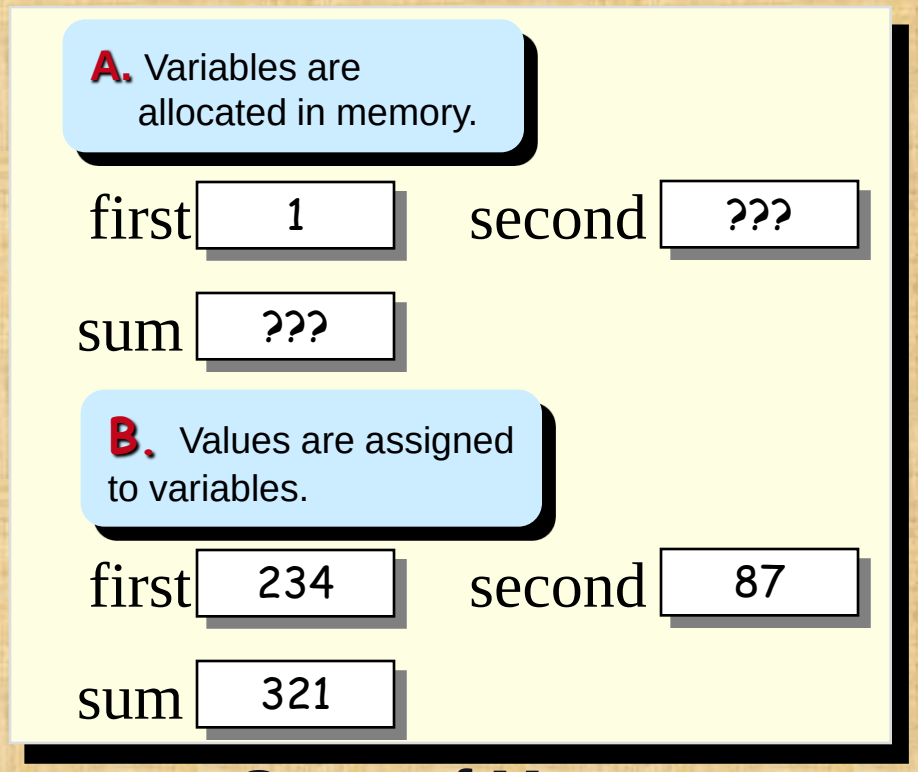
State of Memory

Assigning Expressions

- In this case, the result of the evaluation of the expression is stored in the space memory allocated for variable at the left side.



Code



State of Memory

Updating Data

```
int number;
number = 237;
number = 35;
```

A
B
C

Code

A. The variable is allocated in memory.

number ???

B. The value **237** is assigned to **number**.

number 237

C. The value **35** overwrites the previous value **237**.

number 35

State of Memory

Example: Sum of two integer

```
public class Sum {

 // main method
    public static void main( String args[] ){
        int a, b, sum;
        a = 20;
        b = 10;
        sum = a + b;
        System.out.println(a + " + " + b + "  = " + sum);

    } // end main

} // end class Sum
```

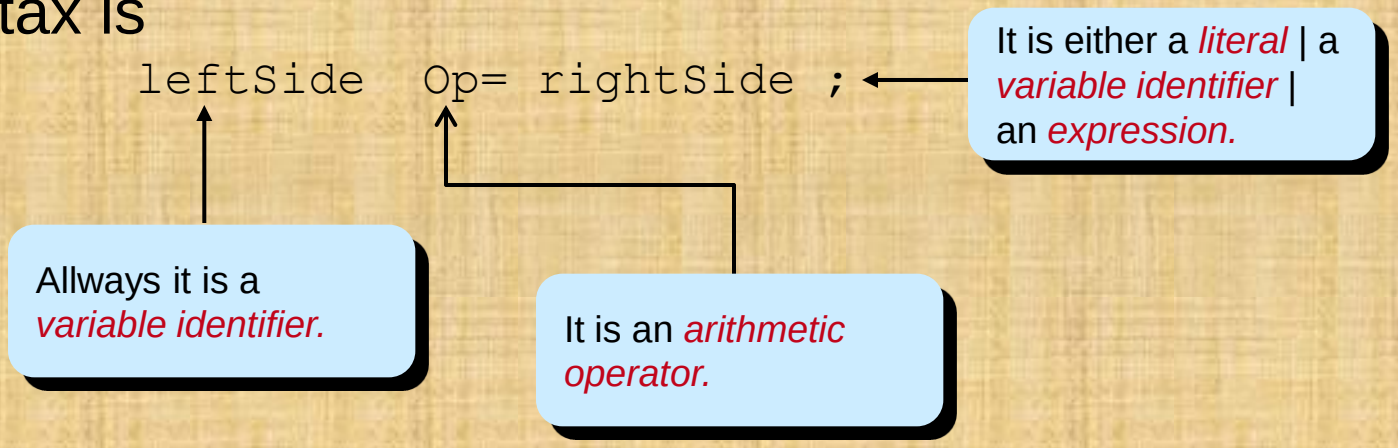
Arithmetic/Assignment Operators

Java allows combining arithmetic and assignment operators into a single operator:

Addition/assignment	**+=**
Subtraction/assignment	**–=**
Multiplication/assignment	***=**
Division/assignment	**/=**
Remainder/assignment	**%=**

Arithmetic/Assignment Operators

- The syntax is



- This is equivalent to:

```
leftSide = leftSide Op rightSide ;
```

- x%=5; ⇔ x = x % 5;
- x*=y+w*z; ⇔ x = x*(y+w*z);

Increment/Decrement Operators

Only use ++ or – – when a variable is being incremented/decremented as a statement by itself.

`x++;` is equivalent to `x = x+1;`

`x--;` is equivalent to `x = x-1;`

Increment and Decrement Operators, cont.

```
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;
```

Contd. PPT-2